JavaScript и jQuery

3-е издание

исчерпывающее руководство

Книга, которая должна быть под рукой

Включает описание плагина jQuery UI

O'REILLY®

Дэвид МакФарланд

МИРОВОЙ
КОМПЬЮТЕРНЫЙ
БЕСТСЕЛЛЕР

David Sawyer McFarland

JavaScript & jQuery

3rd edition

O'REILLY®

Дэвид МакФарланд

JavaScript и jQuery

3-е издание

O'REILLY

Москва
2015

УДК 004.43
ББК 32.973.26-018.1
М17

David McFarland
JAVASCRIPT & JQUERY: THE MISSING MANUAL, 3rd edition
2015 Authorized Russian translation of the English edition of Javascript & jQuery: The Missing manual, 3rd edition, ISBN 9781491947074
© 2014 Sawyer McFarland Media, Inc.

Макфарланд, Дэвид.

М17 JavaScript и jQuery: исчерпывающее руководство / Дэвид Макфарланд ; [пер. с англ. М. А. Райтмана]. — 3-е издание. — Москва : Эксмо, 2015. — 880 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-79119-4

JavaScript — основной инструмент веб-разработчиков, позволяющий делать интернет-страницы интерактивными. Перед вами — наиболее полное и великолепно структурированное руководство по JavaScript, которое позволит в совершенстве овладеть этим востребованным сейчас языком программирования. В книге уделено большое внимание библиотеке jQuery, в том числе самого современного плагина jQuery UI.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-699-79119-4

© Райтман М.А., перевод на русский язык, 2015
© Оформление. ООО «Издательство «Эксмо», 2015

ОГЛАВЛЕНИЕ

Недостающее признание 13
Об авторе 13
О творческом коллективе 13
Слова благодарности 14
О серии «Исчерпывающее руководство» 15

Введение 16
Что такое JavaScript 16
Немного истории 17
JavaScript повсюду 19
Что такое jQuery? 20
HTML: скелет-основа 21
Как работают HTML-элементы 23
CSS: Добавление стилей на веб-страницу 25
Анатомия стиля 26
Программы для верстки кода на языке JavaScript 29
Бесплатные программы 29
Коммерческие программы 31
Об этой книге 32
Подход к JavaScript, принятый в книге 33
Структура книги 34
Самое основное 35
Об ⇒ этих ⇒ стрелках 36
О ресурсах во Всемирной паутине 37
Файлы примеров 37

ЧАСТЬ I
ВВЕДЕНИЕ В JAVASCRIPT

Глава 1. Ваша первая программа на языке JavaScript 40
Введение в программирование 41
Что такое компьютерная программа 43
Добавление JavaScript на страницу 44
Внешние файлы JavaScript 47
Ваша первая программа на языке JavaScript 52
Публикация текста на веб-странице 55
Прикрепление внешнего файла JavaScript 57
Отслеживание ошибок 60
Консоль JavaScript в браузере Chrome 61
Консоль JavaScript в браузере Internet Explorer 64
Консоль JavaScript в браузере Firefox 65
Консоль JavaScript в браузере Safari 66

Глава 2. Грамматика языка JavaScript 68
Инструкции 68
Встроенные функции 69
Типы данных 70
Числа 70
Строки 71
Логический тип данных 73

Переменные 74
Создание переменной 75
Использование переменных 77
Работа с типами данных и переменными 79
Основные математические операции 79
Порядок операций 82
Объединение строк 83
Объединение чисел и строк 84
Изменение значений в переменных 86
Использование переменных для создания сообщений на практике 88
Запрос информации на практике 91
Массивы 93
Создание массива 95
Доступ к элементам массива 96
Добавление элементов в массив 98
Удаление элементов из массива 101
Публикация текста на веб-странице с помощью массивов на практике 103
Вкратце об объектах 107
Комментарии 110
Использование комментариев 111
Комментарии в этой книге 113
Глава 3. Добавление в программу логики и контроля 115
Интеллектуальная реакция программы 115
Управляющие инструкции 117
Запасной план 122
Проверка более одного условия 123
Более сложные условия 127
Использование управляющих инструкций на практике 133
Работа с повторяющимися задачами с использованием циклов 139
Циклы `while` 139
Циклы и массивы 142
Циклы `for` 144
Циклы `do/while` 146
Функции: многократное использование кода 148
Практика 151
Передача данных функциям 152
Запрос данных от функций 155
Предупреждение конфликта переменных 157
Создание простой викторины на практике 161

ЧАСТЬ II
НАЧАЛО РАБОТЫ С JQUERY

Глава 4. Введение в jQuery 170
О библиотеках JavaScript 170
Получение библиотеки jQuery 173
Ссылки на файл jQuery, расположенный на сервере CDN 175
Загрузка файла jQuery 177
Добавление библиотеки jQuery на страницу 179
Модифицирование веб-страниц 182
Объектная модель документа (DOM) 186
Выбор элементов страницы: подход jQuery 188
Основные селекторы 189
Специальные селекторы 193
Фильтры jQuery 196
Понимание выборок jQuery 198

Добавление содержимого на страницу . . . 201
Замена и удаление выборок . . . 205
Установка и чтение атрибутов элемента . . . 206
Классы . . . 206
Чтение и изменение свойств CSS . . . 210
Одновременное изменение нескольких свойств CSS . . . 212
Чтение, установка и удаление атрибутов HTML . . . 215
Работа с каждым элементом выборки . . . 216
Анонимные функции . . . 216
Ключевые слова `this` и `$(this)` . . . 218
Автоматические «броские цитаты» . . . 220
Обзор . . . 221
Верстка кода . . . 222

Глава 5. Действие/реакция: интерактивные страницы с помощью событий . . . 228
Что такое события? . . . 228
События мыши . . . 230
События документа/окна . . . 231
События форм . . . 233
События клавиатуры . . . 234
Использование событий: способ jQuery . . . 235
Использование событий на практике . . . 238
Больше концепций для событий jQuery . . . 245
Ожидание загрузки HTML-кода . . . 245
События наведения и смещения указателя мыши . . . 249
Объект события . . . 252
Отмена обычного поведения событий . . . 254
Удаление событий . . . 254
Профессиональное управление событиями . . . 257
Другие способы использования функции `on()` . . . 259
Делегирование событий с помощью функции `on()` . . . 262
Создание страницы ЧаВо на практике . . . 266
Обзор задачи . . . 267
Верстка кода . . . 268

Глава 6. Анимация и эффекты . . . 276
Эффекты jQuery . . . 276
Основы отображения и сокрытия . . . 277
Постепенное появление и исчезновение элементов . . . 279
Скользящие элементы . . . 280
Всплывающее окно авторизации на практике . . . 282
Верстка кода . . . 283
Анимация . . . 287
Управление скоростью анимации . . . 289
Выполнение действия после завершения эффекта . . . 291
Анимированная панель навигации на практике . . . 295
Верстка кода . . . 297
Библиотека jQuery, а также переходы и анимация CSS3 . . . 303
Библиотека jQuery и переходы CSS . . . 304
jQuery и анимация CSS . . . 307

Глава 7. Распространенные задачи, решаемые с помощью jQuery . . . 312
Смена изображений . . . 312
Изменение атрибута `src` изображения . . . 313
Смена изображений с помощью jQuery . . . 315
Предварительная загрузка изображений . . . 316
Сменяемые изображения . . . 318
Добавление сменяемых изображений на практике . . . 320
Обзор задачи . . . 321
Верстка кода . . . 322

Фотогалерея с эффектами на практике . . . 327
Обзор задачи . . . 328
Верстка кода . . . 330
Управление поведением ссылок . . . 335
Выборка ссылок с помощью языка JavaScript . . . 336
Определение направления ссылки . . . 337
Не переходите по этой ссылке . . . 338
Открытие внешних ссылок в новом окне . . . 340
Создание новых окон . . . 344
Свойства окна . . . 345
Знакомство с плагинами jQuery . . . 350
На что обратить внимание в плагине jQuery? . . . 352
Основы работы с плагинами jQuery . . . 354
Создание отзывчивого меню навигации . . . 357
HTML-код . . . 358
Каскадная таблица стилей . . . 360
JavaScript . . . 361
Руководство . . . 361
Настройка внешнего вида плагина SmartMenu . . . 366
Глава 8. Улучшение веб-форм . . . 369
Структура форм . . . 369
Выбор элементов формы . . . 372
Получение и установка значений элементов форм . . . 375
События формы . . . 378
Усовершенствование форм . . . 385
Фокусировка на первом элементе формы . . . 385
Отключение/включение элементов формы . . . 387
Сокрытие/отображение параметров формы . . . 388
Усовершенствование простой формы на практике . . . 390
Фокусировка на элементе формы . . . 391
Отключение элементов формы . . . 392
Сокрытие элементов формы . . . 396
Проверка формы . . . 398
Плагин jQuery Validation . . . 400
Простая проверка . . . 403
Расширенная проверка . . . 406
Стилизация сообщений об ошибках . . . 415
Проверка формы на практике . . . 416
Простая проверка на практике . . . 417
Расширенная проверка на практике . . . 420
Проверка состояний флажков и переключателей . . . 424
Форматирование сообщений об ошибках . . . 428

ЧАСТЬ III
НАЧАЛО РАБОТЫ С JQUERY UI

Глава 9. Улучшение интерфейса . . . 432
Что такое jQuery UI? . . . 432
Предназначение jQuery UI . . . 434
Использование плагина jQuery UI . . . 436
Добавление jQuery UI на веб-страницу . . . 439
Создание диалоговых окон с сообщениями . . . 440
Создание диалогового окна на практике . . . 442
Настройка свойств диалогового окна . . . 444
Передача параметров виджету Dialog на практике . . . 449

Открытие диалоговых окон с помощью событий . . . 451
Добавление кнопок в диалоговое окно . . . 453
Добавление кнопок в диалоговое окно на практике . . . 455
Предоставление информации с помощью всплывающих подсказок . . . 463
Быстрое добавление всплывающих подсказок на практике . . . 464
Параметры всплывающих подсказок . . . 465
Использование в подсказке HTML-контента . . . 467
Добавление в подсказку HTML-кода на практике . . . 469
Добавление панелей с вкладками . . . 470
Параметры панели с вкладками . . . 475
Добавление панелей с вкладками на практике . . . 477
Вкладки с удаленным содержимым . . . 482
Экономия пространства с помощью аккордеонов . . . 486
Создание аккордеона jQuery UI на практике . . . 491
Добавление меню на страницу . . . 493
Создание горизонтальной панели навигации . . . 498
Глава 10. Стилизация форм . . . 502
Стильный способ выбора даты . . . 502
Настройка свойств виджета Datepicker . . . 504
Добавление панели для выбора даты рождения на практике . . . 510
Стилизация раскрывающихся списков . . . 514
Настройка свойств раскрывающегося списка . . . 516
Выполнение действия при выборе пункта раскрывающегося списка . . . 518
Стилизация кнопок . . . 521
Настройка кнопок . . . 523
Стилизация переключателей и флажков . . . 525
Предоставление подсказок с помощью функции автозаполнения . . . 527
Использование массивов с виджетом Autocomplete . . . 530
Использование отдельных меток и значений . . . 532
Получение данных для функции автозаполнения с сервера . . . 534
Параметры автозаполнения . . . 538
Использование виджета jQuery UI Form на практике . . . 540
Глава 11. Настройка внешнего вида jQuery UI . . . 548
Знакомство с приложением ThemeRoller . . . 548
Загрузка и использование новой темы . . . 555
Добавление новой темы на существующий веб-сайт . . . 556
Подробнее о CSS-файлах jQuery UI . . . 557
Переопределение стилей jQuery UI . . . 558
Понятие специфичности . . . 559
Как плагин jQuery UI стилизует виджеты . . . 562
Глава 12. Взаимодействия и эффекты jQuery UI . . . 565
Виджет Draggable . . . 565
Добавление виджета Draggable на веб-страницу . . . 566
Применение виджета Draggable на практике . . . 567
Параметры виджета Draggable . . . 569
События виджета Draggable . . . 578
Виджет Droppable . . . 583
Использование виджета Droppable . . . 584
Параметры виджета Droppable . . . 586
События виджета Droppable . . . 589
Drag-and-Drop на практике . . . 595
Сортировка элементов страницы . . . 604
Использование виджета Sortable . . . 604
Параметры виджета Sortable . . . 607
События виджета Sortable . . . 612
Методы виджета Sortable . . . 616

Эффекты jQuery UI 620
Эффекты 622
Параметр `easing` 626
Анимация изменения класса 627

ЧАСТЬ IV
РАСШИРЕННЫЕ СПОСОБЫ ИСПОЛЬЗОВАНИЯ JQUERY И JAVASCRIPT

Глава 13. Введение в технологию Ajax 632
Что такое Ajax 632
Технология Ajax: Основы 635
Части мозаики 635
Взаимодействие с веб-сервером 639
Работа с Ajax с помощью средств jQuery 642
Использование метода `load()` 642
Функция `load()` на практике 646
Методы `get()` и `post()` 652
Форматирование данных, посылаемых на сервер 653
Обработка данных с сервера 658
Обработка ошибок 663
Использование метода `get()` на практике 664
Формат JSON 673
Доступ к данным JSON 675
Сложные объекты JSON 677
Введение в JSONP 681
Добавление фида сервиса Flickr на ваш сайт 681
Построение URL-адреса 683
Использование метода `$.getJSON()` 686
JSON-фид сервиса Flickr 686
Добавление на сайт изображений сервиса Flickr на практике 689

Глава 14. Создание приложения «Список дел» 697
Обзор приложения 697
Добавление кнопки 699
Добавление диалогового окна 700
Добавление задач 706
Маркировка задачи как выполненной 714
Делегирование событий 714
Удаление задач 720
Усовершенствование приложения 723
Редактирование задач 724
Подтверждение удаления 725
Сохранение списков 725
Дополнительные идеи 726

ЧАСТЬ V
ДИАГНОСТИКА, СОВЕТЫ И НЮАНСЫ

Глава 15. Дополнительные возможности библиотеки jQuery 728
Полезные советы и сведения о библиотеке jQuery 728
Конструкция `$()` равнозначна функции `jQuery()` 728
Сохранение выборок в переменных 729
Сокращение числа операций по добавлению контента 731
Оптимизация селекторов 733

Использование документации к библиотеке jQuery 735
Описание страницы сайта с документацией 739
Обход дерева DOM 742
Дополнительные функции для работы с кодом HTML. 749

Глава 16. Совершенствуемся в программировании на языке JavaScript 756
Работа со строками 756
Определение длины строки 756
Изменение регистра строки 758
Поиск в строке: техника **`indexOf()`** 759
Извлечение части строки с помощью метода **`slice()`** 761
Поиск по маске в строках 763
Создание и использование регулярного выражения. 764
Построение регулярного выражения 765
Группировка частей маски 770
Полезные регулярные выражения 772
Сопоставление маски 779
Замена текста 782
Тестирование регулярных выражений 784
Работа с числами 785
Преобразование строки в число. 785
Тест на числа 788
Округление чисел. 789
Форматирование значений в валюте. 789
Генерация случайных чисел 791
Дата и время 793
Получение информации о месяце 793
Получение информации о дне недели 794
Получение информации о времени. 795
Создание даты, отличной от сегодняшней 800
Оптимизация сценариев JavaScript 803
Сохранение параметров в переменных 803
Сохранение параметров в объектах 806
Тернарная операция 807
Инструкция-переключатель 809
Объединение массивов и разбиение строк 812
Выводы 813
Использование внешних файлов JavaScript 813
Ускорение загрузки файлов JavaScript 817

Глава 17. Диагностика и отладка 819
Наиболее распространенные ошибки при программировании на языке JavaScript 819
Незакрытые пары. 819
Кавычки. 823
Использование зарезервированных слов 826
Одинарные знаки равенства в управляющих инструкциях 827
Чувствительность к регистру 828
Некорректный путь к внешнему файлу JavaScript 828
Некорректные пути внутри внешнего файла JavaScript 829
Пропадающие переменные и функции 831
Отладка с помощью веб-консоли 833
Открытие веб-консоли 833
Обзор ошибок с помощью консоли. 834
Использование функции **`console.log()`** для отслеживания выполнения сценария 836

Использование консоли на практике 837
Дополнительные средства отладки 842
Отладка на практике 849

ЧАСТЬ VI
ПРИЛОЖЕНИЕ

Приложение А. Источники знаний по JavaScript 858
Справочные материалы 858
Веб-сайты 858
Книги 859
Основы языка JavaScript 859
Веб-сайты 859
Книги 859
Библиотека jQuery 860
Веб-сайты 860
Книги 860
Расширенные возможности языка JavaScript 861
Статьи и презентации 861
Веб-сайты 861
Книги 861
Язык CSS 862
Веб-сайты 862
Книги 863
Предметный указатель 864

НЕДОСТАЮЩЕЕ ПРИЗНАНИЕ

Об авторе

Дэвид Сойер МакФарланд (David Sawyer McFarland) — президент компании Sawyer McFarland Media (город Портленд, штат Орегон), занимающейся веб-разработкой и обучением. В 1995 г. он создал свой первый сайт — интернет-журнал для профессионалов в области коммуникации. Работал веб-мастером в Калифорнийском университете и в Научно-исследовательском центре мультимедиа в Беркли, контролировал полную реконструкцию сайта Macworld.com под CSS.

Дэвид не только веб-разработчик, но еще и автор книг, преподаватель и консультант. Он преподавал веб-дизайн для аспирантов на факультете журналистики Калифорнийского университета в Беркли, в Центре электронных искусств, в колледже Академии искусств, в Центре цифровых искусств Нью-Медиа и в Портлендском государственном университете. У него есть публикации по теме веб-разработок в журналах Practical Web Design, MX Developer's Journal, Macworld и на сайте CreativePro.com.

Автор с радостью познакомится с отзывами об этой книге, которые можно направлять по адресу **missing@sawmac.com**. (Если же вам нужна техническая поддержка, пожалуйста, ознакомьтесь с ресурсами в Приложении).

О творческом коллективе

Нэн Барбер (Nan Barber) (редактор) работает над серией «Исчерпывающее руководство» в качестве помощника редактора. Она живет в штате Массачусетс со своим мужем и разнообразными электронными устройствами. Адрес ее электронной почты: **nanbarber@gmail.com**.

Мелани Ярброу (Melanie Yarbrough) (выпускающий редактор) работает в городе Кембридж, штат Массачусетс, по которому она разъез-

жает на собранных своими руками мотоциклах. Адрес ее электронной почты: **myarbrough@oreilly.com**.

Дженнифер Дэвис (Jennifer Davis) (технический редактор) является инженером с многолетним опытом работы в области повышения эффективности способов разработки платформ. В качестве главного инженера по автоматизации она помогает компаниям создавать собственные передовые практики по улучшению рабочих процессов, позволяющие уменьшить время на их внедрение. Она организует мероприятия для сообщества Reliability Engineering, группы пользователей Bay Area Chef.

Алекс Стэнгл (Alex Stangl) (технический редактор) более 25 лет профессионально занимается разработкой программного обеспечения, используя многочисленные языки программирования и технологии. Он обожает решать сложные задачи и головоломки, изучать новые языки (в настоящее время изучает Clojure), заниматься техническим редактированием и проводить время с женой и детьми. Адрес его электронной почты: **alex@stangl.us**.

Жасмин Куитин (Jasmine Kwityn) (корректор) — внештатный редактор и корректор. Она проживает в Нью-Джерси со своим мужем, Эдом, и тремя кошками, Мушки, Аксли и Панки. Адрес ее электронной почты: **jasminekwityn@gmail.com**.

Боб Пфалер (Bob Pfahler) (составитель предметного указателя) — внештатный сотрудник, который составил предметный указатель для этого издания по заказу международного партнерства Potomac Indexing, LLC, **www.potomacindexing.com**. Кроме компьютерных технологий он также специализируется на составлении предметных указателей для книг, посвященных бизнесу, менеджменту, биографии и истории. Адрес его электронной почты: **bobpfahler@hotmail.com**.

Слова благодарности

Огромное спасибо всем, кто помог в написании этой книги, особенно Дженнифер Дэвис и Алексу Стэнглу, чья внимательность уберегла меня от вопиющих ошибок. Спасибо также студентам из Портлендского государственного университета, которые терпеливо высиживали на моих лекциях по JavaScript и боролись с моими заданиями по программированию — особенно участникам команды Futzbit за тестирование руководств: Джулии Холл, Эмбер Брукер, Кевину Брауну, Джошу Эллиот-

ту, Трэйси О'Коннор и Блэйку Уомаку. Все мы должны поблагодарить Джона Резига и команду jQuery за создание прекрасного инструмента, позволяющего сделать программирование на JavaScript приятным времяпрепровождением.

И наконец, спасибо Дэвиду Поугу за то, что он заставил меня начать эту книгу, Нэн Барбер — за оттачивание стиля письма, моей жене Шолле, мирившейся с капризами автора, и моим детям Грэму и Кейт за то, что они такие классные.

Дэвид Сойер МакФарланд

О серии «Исчерпывающее руководство»

«Исчерпывающие руководства» — это умные, превосходно написанные книги по компьютерным продуктам, которые не сопровождают печатные руководства пользователя. В каждом издании есть составленный вручную предметный указатель и перекрестные ссылки на конкретные страницы, а не просто на главы.

Недавно вышедшие книги серии «Исчерпывающее руководство»:

- Adobe Photoshop CC 2014. Исчерпывающее руководство Дэвида Поуга.
- iPad. Исчерпывающее руководство Джей Ди Байерсдорфер.
- JavaScript. Подробное руководство Дэвида Макфарланда.
- Создание Web-сайтов. Основное руководство Мэтью Макдональда.
- Цифровая фотография. Исчерпывающее руководство Дэвида Поуга и Аарона Миллера.

С полным списком книг серии «Исчерпывающее руководство» вы можете ознакомиться на сайте **eksmo.ru**.

ВВЕДЕНИЕ

Еще совсем недавно Всемирная паутина казалась очень скучным местом. Составленные из одного лишь HTML-текста веб-страницы отображали информацию и ничего больше. Можно было лишь перейти по ссылке и ждать загрузки новой веб-страницы — вот что считалось настоящей интерактивностью.

Сегодня большинство сайтов практически так же быстро реагируют на действия пользователя, как и обычные настольные приложения, отзываясь на каждый щелчок кнопкой мыши. И все это благодаря теме книги, которую вы держите в руках, — JavaScript и jQuery.

Что такое JavaScript

JavaScript — это язык программирования, позволяющий достичь наивысшей производительности HTML-страниц, наполняя их анимацией, интерактивными элементами и динамическими визуальными эффектами.

Язык JavaScript способен сделать веб-страницы более полезными, обеспечивая немедленную обратную связь. Например, управляемый JavaScript интернет-магазин может мгновенно отображать цену и стоимость доставки в тот момент, когда посетитель выбирает товар. JavaScript способен генерировать сообщение об ошибке сразу после того, как кто-либо попытается отправить форму, в которой отсутствует необходимая информация.

Язык JavaScript также позволяет вам создавать увлекательные динамичные и интерактивные интерфейсы. Например, с помощью JavaScript вы можете преобразовать статическую страницу эскизов изображений в анимированное слайд-шоу. Или сделать нечто незаметное — добавить больше информации на страницу, не перегружая ее, организовав контент в небольшие панели, к которым посетитель может получить доступ простым нажатием кнопки мыши (см. раздел «Добавление панелей с вклад-

ками» главы 9). Или добавить что-либо полезное, например, всплывающие подсказки, которые предоставляют дополнительную информацию об элементах на вашей веб-странице (см. раздел «Предоставление информации с помощью всплывающих подсказок» главы 9).

Основной плюс JavaScript — это незамедлительная реакция. Он позволяет веб-страницам мгновенно отвечать на действия, когда посетитель переходит по ссылке, заполняет форму или просто водит указателем мыши по экрану. Язык JavaScript избавлен от раздражающей медлительности, ассоциирующейся с серверными сценарными языками, вроде PHP, которые опираются на связь между браузером и веб-сервером. Поскольку этот язык не зависит от постоянно загружающихся и перезагружающихся веб-страниц, он позволяет создавать сайты, которые воспринимаются и работают скорее как настольные программы, чем как веб-страницы.

Если вы посещали сервис Google Maps (**maps.google.com**), то видели JavaScript в действии. Сервис Google Maps позволяет посмотреть карту вашего города (или карту любого другого места), увеличивать отдельные детали, чтобы видеть подробные планы улиц и автобусных остановок, или уменьшать масштаб, чтобы посмотреть на местность с высоты птичьего полета, или узнать, как пересечь город, штат или страну. Хотя до появления ресурса Google Maps существовало много сайтов с географическими картами, они всегда требовали многократной перезагрузки страниц (а этот процесс обычно медленный), чтобы получить желаемую информацию. Сервис Google Maps работает без обновления страниц — он немедленно реагирует на ваши действия.

С помощью JavaScript можно создавать как совсем простые программы (например, открывающие новое окно браузера с веб-страницей), так и полноценные веб-приложения, вроде Google Docs (**docs.google.com**), которое позволяет создавать презентации, редактировать документы и конструировать динамические таблицы, используя браузер. При этом вы чувствуете себя так, как будто работаете с настольным приложением.

Немного истории

Язык JavaScript был разработан сотрудником корпорации Netscape Бренданом Айком в далеком 1995 г., он почти так же стар, как сама Всемирная паутина. Хотя сегодня JavaScript очень высоко ценят, у него

достаточно непростое прошлое. Этот язык считался забавой для программистов и использовался для добавления практически бесполезных эффектов, таких как сообщения, бегущие вдоль строки состояния браузера, вроде биржевых сводок, или анимированные бабочки, летящие вслед за указателем мыши при его перемещении по странице. В Интернете можно было легко найти тысячи бесплатных программ на JavaScript (также называемых *сценариями* или *скриптами*), но многие из них не работали в некоторых браузерах, а иногда даже приводили к их сбою.

ПРИМЕЧАНИЕ

JavaScript не имеет ничего общего с языком программирования Java. JavaScript первоначально назывался LiveScript, но маркетологи корпорации Netscape сочли, что достигнут гораздо большего успеха, если проассоциируют его с популярным тогда языком Java. Не путайте эти два языка программирования... особенно на собеседовании при приеме на работу!

В былые времена язык JavaScript также страдал от несовместимости двух известных браузеров: Netscape Navigator и Internet Explorer. Поскольку корпорации Netscape и Microsoft пытались превзойти друг друга, добавляя в свои браузеры новые и (как полагалось) лучшие возможности, два браузера работали по-разному, создавая сложности для разработки программ на JavaScript, которые могли бы хорошо функционировать в обоих браузерах.

ПРИМЕЧАНИЕ

После того как корпорация Netscape ввела JavaScript, в Microsoft стали применять язык jScript — собственную версию JavaScript, которая использовалась в браузере Internet Explorer.

К счастью, сейчас современные браузеры, такие как Firefox, Safari, Chrome, Opera и Internet Explorer 11, стандартизировали большинство методов обращения с JavaScript, упростив написание программ, которые могут работать в любом из этих браузеров. Следует отметить, что некоторая несовместимость современных браузеров по-прежнему существует, поэтому вам придется изучить несколько приемов решения подобных проблем, изложенных в данной книге.

Можно сказать, что JavaScript пережил возрождение, благодаря таким многопрофильным сайтам, как Google, Yahoo и Flickr, которые все шире используют данный язык программирования для создания интерактивных веб-приложений. Поэтому изучать JavaScript сейчас очень своевременно. Имея доступ к обширным знаниям и искусно написанным сценариям, вы, даже будучи начинающим программистом, сможете создавать разнообразные интерактивные элементы для вашего сайта.

ПРИМЕЧАНИЕ

Язык JavaScript также известен под названием ECMAScript. ECMAScript — это официальная спецификация JavaScript, разработанная и поддерживаемая организацией по стандартизации Ecma International: **www.ecmascript.org.**

JavaScript повсюду

JavaScript используется не только для создания веб-страниц. Он доказал свою исключительную полезность. Если вы изучите данный язык программирования, то сможете создавать виджеты Yahoo и приложения Google Apps, писать программы для iPhone и начать работу со сценарными функциями многих программ Adobe, таких как Acrobat, Photoshop, Illustrator и Dreamweaver. Кстати, программа Dreamweaver всегда давала возможность умным программистам JavaScript добавлять собственные команды.

В версии операционной системы OS X Yosemite компания Apple позволила пользователям автоматизировать свои компьютеры, используя язык JavaScript. Кроме того, JavaScript применяется во многих полезных инструментах для веб-разработки пользовательских интерфейсов, например, в Gulp.js, который может автоматически сжимать изображения, файлы CSS и JavaScript. Или в Bower, который позволяет быстро и легко загружать на компьютер такие библиотеки JavaScript, как jQuery, jQuery UI или AngularJS.

JavaScript также становится все более популярным языком для разработки серверных приложений. Платформа Node.js (версия движка JavaScript V8 от компании Google, который выполняет код JavaScript на стороне сервера) в настоящее время охотно используется такими компаниями, как Walmart, PayPal и eBay. Изучение языка JavaScript может

положить начало карьере в сфере разработки серверных приложений. На самом деле, применение кода JavaScript в пользовательском интерфейсе — *фронт-энд* (то есть в браузере) и на стороне веб-сервера — *бэк-энд* — известно как «полный цикл разработки на JavaScript».

Иными словами, изучать язык JavaScript сегодня полезно, как никогда!

Что такое jQuery?

У языка JavaScript есть один маленький секрет: писать на нем трудно. Хотя он и проще многих других, JavaScript — все же язык программирования. И для многих людей, в том числе веб-дизайнеров, процесс программирования является трудным.

Еще больше осложняет дело то, что различные веб-браузеры «понимают» JavaScript по-разному, так что программа, которая работает, скажем, в браузере Chrome, может быть полностью безответной в браузере Internet Explorer 9. Эта обычная ситуация может стоить многих часов тестирования на разных компьютерах и в различных браузерах, чтобы убедиться в том, что программа работает правильно для всей аудитории вашего сайта.

Вот где находит свое применение jQuery. Это библиотека JavaScript, предназначенная для того, чтобы сделать программирование на этом языке проще и веселее. Библиотека JavaScript представляет собой комплексную JavaScript программу, которая одновременно упрощает решение сложных задач и разрешает проблемы несовместимости браузеров. Другими словами, jQuery позволяет преодолеть два крупнейших препятствия в работе с JavaScript: сложность программирования и привередливость различных веб-браузеров.

jQuery является секретным оружием веб-дизайнера при программировании на языке JavaScript. jQuery позволяет вам с помощью одной строки кода выполнять задачи, которые потребовали бы сотен строк программирования на JavaScript и многочасового тестирования в браузере. На самом деле книга исключительно о JavaScript была бы, по меньшей мере, в два раза толще той, которую вы держите в руках, и после прочтения (если бы ее осилили) вы не смогли бы сделать и половину того, что можно выполнить с помощью некоторых знаний о jQuery.

Именно поэтому большая часть данного руководства посвящена библиотеке jQuery, которая позволяет сделать так много и так легко. Другим преимуществом jQuery является то, что вы можете добавить дополнительные функции на ваш сайт с помощью тысяч простых в использовании плагинов jQuery. Например, плагин jQueryUI (о котором вы узнаете в главе 9 «Улучшение интерфейса») позволяет создать множество сложных элементов пользовательского интерфейса, например, панели с вкладками, выпадающие меню, всплывающие календари с функцией выбора даты, используя лишь одну строку кода!

Неудивительно, что jQuery используется на миллионах веб-сайтов (**trends.builtwith.com/javascript/jQuery**). Она встроена прямо в популярные системы управления контентом, такие как Drupal и WordPress. Вы можете даже найти объявления о трудоустройстве для «jQuery программистов» без упоминания о JavaScript. Когда освоите работу с библиотекой jQuery, вы присоединитесь к большому сообществу веб-дизайнеров и программистов, которые используют более простой и мощный подход к созданию интерактивных веб-страниц.

HTML: скелет-основа

Язык JavaScript не полноценен без двух других столпов веб-дизайна — HTML и CSS. Многие программисты упоминают три языка, образующих «слои» веб-страницы: HTML, CSS и JavaScript. Первый — язык разметки гипертекста — обеспечивает *структурный* слой, выразительно организуя содержимое страницы, например, изображения и текст. Второй, CSS (то есть Каскадные таблицы стилей), образует *презентационный* слой, обеспечивая красивый внешний вид HTML-контента. А третий — JavaScript — добавляет *поведенческий* слой, оживляя веб-страницу, делая ее интерактивной, то есть создавая элементы для взаимодействия с посетителями.

Итак, чтобы овладеть языком JavaScript, вы должны иметь хорошее представление как о HTML, так и о CSS.

ПРИМЕЧАНИЕ

Для введения в HTML и CSS прочтите работу «Большая книга веб-дизайна», написанную Терри Фельке-Моррис. Для более глубокого понимания темы «Каскадные таблицы стилей» ознакомьтесь с пособием «Большая книга CSS3» Дэвида МакФарланда.

Язык разметки гипертекста (HTML, Hypertext Markup Language) использует простые команды, называемые *элементами* и состоящими из комбинации *начального и конечного тегов*, для определения различных частей веб-страницы. Например, с помощью следующего HTML-кода создается простая веб-страница:

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8>
<title>Привет, я заголовок этой веб-страницы.</title>
</head>
<body>
Привет, я текст на этой веб-странице.
</body>
</html>
```

Следует отметить, что этот код имеет все базовые элементы, необходимые для веб-страницы. Он начинается со строки декларирования типа документа (`doctype`), которая указывает, документом какого типа является страница и какому стандарту она отвечает. Для каждой новой версии HTML вы должны использовать различные типы документов. В рассматриваемом примере строка `doctype` указывает на то, что эта страница является HTML-документом, использующим версию 5; строка `doctype` для HTML 4.01 или XHTML длиннее и включает унифицированный указатель ресурсов (URL, Uniform Resource Locator), который указывает браузеру на файл в Интернете, содержащий определения для данного типа файлов.

В сущности, строка `doctype` сообщает браузеру, как следует отображать страницу. Объявление типа документа даже может влиять на работу CSS и JavaScript. Если вы забудете указать или неправильно укажете элемент `doctype`, то у вас может возникнуть множество проблем из-за межбраузерных различий в работе с вашими сценариями. Иными словами, всегда указывайте в коде HTML тип документа.

Существует несколько типов документа: HTML 4.01 Transitional, HTML 4.01 Strict; XHTML 1.0 Transitional, XHTML 1.0 Strict, однако они требуют использования длинной и трудной строки кода, в котором легко допустить ошибку. Тип документа HTML5 — `<!DOCTYPE`

html> — короче и проще, поэтому вам следует использовать именно его.

Как работают HTML-элементы

Вы могли заметить, что в приведенном выше HTML-коде большинство команд является парными и что пара окружает блок текста или другие команды. Расположенные в скобках *теги* каждого *элемента* являются инструкциями, которые сообщают браузеру, как отображать веб-страницу. Теги элементов — это «разметочная» часть языка HTML.

Начальный (*открывающий*) тег каждого элемента сообщает, с какого места начинается инструкция, а конечный (*закрывающий*) тег говорит о том, где данная инструкция завершается. Конечные теги всегда содержат слеш / после первой скобки <. Например, тег обозначает начало абзаца, а тег — его окончание. Некоторые элементы, например, img, input и br не содержат закрывающих тегов.

Для правильной работы веб-страницы вы должны включить в нее как минимум три элемента.

- Элемент html ставится в начале веб-страницы (после объявления типа документа) и с добавлением косой черты — в ее конце. Он сообщает браузеру, что информация записана с помощью языка HTML. Все содержимое страницы, включая остальные теги, находится между открывающим и закрывающим тегами элемента html.

Если вы представите веб-страницу в виде дерева, то элемент html будет его стволом. От ствола отходят две ветви (основные составляющие веб-страницы) — *раздел заголовка* и *тело*.

- Открывающий и закрывающий теги элемента head содержат *раздел заголовка* страницы. Они также могут сообщать и другую, скрытую информацию (например, ключевые слова, которые используются браузерами и поисковыми системами).

В раздел заголовка иногда также включают сведения, используемые браузером для отображения веб-страницы и для придания ей интерактивности. Например, в заглавие документа можно поместить каскадные таблицы стилей. Кроме того, именно раздел заголовка документа обычно содержит элементы программирования на JavaScript и ссылки на файлы JavaScript.

- Открывающий и закрывающий теги элемента `body` содержат *тело* веб-страницы со всей информацией, которая отображается в окне браузера: заголовки, текст, изображения и т. д.

  Внутри `body` обычно находятся следующие элементы.

- Открывающий тег элемента абзаца, `<p>`, сообщает браузеру, где в тексте начинается абзац, а закрывающий тег, `</p>`, — где он заканчивается.

- Элемент `strong` увеличивает толщину шрифта. Если вы поместите какой-либо текст между открывающим (`<strong>`) и закрывающим (`</strong>`) тегами данного элемента, то этот текст будет выделен полужирным шрифтом. Например, фрагмент HTML-кода `<strong>Внимание!</strong>` указывает браузеру на то, что слово «Внимание!» должно быть выделено полужирным шрифтом.

- Элемент `a` (элемент привязки), создает на веб-странице *гиперссылку*. Если щелкнуть по гиперссылке (или *ссылке*), то можно попасть на любую другую веб-страницу. Вы сообщаете браузеру, куда ведет ссылка, помещая веб-адрес между открывающим и закрывающим тегами элемента `a`. Например, вы можете написать:

```
<a href="www.eksmo.ru">Щелкните здесь!</a>
```

Браузеру «известно», что если посетитель щелкнет кнопкой мыши по фразе «Щелкните здесь!», то следует перейти на сайт издательства «Эксмо». Часть `href` элемента `a` называется *атрибутом*, а URL-адрес — *значением*. В данном примере `www.eksmo.ru` — это *значение* атрибута `href`.

УСКОРЯЕМ РАБОТУ

Проверка веб-страниц

Как уже отмечалось, строка `doctype` определяет, какой тип HTML или XHTML вы использовали для создания веб-страницы. Дело в том, что правила несколько отличаются в зависимости от типа документа. Например, по сравнению с HTML 4.01, XHTML не позволяет использовать незакрытый тег `<p>` и требует, чтобы все имена и атрибуты были набраны с использованием нижнего регистра (например, `<a>` вместо `<A>`). Язык HTML5 включает новые элементы и позволяет вам использовать синтаксис HTML или XHTML. Поскольку у каждой версии HTML свои правила, вы всегда должны *проверять* созданные веб-страницы.

Валидатор HTML-кода — это программа, проверяющая веб-страницу на наличие ошибок. Она проверяет тип документа страницы, а затем анализирует ее код, чтобы удостовериться, что тот отвечает правилам, заданным для данного типа документа. Например, валидатор отмечает такие ошибки, как неправильно записанный или незакрытый тег. Консорциум Всемирной паутины — организация, отвечающая за технологии, применяемые во Всемирной паутине, имеет бесплатный онлайн-валидатор, доступный по адресу **validator.w3.org**. Вы можете скопировать ваш HTML-код и вставить его в веб-форму, загрузить веб-страницу или указать на уже существующий в Интернете документ; валидатор проанализирует HTML-код и выдаст заключение о том, корректна страница или нет. Если на странице есть ошибка, то валидатор опишет ее и сообщит, в какой строке HTML-файла она находится.

Правильный HTML-код — это не просто демонстрация хорошего тона, он помогает обеспечить правильную работу ваших программ, написанных на языке JavaScript. Многие JavaScript-программы подразумевают манипуляцию HTML-кодом страницы, например, идентификацию конкретного элемента формы или размещение в определенном месте нового HTML-кода (наподобие сообщения об ошибке). Как правило, для того чтобы JavaScript мог получить доступ к веб-странице и выполнить какие-либо манипуляции, HTML-код должен находиться в рабочем состоянии. Забыв закрыть элемент, использовав один и тот же идентификатор более одного раза или неправильно разместив HTML-теги, вы спровоцируете хаотичное поведение JavaScript-кода или вообще сделаете его нерабочим.

CSS: Добавление стилей на веб-страницу

В самом начале развития Всемирной паутины HTML был единственным языком, который вам следовало знать. Вы могли создавать страницы с цветным текстом и изображениями, а также выделять слова, используя различные размеры, шрифты или цвета. Но сегодня веб-дизайнеры применяют каскадные таблицы стилей для придания своим страницам визуальной сложности и утонченности. Каскадные таблицы стилей (CSS, Cascading Style Sheets) — это язык форматирования, который позволяет сделать текст внешне привлекательным, создать сложные шаблоны страниц и придать вашему сайту индивидуальность.

К HTML следует относиться просто как к языку, который вы используете для структурирования страницы. Он помогает расположить информацию, которую вы хотите донести. Например, элементы `h1` и `h2`

обозначают заголовки, устанавливая их относительную важность: *заголовок 1* важнее, чем *заголовок 2*. Элемент `p` обозначает простой абзац информации. Другие элементы обеспечивают дальнейшее структурирование: с помощью элемента `ul` можно оформить пункты маркированного списка (например, чтобы сделать список ингредиентов более понятным).

С другой стороны, язык CSS делает хорошо организованный HTML-контент более красивым и удобным для чтения. По сути, *стиль* CSS — это просто правила, которые сообщают браузеру, как следует отображать на странице тот или иной элемент. Например, с помощью CSS вы способны задать для всего текста, заключенного в элемент `h1`, размер 36 пикселов, шрифт Verdana и оранжевый цвет. С помощью CSS также можно делать и более серьезные вещи, например, добавлять границы, изменять поля и даже контролировать точное позиционирование элемента на странице.

Когда речь заходит о JavaScript, некоторые наиболее важные изменения, которые можно внести на страницу, предполагают применение CSS. С помощью JavaScript вы можете добавить или удалить стили CSS из HTML-элемента или динамически изменять свойства CSS, основываясь на введенной пользователем информации или на щелчках кнопкой мыши. Разрешается даже анимировать изменение свойств стиля (скажем, анимировать изменение цвета фона с желтого на красный). Например, вы можете заставить элемент появляться на странице и исчезать с нее, просто изменив значение свойства CSS `display`. Чтобы заставить элемент перемещаться по экрану, следует динамически изменять свойства позиционирования CSS, используя JavaScript.

Анатомия стиля

Стиль, определяющий внешний вид конкретного элемента — это правило, объясняющее браузеру, как оформить что-либо, например, сделать синий заголовок, нарисовать вокруг фотографии красную рамку или создать сбоку окно шириной в 150 пикселов, в котором будет находиться список ссылок. Если бы стиль умел говорить, он сказал бы примерно следующее: «Эй, браузер, вот *это* должно выглядеть *так*». На самом деле стиль состоит из двух элементов: форматируемого браузером элемента веб-страницы (*селектора*) и конкретной инструкции форматирования (*блока объявления*). Селектором, например, может являться заголовок, абзац текста, фотография и т. д. Блоки объявления могут

выделить текст синим цветом, создать красную рамку вокруг абзаца, поместить фотографию в центр страницы — всех возможностей не перечислить.

ПРИМЕЧАНИЕ

Стили CSS часто называют *правилами*. В этой книге термины «стиль» и «правило» взаимозаменяемы.

Разумеется, стили CSS не могут «общаться» на человеческом языке. У них есть собственный язык. Например, чтобы задать стандартный цвет и размер шрифта для всех абзацев веб-страницы, вам следует написать следующее:

```
p { color: red; font-size: 1.5em; }
```

Этот стиль просто говорит: «Сделай текст во всех абзацах, отмеченных элементом p, красным, высотой 1,5 em» (*em* — это единица измерения, основанная на стандартной высоте текста в браузере). Как видно на рис. В.1, даже простой стиль содержит несколько элементов.

- **Селектор.** Сообщает браузеру, какому элементу (или элементам) на странице следует присвоить стиль. Такими элементами могут быть заголовок, абзац, изображение или ссылка. На рис. В.1 селектор p относится к элементу p, что заставляет браузеры форматировать все элементы p, используя указания по форматированию для этого стиля. С помощью большого количества селекторов CSS и с творческим подходом к делу вы можете получить детальный контроль над форматированием вашей страницы. (Селекторы являются важной частью использования библиотеки jQuery, вы найдете их подробное описание в разделе «Выбор элементов страницы: подход jQuery» главы 4.)
- **Блок объявления.** Код, следующий за селектором, включает все условия форматирования, применяемые к селектору. Этот блок начинается с открывающей фигурной скобки { и заканчивается закрывающей фигурной скобкой }.
- **Объявление.** Между открывающей и закрывающей скобками блока объявления вы можете добавить одно или несколько *объявлений* или инструкций по форматированию. Каждая инструкция состоит из двух частей — *свойства* и *значения* — и заканчивается точкой с запятой. Свойство и значение разделяются двоеточием.

- **Свойство.** В языке CSS существует целый ряд вариантов форматирования, называемых *свойствами*. Свойство — это слово или словосочетание, обозначающее определенный стилевой эффект. Большинство свойств имеет прямую связь между их названием и назначением, например, `font-size` (размер шрифта), `margin-top` (отступ от верхнего края) или `background-color` (цвет фона). Например, свойство `background-color`, как вы, вероятно, догадались, позволяет задать цвет фона.

ПРИМЕЧАНИЕ

Если вам необходимо освежить свои знания по теме CSS, прочтите работу Дэвида МакФарланда «Большая книга CSS3».

- **Значение.** И наконец, вы можете проявить свой художественный талант, присвоив свойству CSS *значение*, например, сделав фон синим, красным, фиолетовым или желто-зеленым. Различные свойства CSS требуют особых типов значений: цвет (например, `red` или `#FF0000`), длина (`18px`, `2in` или `5em`), гиперссылка (например, `images/background.gif`) или специальное ключевое слово (например, `top`, `center` или `bottom`).

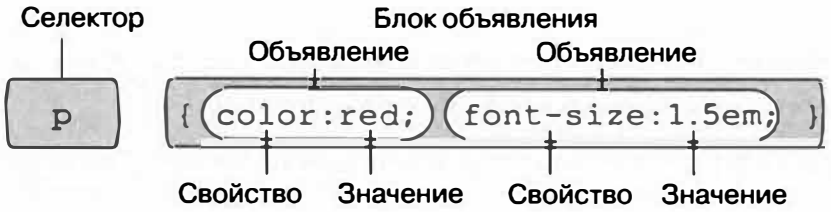

Рис. В.1. Стиль (или правило) состоит из двух основных частей: селектора, сообщающего браузеру, что форматировать, и блока объявления, в котором в виде списка размещены инструкции по форматированию, используемые браузером для оформления селектора

Вам не обязательно записывать код стиля в одну строку, как показано на рис. В.1.

Большинство стилей имеет множество свойств форматирования, поэтому вы можете сделать их более удобными для чтения, разбив код на строки.

Например, поместить селектор и открывающую скобку на первую строку, каждый компонент объявления — на отдельную строку, а закрывающую скобку — на последнюю, как показано далее:

```
p {
color: red;
font-size: 1.5em;
}
```

Также бывает полезно отделять свойства с помощью отступа, табуляцией или несколькими пробелами, чтобы наглядно разделять селектор и компоненты объявления, благодаря чему легко понять, что есть что. И наконец, пробел между значением свойства и двоеточием необязателен, но он повышает удобочитаемость стиля.

Допустимо оставлять между элементами пробел любой ширины. Например, вы можете написать: `color:red`, `color:  red` или `color : red`.

Программы для верстки кода на языке JavaScript

Для создания веб-страниц, состоящих из HTML, CSS и JavaScript, вам не нужно ничего, кроме простого текстового редактора, например, Блокнота (Windows) или TextEdit (OS X). Однако после написания нескольких сотен строк кода JavaScript вы, возможно, захотите перейти к программе, которая больше подходит для работы с веб-страницами. В этом разделе перечислены некоторые распространенные платные и бесплатные программы.

ПРИМЕЧАНИЕ

Существуют сотни инструментов, позволяющих создавать веб-страницы и писать программы на языке JavaScript, так что следующий далее список ни в коем случае не является полным. Это просто перечисление наиболее популярных программ, используемых сегодня фанатами JavaScript.

Бесплатные программы

Существует множество бесплатных программ для редактирования веб-страниц и таблиц стилей. Если вы по-прежнему используете программы Блокнот или TextEdit, попробуйте одну из представленных в следующем кратком списке:

- **Brackets** (Windows, OS X и Linux, **brackets.io**) — редактор компании Adobe с открытым исходным кодом. Эта программа распространяется свободно (коммерческая версия с расширенным функционалом называется Edge Code), предоставляет множество отличных функций, включая предварительный просмотр в браузере в режиме реального времени, и написана на JavaScript!
- **Notepad++** (Windows, **notepad-plus-plus.org**) — друг кодировщика. Программа подсвечивает синтаксис языка JavaScript и HTML-код и позволяет сохранять макросы, а также присваивать им сочетания клавиш, так что вы сможете автоматизировать процесс вставки часто используемых фрагментов кода.
- **HTML-Kit** (Windows, **www.chami.com/html-kit**) — мощный редактор HTML/XHTML, содержащий большое количество полезных возможностей, например, предварительный просмотр веб-страницы прямо в окне программы (вам не придется переключаться между браузером и редактором), сочетания клавиш для вставки HTML-элементов и многое другое.
- **CoffeeCup Free HTML Editor** (Windows, **www.coffeecup.com/free-editor**) — это бесплатная версия коммерческого HTML-редактора CoffeeCup.
- **TextWrangler** (OS X, **www.barebones.com/products/textwrangler**) — бесплатная программа, являющаяся упрощенной версией BBEdit — широко известного текстового редактора для операционной системы OS X. Программа TextWrangler не имеет всех встроенных в редактор BBEdit HTML-инструментов, но в ней есть функция окрашивания синтаксиса (подсвечивание HTML-элементов и свойств разными цветами, что облегчает просмотр страницы и определение ее частей), поддержка FTP (так что вы можете загружать файлы на веб-сервер) и многое другое.
- **Eclipse** (Windows, Linux, OS X; **www.eclipse.org**) — это бесплатная программа, часто выбираемая Java-разработчиками, которая включает инструменты для работы с HTML, CSS и JavaScript. Существует также и специальная версия для JavaScript-разработчиков (**www.eclipse.org/downloads/packages/eclipse-ide-javascript-web-developers/indigor**), а также Eclipse плагины автозаполнения для jQuery (**marketplace.eclipse.org/category/free-tagging/jquery**).
- **Aptana Studio** (Windows, Linux, OS X; **www.aptana.org**) — мощная бесплатная среда для программирования, предусматривающая

инструменты для работы с HTML, CSS, JavaScript, PHP и Ruby on Rails.

- **Vim** и **Emacs** — проверенные временем текстовые редакторы Unix world. Они входят в комплект операционных систем OS X и Linux, и вы можете скачать версию для операционной системы Windows. Эти программы используются серьезными программистами, однако для большинства людей являются сложными на этапе освоения.

Коммерческие программы

Коммерческие продукты варьируются от недорогих текстовых редакторов до полноценных конструкторов для создания сайтов со всем, чего душа пожелает:

- **Atom** (Windows и OS X, **atom.io**) является новой программой, коммерческая версия которой еще не поступила в продажу, однако в данный момент доступна бесплатная бета-версия. Данное приложение разработано сообществом GitHub (сайт, пользователи которого совместно работают над проектами). Оно предоставляет множество функций, отвечающих требованиям современных разработчиков. Программа является модульной, что позволяет использовать плагины сторонних разработчиков, которые позволяют расширить функционал приложения.
- **SublimeText** (Windows, OS X и Linux, **www.sublimetext.com**) пользуется большой популярностью среди программистов. Этот текстовый редактор содержит множество функций, сберегающих время программистов, работающих с JavaScript, таких как автовставка парных символов, автоматически ставящая пару для знака препинания (например, программа добавляет закрывающую круглую скобку, после того как вы напечатали открывающую).
- **EditPlus** (Windows, **www.editplus.com**) — недорогой текстовый редактор, включающий окрашивание синтаксиса, FTP, автозаполнение и другие полезные функции.
- **BBEdit** (OS X, **www.barebones.com/products/bbedit**) — популярный текстовый редактор с большим количеством инструментов для работы с HTML, XHTML, CSS, JavaScript и др. Включает множество полезных инструментов и сочетаний клавиш.
- **Dreamweaver** (OS X и Windows, **www.adobe.com/ru/products/dreamweaver.html**) — визуальный редактор веб-страниц, который

позволяет увидеть, как ваша страница выглядит в браузере, включает мощный текстовый редактор для написания программ на JavaScript и отличные инструменты для создания и управления CSS. Чтобы разобраться в том, как использовать данную программу, обратитесь к пособию «Adobe Dreamweaver CC. Официальный учебный курс».

Об этой книге

В отличие от обычных программ, таких как Microsoft Word или Dreamweaver, JavaScript — это не обособленный продукт, разработанный отдельной компанией. Не существует службы поддержки по JavaScript, где было бы написано качественное руководство для рядового веб-разработчика. Хотя вы можете найти большое количество полезной информации на таких сайтах, как Mozilla.org (**developer.mozilla.org/ru/docs/Web/JavaScript/Reference**) или **www.ecmascript.org**, единственного и авторитетного источника информации по языку программирования JavaScript не существует.

Из-за отсутствия руководства по JavaScript, люди, которые пытаются изучить этот язык программирования, часто не знают, с чего начать, а некоторые нюансы JavaScript могут подставить подножку даже маститым веб-профи, поэтому задача данной книги — послужить в качестве этого недостающего руководства по JavaScript. Здесь вы найдете пошаговые инструкции по использованию языка JavaScript для создания интерактивных веб-страниц.

Вы также найдете документацию по jQuery по адресу: **api.jquery.com**. Однако она написана программистами для программистов, так что объяснения по большей части краткие и технические. И поскольку использование библиотеки jQuery проще обычного программирования на языке JavaScript, это издание научит вас фундаментальным принципам и техникам работы с этим инструментом.

Книга «JavaScript и jQuery: Исчерпывающее руководство» предназначена для подготовленных читателей, уже имеющих опыт создания веб-страниц. Вы должны разбираться в HTML и CSS, чтобы получить максимум пользы от этой книги, поскольку JavaScript часто тесно взаимодействует с этими языками. Основные обсуждения предназначены для «опытных новичков» или средних пользователей компьютера. Но если написание веб-страниц для вас — новинка, то специальные рубрики под названием «Ускоряем работу» позволят получить вводную

информацию, которая необходима для понимания обсуждаемой темы. Если же вы — продвинутый веб-виртуоз, то обращайте внимание на рубрики «Курс опытного пользователя». В них предлагаются технические советы, полезные приемы и сочетания клавиш.

ПРИМЕЧАНИЕ

В данном издании периодически встречаются рекомендации *других* книг, посвященных темам, имеющим отношение к JavaScript.

Подход к JavaScript, принятый в книге

JavaScript — это настоящий язык программирования. Он работает не так, как HTML или CSS, и имеет свои собственные правила (зачастую весьма сложные). Веб-дизайнерам иногда трудно переключиться и начать думать как программист. Помните, что не существует *единой* книги, которая рассказала бы вам обо всем, что следует знать о языке JavaScript.

Автор книги «JavaScript и jQuery: Исчерпывающее руководство» не ставил перед собой цель сделать из вас еще одного великого программиста. Издание было задумано для того, чтобы поближе познакомить веб-дизайнеров с нюансами языка JavaScript и затем перейти к изучению библиотеки jQuery, чтобы вы смогли сделать сайт по-настоящему интерактивным как можно быстрее и проще.

Здесь вы познакомитесь с основами программирования и языка JavaScript, однако знание только этих основ не позволит вам создавать высококлассные сайты. 800 страниц недостаточно, чтобы научить вас всему, что следует знать о JavaScript для построения сложных интерактивных веб-страниц. Именно поэтому большая часть книги посвящена jQuery — популярной библиотеке JavaScript, которая освободит вас от незначительных и трудоемких этапов создания программ, работающих во всех браузерах.

После изучения основ языка JavaScript вы сразу же перейдете к созданию продвинутых интерактивных веб-страниц с небольшой, ну хорошо, с БОЛЬШОЙ помощью библиотеки jQuery. Задумайтесь об этом. Вы, конечно, могли бы построить дом, срубая и распиливая собственный строевой лес, собирая собственные окна, дверные проемы, двери, делая собственную плитку и т. д. Подход «сделай сам» характерен для многих

пособий по JavaScript. Но у кого на это есть время? Принцип, используемый в данной книге, больше напоминает возведение дома с применением готовых деталей и их сборкой на основе базовых умений. В итоге получится красивый функциональный дом, построенный за время, которое у вас бы ушло на изучение каждой отдельной стадии процесса строительства.

Структура книги

«JavaScript и jQuery: Исчерпывающее руководство» включает в себя пять частей, каждая из которых состоит из нескольких глав:

- **Часть I** предоставляет вам базовую информацию. Вы изучите основные компоненты языка JavaScript, а также получите некоторые полезные советы по компьютерному программированию в целом. Научитесь добавлять сценарий на веб-страницу, хранить информацию и манипулировать ею, добавлять в программу логические элементы, чтобы она могла реагировать на различные ситуации. Вы также узнаете, как обращаться с окном браузера, хранить и читать файлы cookie, реагировать на различные события, такие как щелчки кнопкой мыши и отправку форм, а также изменять HTML-код веб-страницы.
- **Часть II** представляет самую популярную библиотеку языка JavaScript — jQuery. В данной части вы изучите основы этого удивительного инструмента, который повысит вашу продуктивность и расширит ваши способности в программировании. Вы научитесь выделять и манипулировать элементами страницы, добавлять интерактивность, заставляя элементы страницы реагировать на действия ваших посетителей, а также создавать визуальные эффекты и анимацию.
- **Часть III** содержит информацию о проекте jQuery UI — библиотеке JavaScript, позволяющей создавать полезные «виджеты» и эффекты, а также легко добавлять такие элементы пользовательского интерфейса, как панели с вкладками, диалоговые окна, аккордеоны и выпадающие меню. Библиотека jQuery UI позволит вам создавать стильные и гармоничные пользовательские интерфейсы для ваших крупных веб-приложений.
- **Часть IV** посвящена некоторым расширенным способам применения jQuery и JavaScript. Например, в главе 13 описана технология, которая сделала JavaScript одним из наиболее привлекательных для изучения веб-языков. Вы научитесь использовать JavaScript для того, чтобы ваши страницы могли получать информацию и обнов-

ляться на основе данных, предоставляемых веб-сервером, — без необходимости загрузки новой веб-страницы. Глава 14 позволит вам шаг за шагом разработать приложение для создания списков дел, используя jQuery и jQuery UI.

- **Часть V** представит более сложные концепции. Вы больше узнаете о том, как эффективно использовать библиотеку jQuery и познакомитесь с расширенными функциями языка JavaScript. Материал этой части пригодится, когда окажется, что ничего не работает и ваша безупречная программа на JavaScript даже не собирается делать то, что вы хотите (или хуже: она вообще ничего не делает!). Вы узнаете о наиболее типичных ошибках, которые часто совершают новоиспеченные программисты, а также о технике нахождения и устранения неточностей в ваших программах.

В приложении, которое расположено в конце книги, приводится подробный список справочных материалов, которые пригодятся вам для дальнейшего исследования языка программирования JavaScript.

Самое основное

Для использования этой книги и, конечно же, компьютера вы должны овладеть некоторыми основами. Данное издание предполагает, что вы знакомы со следующими терминами и концепциями:

- **Щелчок кнопкой мыши.** В этой книге предполагаются три способа использования компьютерной мыши или сенсорной панели ноутбука. *Щелкнуть* кнопкой мыши (по умолчанию предполагается щелчок левой кнопкой) — значит навести на какой-либо объект на экране указатель и, не двигая его, нажать и отпустить кнопку мыши (или кнопку сенсорной панели ноутбука). *Щелчок правой кнопкой мыши* подразумевает такое же действие, только правой кнопкой. *Двойной щелчок* означает два быстрых щелчка друг за другом, опять же при неподвижном указателе мыши на экране. *Перетаскивание* — это перемещение указателя мыши по экрану, *во время* которого ее кнопка удерживается нажатой.

> **СОВЕТ**
>
> Если вы используете компьютер Mac и у вас нет правой кнопки мыши, вы можете добиться того же эффекта, нажимая клавишу **Control**, во время щелчка кнопкой мыши.

Когда от вас требуется **⌘**+*щелчок* на компьютере Mac или щелчок с нажатой клавишей **Ctrl** на ПК, вы должны щелкнуть кнопкой мыши, удерживая нажатой клавишу **⌘** или **Ctrl** соответственно (обе находятся рядом с клавишей **Пробел**).

- **Меню.** *Меню* — это строка в верхней части окна, содержащая слова: **Файл** (File), **Правка** (Edit) и т. д. Щелчок кнопкой мыши по названию меню вызывает появление списка команд.
- **Сочетания клавиш.** Если при наборе текста вы испытываете прилив творческой энергии, иногда совсем некстати отрывать руку от клавиатуры, брать мышь и выбирать пункт меню (например, для включения жирного начертания). Поэтому многие опытные пользователи компьютера предпочитают запускать команды меню, нажимая определенные сочетания клавиш на клавиатуре. Например, в браузере Firefox вы можете нажать сочетание клавиш **Ctrl++** (Windows) или **⌘++** (OS X), чтобы увеличить текст на веб-странице. Когда вы видите такую инструкцию, как «нажмите сочетание клавиш **⌘+B**», нажмите клавишу **⌘** и, не отпуская ее, нажмите клавишу **B**, а затем отпустите обе клавиши.
- **Основы использования операционной системы.** В этой книге предполагается, что вы умеете открывать программу, путешествовать по Всемирной паутине и загружать файлы. Вы должны знать, как использовать меню **Пуск** (Windows), панель **Dock** или меню **Apple** (OS X), а также окно **Панель управления** (Windows) и **Системные настройки** (OS X).

Если вы усвоили эту информацию, то располагаете достаточной технической базой, чтобы получать удовольствие от чтения книги «JavaScript и jQuery: Исчерпывающее руководство».

Об ⇒ этих ⇒ стрелках

В этой книге, а также в других изданиях серии «Исчерпывающее руководство», вы будете встречаться с конструкциями, подобными следующей: «Откройте каталог *Система/Библиотека/Шрифты*».

Это сокращение гораздо более длинной инструкции, которая просит вас последовательно открыть три вложенных каталога: «На вашем жестком диске находится каталог *Система*. Откройте его. В окне каталога «Система» находится каталог *Библиотека*; дважды щелкните по

нему кнопкой мыши. А в этом каталоге есть еще один каталог с именем *Шрифты*. Откройте его, дважды щелкнув по нему кнопкой мыши».

Аналогично данные стрелки помогают упростить выбор пунктов меню, как показано на рис. В.2.

Рис. В.2. В этой книге используются стрелки для упрощения инструкций, связанных с использованием меню. Например, **Вид** ⇒ **Масштаб** ⇒ **Увеличить** (View ⇒ Zoom ⇒ Zoom In) соответствует инструкции «Зайдите в меню **Вид** (View), найдите в нем меню **Масштаб** (Zoom), в котором задействуйте команду **Увеличить** (Zoom In)»

О ресурсах во Всемирной паутине

Данная книга написана для того, чтобы сделать вашу работу во Всемирной паутине оперативнее и профессиональнее, поэтому естественно, что большая часть ценности этого пособия также находится во Всемирной паутине. Во всемирной паутине вы найдете файлы примеров для накопления опыта.

Файлы примеров

На страницах данной книги вы найдете множество *реальных примеров* — пошаговых инструкций. Вы сможете выполнить их, используя «сырые» материалы (например, изображения и полуготовые веб-страницы),

которые доступны по адресу **http://eksmo.ru/Primers_js_jq.zip**. Вы почти ничему не научитесь, если станете просто читать примеры, расслабленно сидя в мягком кресле. Но если вы потратите время и проработаете их на компьютере, то поймете, как профессиональный дизайнер создает веб-страницы.

Среди файлов примеров вы также найдете готовые веб-страницы, так что вы сможете сравнить результаты вашей работы с тем, что должно было получиться в итоге. Другими словами, вы не просто будете просматривать JavaScript-код на страницах книги, но и найдете настоящие работающие веб-страницы.

Часть I

ВВЕДЕНИЕ В JAVASCRIPT

Глава 1. Ваша первая программа на языке JavaScript

Глава 2. Грамматика языка JavaScript

Глава 3. Добавление в программу логики и контроля

Глава 1
ВАША ПЕРВАЯ ПРОГРАММА НА ЯЗЫКЕ JAVASCRIPT

Язык HTML не имеет никаких расширенных функций: он не умеет считать, не в состоянии проанализировать, правильно ли пользователь заполнил анкету, не способен принять решение на основе диалога с посетителем сайта. В основном HTML позволяет читать тексты, просматривать изображения и переходить по ссылкам на другие веб-страницы, содержащие текст и картинки. Чтобы веб-страницы реагировали на действия посетителей вашего сайта, необходимо «добавить им интеллекта». В этом случае вам не обойтись без помощи языка JavaScript.

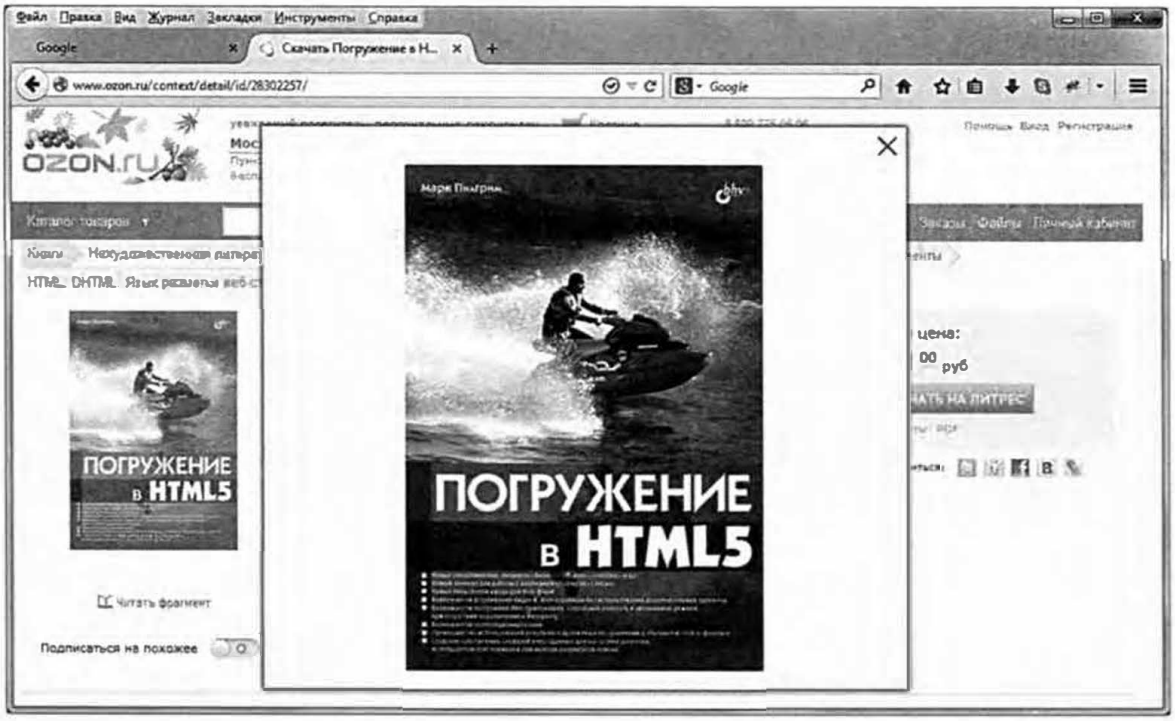

Рис. 1.1. JavaScript позволяет веб-страницам реагировать на действия посетителей. Например, на сайте **Ozon.ru** при щелчке кнопкой мыши по эскизу книги появляется его увеличенное изображение

JavaScript наделяет веб-страницы способностью осознанно реагировать на действия пользователя. С его помощью вы можете создать *умные* веб-формы, которые уведомят посетителя в случае, если он забудет ввести важную информацию; заставить элементы появляться, ис-

чезать или перемещаться по экрану (рис. 1.1); вы даже можете обновить содержимое страницы на основе информации, полученной с сервера, не прибегая к ее перезагрузке. Одним словом, язык JavaScript позволит сделать ваши веб-сайты более привлекательными, полезными и эффективными.

ПРИМЕЧАНИЕ

HTML5 *предусматривает* некоторые дополнительные возможности, включая проверку форм. Но так как не все браузеры поддерживают эти нововведения (и так как вы можете сделать с формами гораздо больше с помощью JavaScript), вам не обойтись без языка JavaScript, если вы хотите создавать самые лучшие, интерактивные и дружественные пользователю формы. Вы можете узнать больше о HTML5 и веб-формах из книги Джона Дакетта «HTML и CSS. Разработка и дизайн веб-сайтов».

Введение в программирование

Большинство людей, слыша словосочетание «компьютерное программирование», представляют себе гениальных очкариков, согнувшихся над клавиатурой и с утра до ночи набирающих совершенно непонятную тарабарщину. И, по правде говоря, в некоторых случаях так и бывает. Программирование может показаться невероятным волшебством, недоступным простым смертным. Однако большинство его концепций несложны для понимания. Что касается языков программирования, то JavaScript достаточно дружелюбен по отношению к начинающим программистам.

Тем не менее JavaScript сложнее, чем языки HTML или CSS, а программирование для веб-дизайнера часто является terra incognita, поэтому одна из целей этой книги — помочь вам начать думать как программист. Вы изучите фундаментальные концепции программирования, которые сможете применять при написании сценариев JavaScript, ActionScript или при создании настольных приложений на языке C++. Еще более важно то, что вы научитесь подходить к задаче по программированию, точно зная, чего хотите.

Многие веб-дизайнеры пасуют перед странными символами и словами, используемыми в JavaScript. Обычная программа на языке JavaScript наполнена символами `{}`, `[]`, `;`, `,`, `()`, `!`, `=` и незнакомыми словами (`var`,

`null`, `else if`). Это похоже на изучение иностранного языка. Вам приходится учить новые слова, новые правила пунктуации и разбираться в том, как их следует сочетать, чтобы общение было эффективным.

Каждый язык программирования имеет собственный набор ключевых слов и символов, а также уникальный набор правил их сочетания — *синтаксис* языка. Изучение синтаксиса JavaScript подобно изучению грамматики другого языка и пополнению словарного запаса. Вы должны запоминать слова и правила (или, как минимум, держать под рукой книгу для справки). Когда вы учитесь говорить на другом языке, то быстро обнаруживаете, что ударение, поставленное не на тот слог, может сделать слово непонятным. Подобным образом обычная опечатка или даже пропуск знака препинания может помешать работе программы JavaScript или инициировать ошибку в браузере. В процессе обучения вы будете допускать много ошибок — такова природа программирования.

Возможно, программирование на языке JavaScript сначала вас разочарует — вы потратите много времени, отслеживая ошибки, допущенные в ходе написания сценария. Кроме того, вы можете решить, что некоторые концепции программирования слишком сложны для понимания. Но не волнуйтесь: если вы раньше уже пытались изучать язык JavaScript и бросили, потому что это показалось вам слишком сложным, то данное пособие поможет вам преодолеть преграды, с которыми часто сталкиваются начинающие программисты. Если же у вас есть опыт программирования, то эта книга расскажет вам о характерных особенностях языка JavaScript и уникальных концепциях, связанных с веб-программированием.

К тому же, данная книга посвящена не только языку JavaScript, но и jQuery — самой популярной библиотеке JavaScript. jQuery делает программирование на JavaScript намного легче. Так что, приобретя некоторые знания по JavaScript и вооружившись помощью jQuery, вы уже очень скоро сможете создавать сложные интерактивные веб-сайты.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Компилируемые и сценарные языки

JavaScript называют сценарным языком. Я уже слышал этот термин, когда речь шла о других языках, таких как PHP и ColdFusion. Что же такое сценарный язык?

Большинство программ, работающих на вашем компьютере, написано на языках, называемых *компилируемыми.* Компилирование — это превращение кода, написанного программистом, в инструкции, понятные компьютеру. Когда программа скомпилирована, вы можете запустить ее на вашем компьютере, и, поскольку скомпилированная программа превращена напрямую в инструкции, понятные компьютеру, она будет работать быстрее, чем программа, написанная на сценарном языке. К сожалению, компилирование программы — очень длительный процесс: вы должны написать программу, скомпилировать ее, а затем испытать. Если программа не работает, то потребуется повторить весь процесс.

Сценарный язык, в свою очередь, компилируется, только когда *интерпретатор* (специальная программа, способная превратить сценарий в информацию, понятную компьютеру) читает его. В случае с языком JavaScript интерпретатор встроен в браузер. То есть если ваш браузер читает веб-страницу, в которой есть программа JavaScript, то он сразу же переводит ее в информацию, понятную компьютеру. В результате сценарный язык работает медленнее компилируемого, поскольку программа каждый раз должна быть заново объяснена компьютеру. Сценарные языки отлично подходят для веб-разработчиков, так как сценарии обычно гораздо меньше и проще, чем обычные программы, и медленная скорость не играет решающей роли. Поскольку сценарии не требуют компилирования, процесс создания и тестирования таких программ протекает гораздо быстрее.

Что такое компьютерная программа

Когда вы добавляете код JavaScript на веб-страницу, вы пишете компьютерную программу. Несмотря на то, что большинство программ на языке JavaScript (также называемые *сценариями*) гораздо проще тех, с помощью которых вы читаете почту, ретушируете фотографии и создаете веб-страницы, они обладают многими свойствами более сложных приложений. В принципе, любая компьютерная программа — это серия шагов, выполняемых в заданном порядке. Допустим, вы хотите создать приветственное сообщение, используя в нем имя человека, просматривающего веб-страницу, например, «Добро пожаловать, Иван!» Для выполнения этой задачи ваша программа должна выполнить следующие действия.

1. Запросить у посетителя имя.
2. Получить от него ответ.
3. Напечатать (то есть отобразить) сообщение на веб-странице.

Речь может идти не только о приветствии, просто в этом примере показана основа процесса программирования: определите, что вы хотите сделать, затем разбейте задачу на этапы, прохождение каждого из которых необходимо для выполнения задачи. Создавая программу на языке JavaScript, вы должны определить шаги, необходимые для достижения цели. Как только вы наметите их, вы будете готовы к написанию программы. Другими словами, вы превращаете собственные мысли в программный *код* — слова и символы, заставляющие браузер действовать так, как вам нужно.

Добавление JavaScript на страницу

Браузеры понимают HTML и CSS и превращают эти языки в визуальную информацию на экране. Та часть браузера, которая понимает HTML и CSS, называется браузерным движком. Но большинство браузеров также обладает компонентом под названием *интерпретатор JavaScript*. Это часть браузера, понимающая язык JavaScript и способная выполнять шаги программы JavaScript. Поскольку браузер обычно работает с HTML-кодом, вы должны особым образом сообщить ему, когда появляется JavaScript, используя элемент `script`.

Тег `<script>` — это команда языка HTML, которая работает как переключатель, сообщающий: «Эй, браузер, сейчас появится код JavaScript. Ты не знаешь, что с ним делать, поэтому оставь это интерпретатору JavaScript». Когда браузер встречает закрывающий тег `</script>`, он знает, что достиг конца программы JavaScript и может возвращаться к исполнению своих обычных обязанностей.

Довольно часто вы будете добавлять элемент `script` в заглавную часть веб-страницы — раздел `head`) следующим образом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Моя веб-страница</title>
<script type="text/javascript")
</script>
</head>
```

Атрибут `type` элемента `script` говорит о формате и типе последующего сценария. В данном случае `type="text/javascript"` означает, что сценарий — это обычный текст (как и HTML), и написан он на языке JavaScript.

Если вы используете HTML5, то все гораздо проще. Вы можете просто опустить атрибут `type`:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя веб-страница</title>
<script>
</script>
</head>
```

На самом деле браузеры позволяют опустить атрибут `type` в файлах HTML 4.01 и XHTML 1.0 — сценарий все равно станет работать. Однако в отсутствие атрибута `type` ваша страница не будет проверена должным образом (см. врезку «Проверка веб-страниц»). В приведенных примерах используется HTML5, но код JavaScript будет тем же в случае с HTML 4.01 и XHTML 1.

Добавьте ваш JavaScript-код между открывающим и закрывающим тегами элемента `script`:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя веб-страница</title>
<script>
alert('привет, мир!');
</script>
</head>
```

Через мгновение вы узнаете, что делает этот фрагмент JavaScript-кода. А пока обратите внимание на открывающий и закрывающий теги

элемента `script`. Добавляя на страницу сценарий, начните со вставки именно этих тегов. В большинстве случаев их помещают в раздел заголовка страницы, `head`, чтобы код JavaScript был организован в одной части документа.

Однако вы вправе добавить элемент `script` в любое место вашего HTML-кода. Далее вы узнаете, что существует команда JavaScript, позволяющая записывать информацию непосредственно на веб-страницу. Используя эту команду, вы помещаете элемент `script` в любое место страницы (в рамках элемента `body`) — туда, где должно появиться сообщение сценария. Часто можно встретить элемент `script` сразу под закрывающим тегом `</body>` — данный подход позволяет удостовериться в том, что страница загружена, и посетитель видит ее, прежде чем запускать JavaScript.

УСКОРЯЕМ РАБОТУ

Серверы и клиенты

JavaScript — это *клиентский* язык, то есть он работает внутри браузера. Клиентский JavaScript предоставляется браузерам веб-сервером. Посетители вашего сайта загружают страницу и код JavaScript, после чего их веб-браузер (клиент) обрабатывает этот код.

Другой тип языков веб-программирования — это *серверные* языки (например, PHP, .NET, ASP, ColdFusion, Ruby on Rails). Как можно догадаться из названия, эти языки работают на сервере. Они «проявляют признаки интеллекта», обращаясь к базам данных, обрабатывая кредитные карточки и рассылая электронную почту по всему миру. Проблема серверных языков заключается в том, что при их использовании браузер должен посылать запросы на сервер, заставляя пользователей ждать загрузки новой страницы.

Клиентские языки, напротив, реагируют сразу и изменяют видимое посетителем содержимое в браузере без загрузки новой страницы. Контент может появляться и исчезать, перемещаться по экрану или автоматически обновляться в зависимости от того, что делает посетитель. Эта способность реагировать на действия пользователя позволяет вам создавать сайты, которые ведут себя скорее как настольные приложения, чем как статичные веб-страницы. Однако JavaScript — это не единственная технология, используемая на стороне клиента. Для добавления на веб-страницу логических элементов вы также можете использовать плагины. Например, апплеты

Java — небольшие программы, написанные на языке Java, которые работают в браузере. Они известны медленным запуском и, случается, выводят браузер из строя.

Flash — еще одна технология, основанная на плагинах, которая предлагает сложную анимацию, видео, звуки и огромную потенциальную интерактивность. Зачастую сложно сказать, какая технология используется на данной интерактивной веб-странице: JavaScript или Flash. Например, сервис Google Maps можно было создать с помощью технологии Flash (кстати, сервис Yahoo Maps одно время представлял собой Flash-приложение, пока его не переработали с использованием языка JavaScript). Чтобы понять, какая технология используется на странице, щелкните правой кнопкой мыши по предполагаемому Flash-элементу (в данном случае, по карте). Если это действительно Flash-элемент, то в появившемся контекстном меню будет содержаться пункт **О программе Adobe Flash Player** (About the Flash Player).

Технология Ajax, о которой вы прочитаете в части IV, совмещает черты клиентского и серверного языков. Ajax — это метод, позволяющий использовать JavaScript для общения с сервером, получения с него информации и обновления веб-страницы без необходимости в ее перезагрузке. Сервис Google Maps использует эту технологию, что позволяет вам перемещаться по карте, не ожидая загрузки новой страницы.

На самом деле JavaScript может быть и серверным языком программирования. Например, веб-сервер node.js (**nodejs.org/**) использует JavaScript в качестве серверного языка для обращения к базе данных, получения доступа к файловой системе сервера и произведения многих других действий на сервере. Тем не менее, данный аспект программирования на языке JavaScript не обсуждается в данной книге.

Кроме того, некоторые новые базы данных, например, MongoDB и CouchDB, используют язык JavaScript в качестве языка для создания, извлечения и обновления записей. Вы, вероятно, слышали выражение «полный цикл разработки на JavaScript», которое означает использование JavaScript в качестве языка для управления клиентским браузером, веб-сервером и базой данных.

Внешние файлы JavaScript

Использование элемента `script` так, как описано выше, позволит вам добавить код JavaScript на одну страницу. Однако вам нередко придется создавать сценарии, которые вы захотите использовать

одновременно на всех страницах сайта. Например, вы можете написать программу JavaScript для добавления на веб-страницу анимированных выпадающих навигационных меню (рис. 1.2). Вам нужно, чтобы они появлялись на каждой из страниц вашего сайта. Но просто копировать и вставлять один и тот же код JavaScript на каждую из страниц сайта — это плохая идея.

Во-первых, вы потратите много времени, копируя и вставляя один и тот же код снова и снова, особенно если ваш сайт содержит сотни страниц. Во-вторых, если вы когда-нибудь решите изменить или улучшить код JavaScript, то должны будете найти каждую веб-страницу, использующую этот код, и обновить его. Наконец, поскольку весь код программы JavaScript будет находиться на каждой отдельной веб-странице, они окажутся достаточно большого размера и, естественно, станут медленнее загружаться.

Гораздо лучший подход — это применение внешнего файла JavaScript. Если вы использовали внешние файлы CSS для своих веб-страниц, то эта техника должна быть вам знакома. Внешний файл JavaScript — это просто текстовый файл с расширением *.js* (например, *navigation.js*). Он содержит только код JavaScript и связан с веб-страницей с помощью элемента `script`. Например, для добавления файла JavaScript с именем *navigation.js* на вашу домашнюю страницу напишите следующее:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя веб-страница</title>
<script src="navigation.js"></script>
</head>
```

Атрибут `src` элемента `script` работает так же, как атрибут `src` элемента `img` или атрибут `href` элемента `a`. Другими словами, он указывает на файл, находящийся либо на вашем, либо на другом сайте (см. врезку «Типы URL-адресов»).

ПРИМЕЧАНИЕ

Добавляя атрибут `src` к ссылке на внешний файл JavaScript, не записывайте какой-либо JavaScript-код между открывающим и закры-

вающим тегами элемента `script`. Если вы хотите сделать ссылку на внешний файл JavaScript и добавить еще один фрагмент кода JavaScript на страницу, используйте вторую пару тегов элемента `script`. Например:

```
<script src="navigation.js"></script>
<script>
alert('Привет, мир!');
</script>
```

Вы можете (и часто будете) прикреплять несколько внешних файлов JavaScript к одной веб-странице. Например, вы создали внешний файл JavaScript, который контролирует выпадающую панель навигации, и файл, позволяющий добавлять стильные слайд-шоу на страницу с фотографиями. На странице с фотогалереей понадобятся обе программы JavaScript, поэтому вы прикрепите оба файла.

Вы можете прикрепить внешние файлы JavaScript и добавить программу JavaScript к одной и той же странице следующим образом:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя веб-страница</title>
<script src="navigation.js"></script>
<script src="slideshow.js"></script>
<script>
alert('привет, мир!');
</script>
</head>
```

Помните, что вы должны использовать открывающий и закрывающий теги элемента `script` для каждого внешнего файла JavaScript. Вы создадите внешний файл JavaScript по инструкции в разделе «Прикрепление внешнего файла JavaScript».

Вы можете хранить внешние файлы JavaScript где угодно в корневом каталоге вашего веб-сайта (или в подкаталогах корневого каталога). Многие веб-разработчики создают специальный каталог для внешних

файлов JavaScript в корневом каталоге на сайте, обычно под названием *js* (то есть JavaScript) или *libs* (то есть библиотеки).

Рис. 1.2. На сайте Nike.com код JavaScript используется для эффективного представления продуктов компании. На домашней странице присутствует ряд навигационных кнопок, при наведении указателя мыши на которые отображаются дополнительные варианты. Например, при наведении указателя мыши на кнопку «Виды спорта» отображается панель, в которой перечислены виды спорта, для которых компания Nike создает продукты

УСКОРЯЕМ РАБОТУ

Типы URL-адресов

Прикрепляя внешний файл JavaScript к веб-странице, вам следует указать *URL-адрес* для атрибута `src` элемента `script`. URL или *унифицированный указатель ресурса*, — это путь к файлу, расположенному во Всемирной паутине, то есть его адрес. Есть три вида адресов: *абсолютный*, *относительно корневого каталога* и *относительно документа*. Все они указывают браузеру, где он может найти конкретный файл (например, другую веб-страницу, изображение или файл JavaScript).

Абсолютный адрес похож на почтовый: он содержит всю информацию, необходимую браузеру, чтобы найти файл, где бы он ни находился. Абсолютный путь содержит http://, имя сервера, имя каталога и имя файла. Например: http://www.uptospeedguides.com/scripts/site.js.

Путь относительно корневого каталога указывает, где находится файл по отношению к корневому каталогу — наивысшему каталогу в иерархии сайта. Путь относительно корневого каталога не имеет http:// или доменного имени. Он начинается с символа / (слеша), обозначающего корневой каталог сайта — каталог, в котором находится домашняя страница. Например, адрес /scripts/site.js сообщает, что файл *site.js* расположен в каталоге с именем *scripts*, который находится в каталоге высшего уровня. Простой способ создать путь относительно корневого каталога — это взять абсолютный адрес и отбросить http:// и имя сервера. Например, путь http://www.uptospeedguides.com/index.html, записанный как ссылка относительно корневого каталога, — это */index.html*.

Путь относительно документа показывает путь от веб-страницы к файлу JavaScript. Если на вашем сайте много уровней и каталогов, вы должны использовать различные пути, чтобы указать на один и тот же файл JavaScript. Например, представьте, что у вас есть файл JavaScript с именем *site.js*, размещенный в каталоге *scripts,* в главном каталоге вашего сайта. Путь относительно документа для этого файла будет одним для домашней страницы: script/site.js, но другим для страницы, находящейся внутри каталога *about*: ../scripts/site.js. Символы ../ означают подъем на уровень выше каталога *about*, тогда как /scripts/site.js означает переход к каталогу *scripts* и к файлу *site.js*.

Ниже представлено несколько советов по поводу того, какой тип URL-адреса лучше использовать:

- Если вы указываете на файл, находящийся на сервере, отличном от того, на котором расположена веб-страница, *необходимо* использовать абсолютный адрес. Это единственный тип URL-адреса, который может указывать на другой сайт.
- Пути относительно корневого каталога подходят для файлов JavaScript, хранящихся на вашем собственном сайте. Поскольку такие адреса всегда начинаются с указания корневого каталога, URL-адрес файла JavaScript будет одинаковым для всех страниц вашего сайта, даже если они расположены в каталогах и подкаталогах сайта. Однако пути относительно корневого каталога работают только в том случае, если вы просматриваете веб-страницы, либо через ваш сервер в Интернете, либо через веб-сервер, который вы настроили на собственном компьютере для целей тестирования. Другими словами, если вы просто открываете веб-страницу с вашего компьютера, используя команду меню **Файл** ⇒ **Открыть** (File ⇒ Open), браузер не сможет определить местонахождение файлов JavaScript, путь к которым относится к корневому каталогу, а также загрузить или запустить их.

- Пути относительно документа являются наилучшим выбором, если вы занимаетесь дизайном на собственном компьютере, не обращаясь к серверу. Вы можете создать внешний файл JavaScript, прикрепить его к веб-странице, а потом просмотреть в браузере, просто открыв веб-страницу с жесткого диска. Пути относительно документа отлично работают, будучи помещенными на «живой» рабочий сайт в Интернете, но вам придется переписать ссылки на файл JavaScript, если вы переместите веб-страницу в другое место на сервере. В этой книге мы станем использовать пути относительно документа, поскольку они позволят вам протестировать руководства на собственном компьютере, не обращаясь к веб-серверу.

ПРИМЕЧАНИЕ

Иногда имеет значение порядок, в котором вы прикрепляете внешние файлы JavaScript. Как вы узнаете далее, иногда сценарии, которые вы пишете, зависят от кода, находящегося во внешнем файле. Часто это имеет место при использовании библиотек JavaScript (JavaScript-кода, упрощающего решение сложных задач программирования). Пример использования библиотеки JavaScript вы найдете в разделе «Прикрепление внешнего файла JavaScript».

Ваша первая программа на языке JavaScript

Лучший способ освоить программирование на JavaScript — начать программировать. В этой книге вы найдете инструкции, с помощью которых сможете шаг за шагом создавать программы на этом языке. Чтобы начать, вам понадобится текстовый редактор (см. рекомендации в разделе «Обеспечение для программирования на языке JavaScript»), браузер и файлы примеров, доступные на сайте издательства «Эксмо» (см. следующее примечание).

ПРИМЕЧАНИЕ

Для выполнения приведенных здесь инструкций вам необходимо загрузить файлы примеров по адресу www.eksmo.ru. После того как вы загрузили и распаковали архив, на вашем компьютере должна появиться папка, в которой содержатся все файлы к данной книге.

Чтобы JavaScript с самого начала не испугал вас, первая программа, которую мы рассмотрим, будет очень простой.

1. **В вашем любимом текстовом редакторе откройте файл *01_01.html*.**

Данный файл расположен в каталоге *глава01*, который находится в скопированной вами папке с примерами. Это очень простая HTML-страница с внешней каскадной таблицей стиля.

2. **Щелкните в пустой строке *перед* закрывающим тегом `</head>` и напечатайте**:

```
<script>
```

Этот HTML-код сообщает браузеру о том, что за данным тегом последует JavaScript.

3. **Нажмите клавишу Enter, чтобы добавить пустую строку, и напечатайте в ней следующее:**

```
alert('привет, мир');
```

Вы только что создали свою первую строку JavaScript-кода. Функция JavaScript `alert()` — это команда, вызывающая окно оповещения и отображающая сообщение, находящееся в скобках, — в данном случае: `привет, мир`. Пока не задумывайтесь о пунктуации (скобки, кавычки и точка с запятой). В следующей главе вы узнаете, зачем они нужны.

4. **Еще раз нажмите клавишу Enter и напечатайте `</script>`. Теперь код должен выглядеть так:**

```
<link href="../_css/site.css" rel="stylesheet">
<script>
alert('привет, мир');
</script>
</head>
```

В данном примере напечатанные вами символы выделены полужирным шрифтом. Два HTML-тега уже присутствуют в файле, поэтому убедитесь, что печатаете код именно там, где показано.

5. **Запустите браузер и откройте файл *01_01.html* для предварительного просмотра.**

Откроется окно оповещения JavaScript (рис. 1.3). Обратите внимание, что при появлении этого окна страница пуста. Если окно оповещения, изображенное на рис. 1.3, не появилось, возможно, вы опечатались, выполняя шаги инструкции. Перепроверьте код и прочитайте приведенный ниже совет.

СОВЕТ

Если вы только начали программировать, то будете шокированы тем, как часто ваши программы на JavaScript станут отказываться работать. Чаще всего причиной этого служат простые опечатки. Всегда проверяйте, правильно ли вы набрали команды (например, `alert` в первом сценарии). Обратите также внимание на то, что пунктуация часто бывает парной (например, открывающие и закрывающие скобки и одинарные кавычки в вашем первом сценарии). Убедитесь, что вы напечатали все закрывающие знаки пунктуации.

6. Щелкните по кнопке OK в окне оповещения, чтобы закрыть его.

Когда окно исчезнет, в браузере появится веб-страница.

Хотя эта первая программа не такая уж и сложная, она демонстрирует важную концепцию: браузер выполняет программу на JavaScript одновременно с чтением JavaScript-кода. В этом примере функция `alert()` сработала *до того*, как браузер отобразил веб-страницу, поскольку код JavaScript *предшествовал* HTML-коду в элементе `body`. Подобная концепция будет иметь значение, когда вы начнете писать программы, манипулирующие HTML-кодом веб-страницы в главе 3.

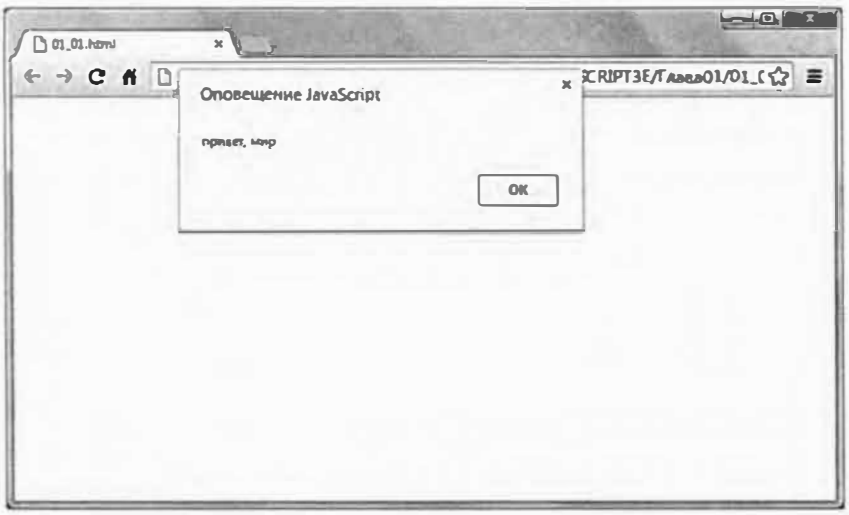

Рис. 1.3. Окно оповещения JavaScript — это быстрый способ привлечь чье-либо внимание. Это одна из простейших команд JavaScript

ПРИМЕЧАНИЕ

Некоторые версии браузера Internet Explorer не любят запускать JavaScript программы, содержащиеся на веб-страницах, которые вы открываете непосредственно с жесткого диска, из-за опасения, что данная программа может навредить компьютеру. Поэтому когда вы пытаетесь просмотреть учебные файлы в браузере Internet Explorer, вы можете увидеть сообщение, говорящее вам о том, что браузер заблокировал сценарий. Щелкните по кнопке **Разрешить заблокированное содержимое** (Allow blocked content), чтобы запустить программу.

Эта раздражающая особенность проявляется только по отношению к веб-страницам, которые вы просматриваете со своего компьютера, но не к страницам, которые вы размещаете на сервере. Прорабатывая руководства в данной книге, используйте разные браузеры, например, Firefox, Chrome или Safari, чтобы избежать необходимости щелкать по кнопке **Разрешить заблокированное содержимое** (Allow blocked content) каждый раз при просмотре страниц.

Публикация текста на веб-странице

Рассмотренный нами сценарий заставил диалоговое окно появиться на мониторе вашего компьютера. Но что делать, если вы хотите написать сообщение прямо на веб-странице, используя JavaScript? Это можно сделать разными способами, и далее в книге вы найдете несколько интересных техник. Однако вы можете решить эту простую задачу с помощью встроенной команды JavaScript, что вы сейчас и сделаете:

1. **В текстовом редакторе откройте файл *01_02.html*.**

Элемент `script` обычно указывается внутри раздела заголовка, однако вы можете разместить их и программу JavaScript в теле (элементе `body`) веб-страницы.

2. **Прямо под строкой `<h1>Написание текста в окне документа</h1>` напечатайте следующий код:**

```
<script>
document.write('<p>Привет, мир!</p>');
</script>
```

Подобно функции `alert()`, `document.write()` — это команда JavaScript, которая буквально отображает то, что вы помещаете между открывающей и закрывающей скобками. В данном случае на веб-страницу добавляется HTML-код, состоящий из элемента абзаца и двух слов: `<p>Привет, мир!</p>`.

3. Сохраните страницу и откройте ее в браузере.

На открытой странице фраза «Привет, мир!» находится прямо под заголовком (рис. 1.4).

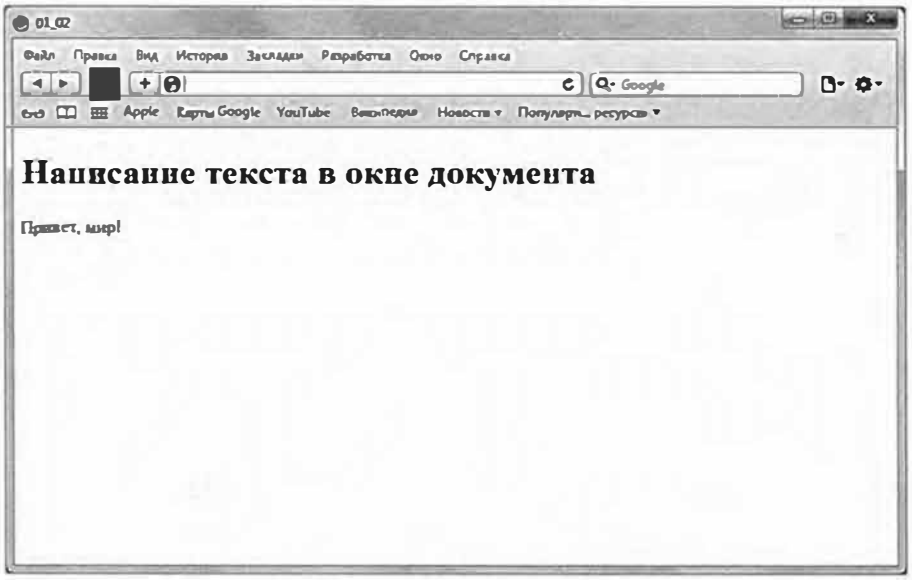

Рис. 1.4. Этот сценарий показывает, как вы можете использовать JavaScript для добавления контента на веб-страницу — прием, который может пригодиться, если вам потребуется отобразить сообщение (вроде «С возвращением на сайт, Иван») после загрузки веб-страницы

ПРИМЕЧАНИЕ

В скопированной вами папке с учебными файлами есть готовые версии всех примеров. Если ваш JavaScript-код не работает, сравните его с кодом из файла, имя которого начинается со слова *готовый_* и который хранится в той же папке, что и исходный файл примера. Например, файл *готовый_01_02.html* содержит рабочую версию сценария, добавленного в файл *01_02.html*.

Два написанных вами сценария могут оставить ощущение несколько неоправданных ожиданий от языка JavaScript... или этой книги. Не волнуйтесь. Вначале важно полностью постичь основы. Прочитав еще несколько глав, вы сможете делать сложные и полезные вещи с помощью JavaScript. В оставшейся части этой главы вы узнаете о несколь-

ких расширенных функциях, которые вы сможете добавить на свои веб-страницы, после освоения первых двух частей данной книги.

Прикрепление внешнего файла JavaScript

Как уже говорилось, код JavaScript обычно помещается в отдельный файл в случае, если вы хотите использовать его более чем на одной веб-странице. Затем вы приказываете веб-странице загрузить этот файл и выполнить содержащийся в нем JavaScript-код. Внешние файлы JavaScript также могут пригодиться, когда вы используете чужой код JavaScript. В частности, существуют коллекции кодов JavaScript, называемые *библиотеками*, они содержат полезные материалы для программирования на языке JavaScript. Обычно эти библиотеки облегчают решение сложных задач. Подробнее о библиотеках, и в частности о библиотеке jQuery, вы прочитаете в главе 4.

А сейчас вы научитесь прикреплять внешний файл JavaScript к странице и напишите небольшую программу, которая делает нечто удивительное:

1. **В текстовом редакторе откройте файл 01_03.html.**

Данная страница содержит простой HTML-код — несколько элементов `div`, заголовок и два абзаца. Вам предстоит добавить на страницу простой визуальный эффект, который заставит содержимое медленно проявиться.

2. **Щелкните в пустой строке между тегом `<link>` и закрывающим тегом `</head>` в верхней части страницы и напечатайте:**

```
<script src="../_js/jquery.min.js"></script>
```

Этот код связывает файл *jquery.min.js*, который находится в каталоге *_js* с данной веб-страницей. Когда браузер загружает веб-страницу, он также загружает JavaScript-файл *jquery.min.js* и выполняет содержащийся в нем код.

Далее вы добавите на эту страницу вашу собственную JavaScript-программу.

ПРИМЕЧАНИЕ

Слово *min* означает, что код в файле *минимизирован*, что делает его лаконичным и ускоряет его загрузку.

3. **Нажмите клавишу Enter, чтобы создать новую пустую строку, и напечатайте:**

```
<script>
```

Как вы уже знаете, HTML-элементы обычно «ходят» парой — содержат открывающий и закрывающий тег. Чтобы не забыть закрыть элемент, полезно ставить закрывающий тег сразу после открывающего, а затем набирать код между ними.

4. **Дважды нажмите клавишу Enter, чтобы создать две пустые строки и напечатайте**:

```
</script>
```

Так заканчивается блок кода JavaScript. Теперь давайте добавим программу.

5. **Щелкните на пустой строке между открывающим и закрывающим тегами элемента `script` и напечатайте:**

```
$(document).ready(function() {
```

Вы, вероятно, недоумеваете относительно того, что же это такое. Детальное описание этого кода вы найдете в разделе «Добавление библиотеки jQuery на страницу» главы 4. Если говорить кратко, то данная строка использует программу, находящуюся в файле *jquery.min.js*, чтобы браузер выполнил следующую строку кода в нужное время.

6. **Нажмите клавишу Enter, чтобы создать новую строку, и напечатайте:**

```
$('header').hide().slideDown(3000);
```

Эта строка творит волшебство: она заставляет содержимое страницы сначала исчезнуть, а затем медленно проявиться в течение 3 секунд (или 3000 миллисекунд). Как она это делает? Ну, это часть магии библиотеки jQuery, которая позволяет создавать сложные эффекты с помощью одной лишь строки кода.

7. **Еще раз нажмите клавишу Enter и введите:**

```
});
```

Этот фрагмент завершает код JavaScript и, подобно закрывающему тегу `</script>`, означает конец программы. Не беспокойтесь обо всей этой странной пунктуации, вы еще поймете, как она работает. Сейчас ваша основная задача — записать код так, как показано в примере. Пом-

ните, что если вы допустите хоть одну опечатку, то программа может не заработать.

Окончательный код, который вы должны были добавить на страницу, выделен полужирным шрифтом:

```
<link href="../_css/site.css" rel="stylesheet">
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('header').hide().slideDown(3000);
});
</script>
</head>
```

ПРИМЕЧАНИЕ

Для того чтобы сделать код программы более легким для чтения, следует добавлять отступы для некоторых строк. Вы делаете отступы для HTML-элементов, чтобы показать, какие из них вложены в другие элементы. Точно также вы можете добавить отступы для строк кода, находящихся в других блоках кода. Например, строка кода, которую вы добавили в шаге 6, вложена в блок кода, введенного на шагах 5 и 7, поэтому если вы воспользуетесь клавишей **Tab** или **Пробел**, прежде чем вводить фрагмент на шаге 6, ваш код будет легче воспринимать. (В данной книге отступы не используются для экономии места.)

8. Сохраните HTML-файл и откройте его в браузере.

Вы должны увидеть, как содержимое страницы медленно проявляется. Измените значение с 3000 на 250 или 10000, чтобы посмотреть, как это повлияет на работу страницы.

ПРИМЕЧАНИЕ

Если вы пытаетесь просмотреть эту страницу в браузере Internet Explorer, и ничего не происходит, то вам нужно щелкнуть по кнопке **Разрешить заблокированное содержимое** (Enable blocked content) внизу страницы (см. примечание в разделе «Ваша первая программа на языке JavaScript» ранее в этой главе).

Как видите, даже небольшие фрагменты кода JavaScript способны сотворить с вашей страницей удивительные вещи. Благодаря библиотекам JavaScript, например, jQuery, вы сможете создавать сложные интерактивные веб-страницы, не будучи профессиональным программистом. Однако для этого вам нужно еще многое узнать. В двух следующих главах вы изучите основы языка JavaScript и познакомитесь с его фундаментальными концепциями и синтаксисом.

Отслеживание ошибок

Самый неприятный момент наступает тогда, когда вы пытаетесь просмотреть в браузере страницу, содержащую код JavaScript... и ничего не происходит. Это распространенное явление. Даже опытные программисты не всегда обходятся без ошибок при первом написании программы, поэтому определение того, что было сделано неправильно, — это неотъемлемая часть работы.

Большинство браузеров настроено на тихое игнорирование ошибок JavaScript, поэтому зачастую вы даже не увидите диалогового окна «Ой, эта программа не работает!» (В принципе, это неплохо, поскольку вы не хотите, чтобы ошибка JavaScript прерывала просмотр ваших веб-страниц.)

Итак, как определить, что работает неправильно? Есть много способов отследить ошибки в программе JavaScript. Вы изучите некоторые расширенные техники *отладки* в главе 17, но самый простой способ — это обратиться к браузеру. Большинство браузеров отслеживает ошибки JavaScript и записывает их в отдельном окне, которое называется *консолью ошибок*. Когда вы загрузите веб-страницу, содержащую ошибку, вы сможете открыть консоль для получения полезной информации о данной ошибке, например, ее описания, а также того, в какой строке веб-страницы она находится.

В большинстве случаев вы сможете найти ответ в консоли ошибок и исправить код JavaScript так, чтобы страница работала правильно. Консоль поможет вам «отловить» основные опечатки, которые вы непременно сделаете, начав программировать, например, пропуски закрывающих знаков пунктуации или неправильно введенные имена команд JavaScript. Вы можете использовать консоль ошибок своего любимого браузера, но, поскольку сценарии не одинаково работают в различных браузерах, в этом разделе вам будет показано, как пользоваться консо-

лью JavaScript во всех основных браузерах, так что вы сможете отслеживать ошибки, применяя любой из них.

Консоль JavaScript в браузере Chrome

Браузер Chrome компании Google пользуется большой популярностью среди веб-разработчиков. Предусмотренные в данной программе инструменты позволяют решить множество проблем, связанных с кодом HTML, CSS и JavaScript. Кроме того, консоль JavaScript помогает вам отслеживать ошибки в вашем коде. В ней не только описаны найденные ошибки, но и указаны номера строк кода, в которых они обнаружены.

Чтобы открыть консоль, щелкните по значку **Настройка** (Customize) (который обведен на рис. 1.5), выберите в меню **Инструменты** (Tools) команду **Консоль JavaScript** (JavaScript Console) или используйте сочетание клавиш **Ctrl+Shift+J** (Windows) или **⌘+⌥+J** (OS X).

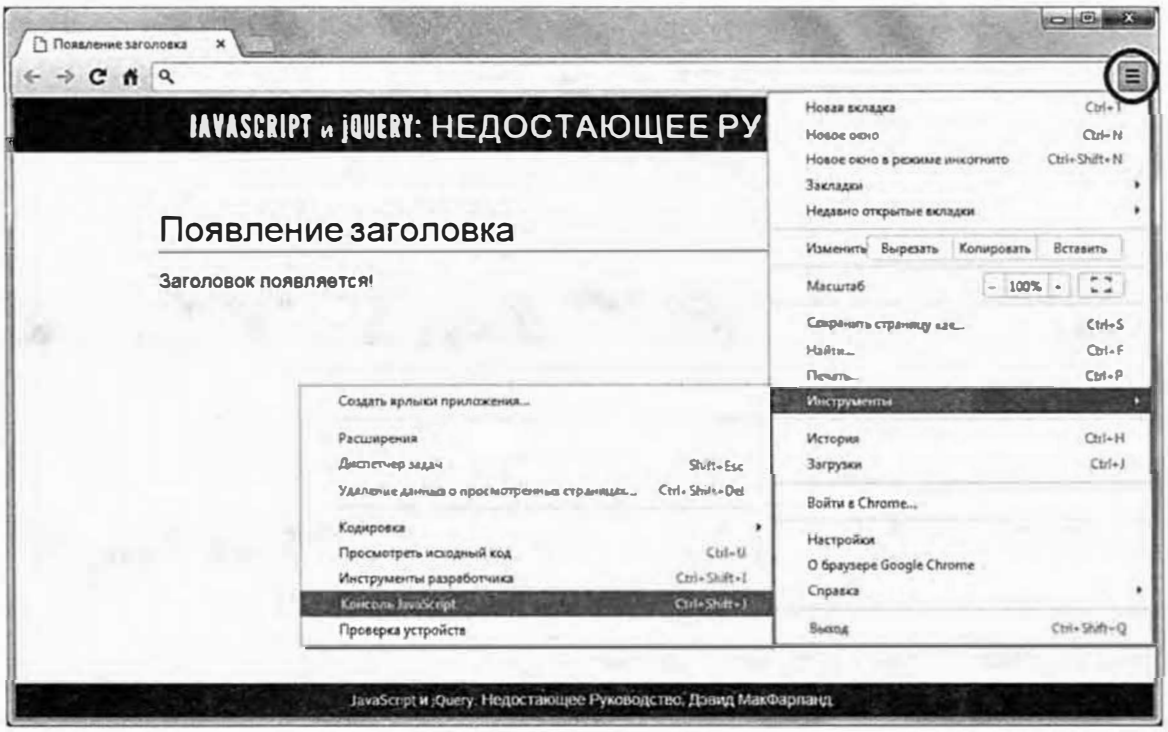

Рис. 1.5. Щелкните по значку **Настройка** (Customize) (выделен на рисунке), чтобы получить доступ к консоли JavaScript и другим полезным инструментам. Вы также можете выбрать пункт **Инструменты разработчика** (Developer Tools), поскольку консоль JavaScript является частью этого более обширного набора инструментов, о котором вы узнаете подробнее в главе 17

Как только консоль откроется, вы сможете просмотреть информацию о найденных на странице ошибках. Например, на рис. 1.6 показа-

но сообщение об ошибке `Uncaught SyntaxError: Unexpected token ILLEGAL`, которое является не совсем понятным, однако со временем вы привыкнете к таким описаниям. По сути, ошибка синтаксиса связана с какой-то опечаткой — ошибкой в коде программы. Строка `Unexpected token ILLEGAL` означает, что браузер обнаружил недопустимый или (что сложнее) пропущенный символ. В данном случае, если вы внимательно просмотрите код, то увидите, что перед словом `slow` поставлена открывающая кавычка, а закрывающая кавычка отсутствует.

В консоли также указано имя файла с ошибкой (*01_03.html*) и номер строки (line 10). Щелкните мышью по имени файла, чтобы открыть его и увидеть выделенную строку с ошибкой (см. рис. 1.6).

▶ СОВЕТ

Поскольку консоль отображает номер строки с ошибкой, вам может показаться целесообразным использовать текстовый редактор, в котором указывается номер строки. Это поможет быстро перейти из консоли ошибок в текстовый редактор и определить, какую строку кода следует исправить.

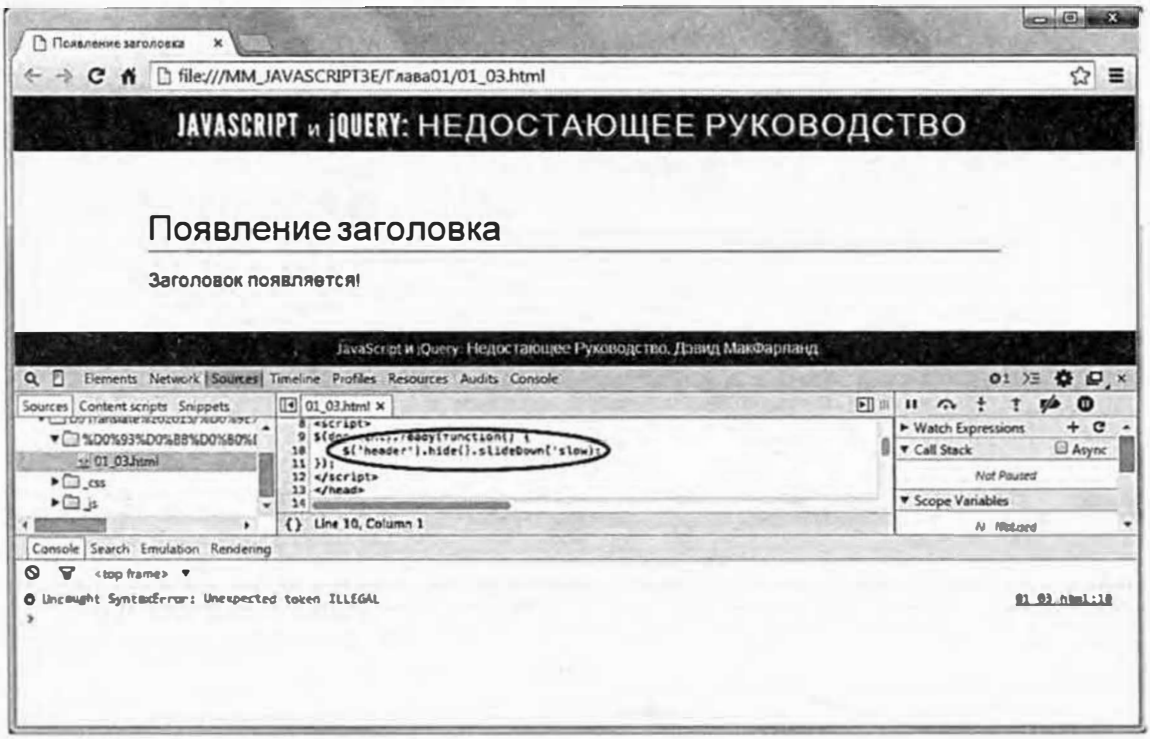

Рис. 1.6. Консоль JavaScript браузера Chrome отображает ошибки в ваших программах. Щелкните мышью по имени файла справа от описания ошибки, чтобы открыть его и увидеть выделенную строку с ошибкой (выделена на рисунке)

К сожалению, список причин, по которым сценарий может работать неправильно, очень велик: от тривиальных опечаток до логических ошибок. Если вы только начинаете программировать на языке JavaScript, то большинство ваших ошибок будут относиться именно к опечаткам. Например, вы можете забыть поставить точку с запятой, кавычку, круглые скобки или неправильно записать команду. Будьте особенно внимательны, воспроизводя примеры из книги. Рассмотрим некоторые ошибки, которые вы будете часто допускать, переписывая код.

- **Пропущенный символ пунктуации.** Как говорилось ранее, программы JavaScript включают множество пар символов, например открывающих и закрывающих круглых или квадратных скобок. Если вы введете `alert('привет';`, забыв поставить круглую скобку, то, вероятно, увидите сообщение `Unexpected token ;`, которое означает, что браузер Chrome ожидал обнаружить другие символы. В данном случае программе встретилась точка с запятой вместо закрывающей круглой скобки.

- **Отсутствие кавычек.** *Строка* — это серия символов, заключенных в кавычки (см. раздел «Строки» главы 2). Например, `'привет'` — это строка в коде `alert('привет');`. В данном случае легко забыть открывающую или закрывающую кавычку. Также легко перепутать кавычки, например, вместо одинарной использовать двойную: `alert('привет");`. В любом случае вы, вероятно, увидите сообщение об ошибке `Uncaught SyntaxError: Unexpected token ILLEGAL`.

- **Опечатки в командах.** Если вы неправильно запишете команду JavaScript, например, `aler('привет');`, то получите сообщение об ошибке, из которого узнаете, что неправильно записанная команда не определена, например, `Uncaught ReferenceError: aler is not defined`. У вас также возникнут проблемы при неправильном вводе jQuery-функций, например, используемых ранее `.hide()` и `.slideDown()`. В этом случае возникнет другая ошибка. Если бы вы ввели `hid` вместо `hide` в шаге 6 в разделе «Прикрепление внешнего файла JavaScript» ранее в этой главе, то программа Chrome отобразила бы сообщение `Uncaught TypeError: Object [object Object] has no method 'hid'`.

- **Синтаксическая ошибка.** Иногда браузер Chrome не понимает, что вы хотите сделать, и выводит общее сообщение об ошибке. Синтаксическая ошибка представляет какую-то ошибку в вашем коде. Речь не только об опечатках. Вы можете случайно неправильно упо-

требить одно или несколько инструкций JavaScript. В этом случае нужно проанализировать указанную строку и попытаться отыскать ошибку. К сожалению, для исправления подобных ошибок часто необходим опыт и глубокое понимание языка JavaScript.

Из приведенного выше списка можно сделать вывод, что многие ошибки связаны с обычной невнимательностью, вследствие чего вы можете пропустить один из двух знаков пунктуации, например, кавычки или круглые скобки. К счастью, исправить такие неточности легко, и, приобретя опыт программирования, вы будете допускать их реже и реже (избавиться от подобных ошибок полностью не может ни один программист).

Консоль JavaScript в браузере Internet Explorer

Браузер Internet Explorer предоставляет обширный набор инструментов, позволяющий не только просматривать ошибки JavaScript, но и анализировать CSS, HTML и процесс передачи информации через сеть. Открытое окно **Средства разработчика** (Developer tool) занимает нижнюю половину окна браузера.

Нажмите клавишу **F12** один раз, чтобы открыть окно **Средства разработчика** (Developer tool) и еще раз, чтобы закрыть его. Вы увидите, что ошибки JavaScript перечислены во вкладке **Консоль** (Console) (обведена на рис. 1.7).

ПРИМЕЧАНИЕ

Если вы сначала откроете веб-страницу, а затем консоль Internet Explorer, вы не увидите никаких ошибок, даже если они присутствуют в коде. Вам необходимо перезагрузить страницу. После открытия консоли вы сможете увидеть присутствующие на странице ошибки по мере ее загрузки.

Консоль браузера Internet Explorer отображает сообщения об ошибках подобно консоли браузера Chrome, описанной ранее. Например, сообщение **Незавершенная строковая константа** (Unterminated string constant) равнозначно сообщению консоли браузера Chrome `Unexpected token ILLEGAL`. Как и Chrome, программа Internet Explorer определяет строку HTML-файла, в которой допущена ошибка. Вы можете щелкнуть по ней для отображения кода.

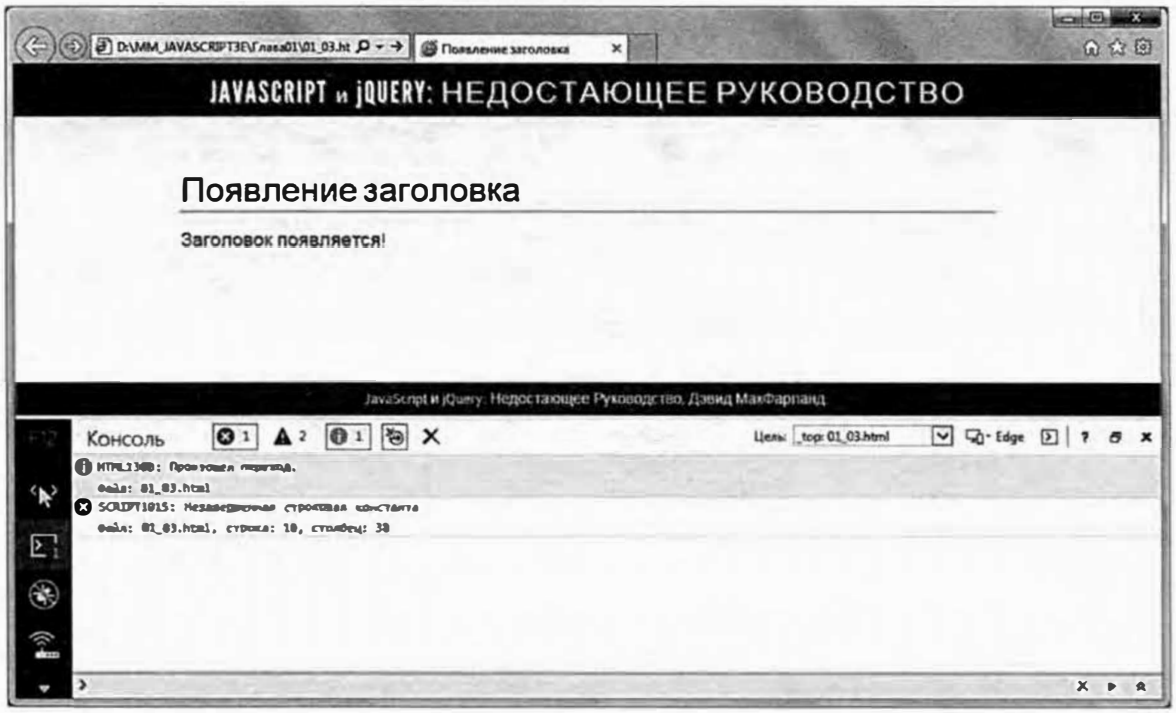

Рис. 1.7. Средства разработчика браузера Internet Explorer предоставляют доступ к ошибкам JavaScript, содержащимся на странице, а также к другой полезной информации

Консоль JavaScript в браузере Firefox

Браузер Firefox компании Mozilla также предусматривает консоль JavaScript для отслеживания ошибок в коде. Чтобы отобразить консоль JavaScript в версии браузера для операционной системы Windows, выберите в меню **Инструменты** ⇒ **Веб-разработка** (Tools ⇒ Web Developer) команду **Веб-консоль** (Web Console) или нажмите сочетание клавиш **Ctrl+Shift+I**. В операционной системе OS X выберите аналогичную команду или нажмите сочетание клавиш **⌘+⌥+K** (OS X).

После открытия консоли вы увидите присутствующие на странице ошибки. К сожалению, консоль Firefox содержит слишком большой объем данных (рис. 1.8). Это связано с тем, что она предоставляет сведения о загруженных файлах, ошибках CSS и HTML и т. д.

ПРИМЕЧАНИЕ

Плагин Firebug (**getfirebug.com**) расширяет возможности консоли ошибок данного браузера. Эта программа послужила моделью для других инструментов разработчика, которые можно найти в браузерах Internet Explorer, Chrome и Safari (обсуждаемых далее).

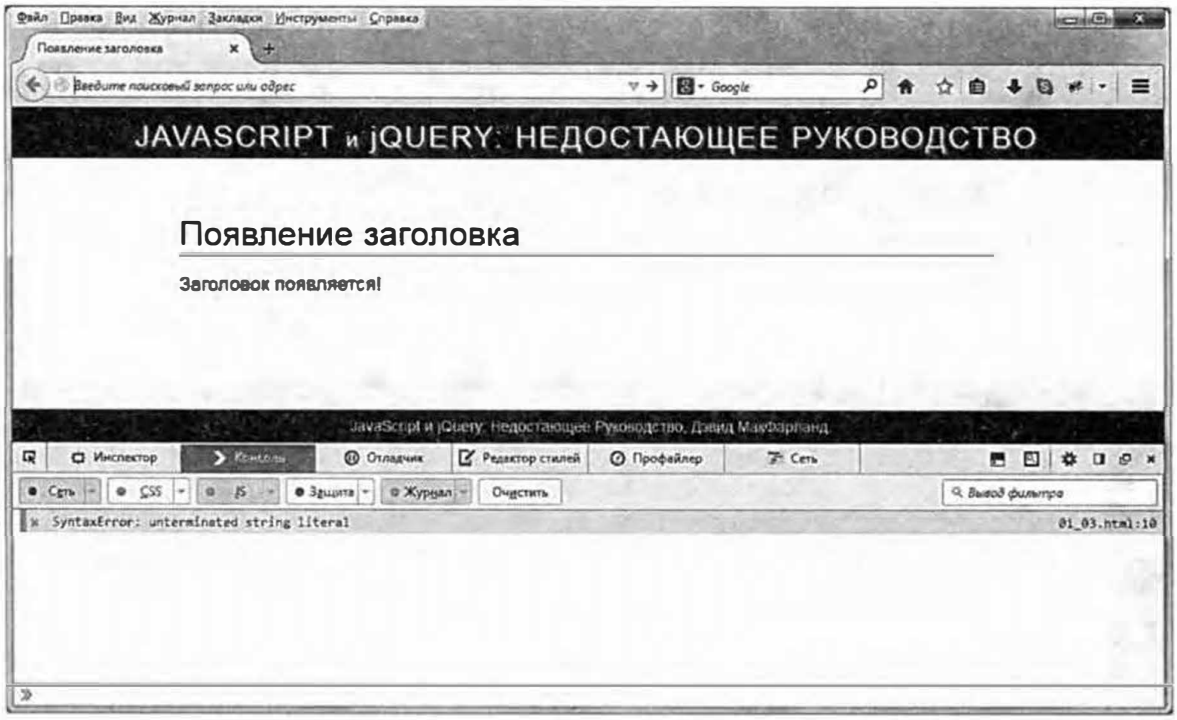

Рис. 1.8. Если вы не хотите просматривать все сообщения в консоли Firefox, просто щелкните по кнопке, соответствующей типу сообщений, которые вам нужно скрыть. Например, щелкните по кнопке **CSS**, чтобы скрыть сообщения об ошибках CSS, или по кнопке **Защита** (Security), чтобы скрыть соответствующие уведомления. Отключенная кнопка окрашена в светло-серый цвет, как кнопки **CSS** и **Защита** (Security) в данном примере. Активная кнопка выделена более темным цветом и выглядит «нажатой», например, кнопки **Сеть** (Net), **JS** и **Журнал** (Logging) на приведенном рисунке

Консоль JavaScript в браузере Safari

Консоль ошибок в браузере Safari доступна с помощью команды меню **Разработка ⇒ Показать консоль ошибок** (Develop ⇒ Show Error Console) (или воспользуйтесь сочетанием клавиш **⌥+⌘+C**, а в операционной системе Windows — **Ctrl+Alt+C**). Однако следует помнить, что меню **Разработка** (Develop) в браузере Safari обычно отключено, поэтому, прежде чем перейти к консоли ошибок JavaScript, необходимо выполнить некоторые операции.

Чтобы отобразить меню **Разработка** (Develop), вам нужно вызвать диалоговое окно **Настройки** (Preferences). В операционной системе OS X выберите в меню **Safari** пункт **Настройки** (Preferences). В операционной системе Windows щелкните по значку основных настроек в верхнем правом углу браузера и в появившемся меню выберите пункт **Настройки** (Preferences). В открывшемся диалоговом окне **Настройки** (Preferences) щелкните по кнопке **Дополнения** (Advanced). Установите флажок **По-**

казывать меню «Разработка» в строке меню (Show Develop menu in menu bar) и закройте окно **Настройки** (Preferences).

Когда вы перезапустите браузер Safari, пункт **Разработка** (Develop) появится между меню **Закладки** (Bookmarks) и **Окно** (Window) в верхней части экрана. Выберите команду меню **Разработка ⇒ Показать консоль ошибок** (Develop ⇒ Show Error Console), чтобы открыть консоль ошибок (рис. 1.9).

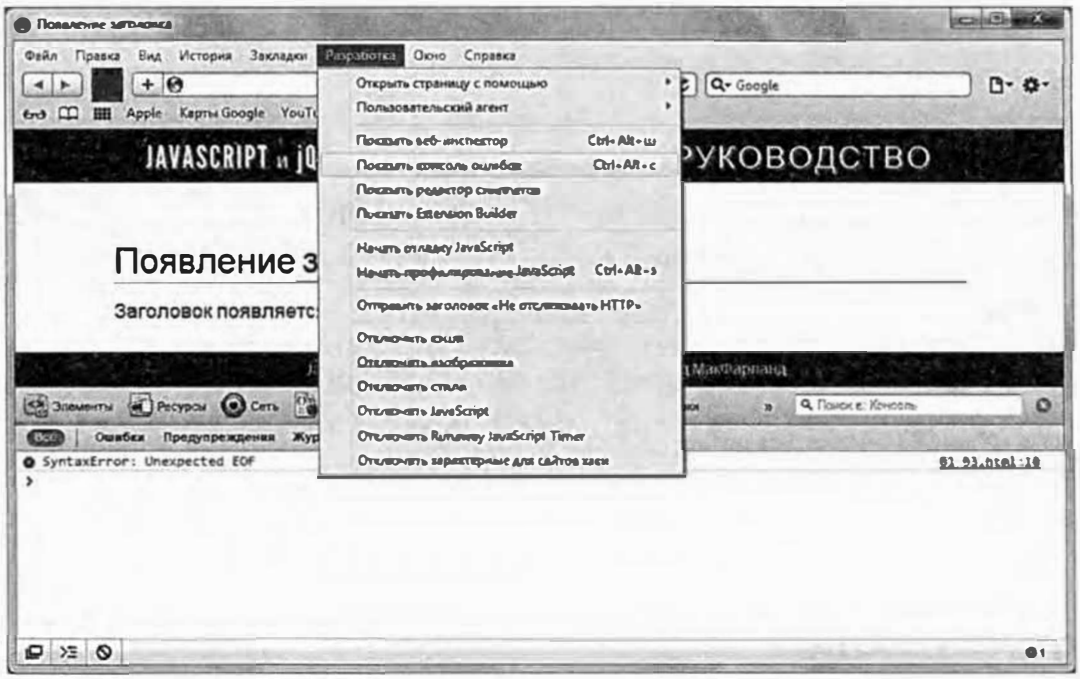

Рис. 1.9. Консоль ошибок в браузере Safari отображает название ошибки, имя файла и путь к нему, а также строку, в которой браузер нашел ошибку. Каждая вкладка или окно браузера имеет собственную консоль ошибок, поэтому если вы уже открыли консоль для одной вкладки, вам придется выбрать команду меню **Разработка ⇒ Показать консоль ошибок** (Develop ⇒ Show Error Console), если вы хотите увидеть ошибку, касающуюся другого окна или вкладки. Кроме того, если вы перезагрузите страницу, браузер Safari не удалит предыдущие ошибки, поэтому в процессе работы у вас может накопиться длинный список ошибок. Для его очистки щелкните по значку в виде мусорной корзины, а затем обновите страницу

Глава 2
ГРАММАТИКА ЯЗЫКА JAVASCRIPT

Как уже говорилось ранее, изучение языка программирования во многом похоже на изучение иностранного языка: необходимо запоминать новые слова, разбираться с пунктуацией и овладевать новыми правилами. И точно так же, как вы должны изучить грамматику французского языка, чтобы говорить по-французски, вам нужно выучить грамматику языка JavaScript, чтобы программировать с его помощью. Эта глава посвящена концепциям, на которых основаны все программы JavaScript.

Если вы имеете опыт программирования на языке JavaScript, эти концепции вам давно известны, так что можете просто пропустить данную главу. Но если вы новичок в JavaScript, то эта глава разъяснит вам элементарные (но крайне важные) темы.

Инструкции

Инструкция — это базовая единица программирования, обычно обозначающая одно действие в программе JavaScript. Представьте, что инструкция — это предложение. Точно так же, как вы объединяете предложения, чтобы получился абзац (глава или книга), вы комбинируете инструкции для создания программы на языке JavaScript. В предыдущей главе вы уже сталкивались с инструкциями. Например:

```
alert('Привет, Мир!');
```

Эта инструкция открывает окно оповещения, содержащее фразу «Привет, Мир!» Во многих случаях инструкция — это одна строка кода. Каждая инструкция заканчивается точкой с запятой — это как точка в конце предложения. Точка с запятой означает, что инструкция закончена, и интерпретатор JavaScript должен перейти к следующему действию.

Общий алгоритм написания программы на языке JavaScript таков: напечатать инструкцию, поставить точку с запятой, нажать клавишу

Enter, чтобы создать пустую строку, напечатать в ней следующую инструкцию, закончить ее точкой с запятой и т. д. до тех пор, пока программа не будет завершена.

ПРИМЕЧАНИЕ

В более сложных примерах далее в книге вы увидите, что точка с запятой отсутствует в конце некоторых строк. Иногда точка с запятой появляется после многочисленных строк кода. Тем не менее эти строки следует рассматривать в качестве одной длинной инструкции с множеством различных знаков пунктуации (вроде данного предложения).

Официально точка с запятой не считается обязательным знаком в конце инструкции, и некоторые программисты опускают его, чтобы сократить код. Помните, что так делать не следует. Отсутствие точки с запятой затрудняет чтение вашего кода, а иногда может быть причиной ошибок в JavaScript. Если вы хотите сделать свой код более компактным, чтобы он быстрее загружался, обратитесь к разделу «Оптимизация сценариев JavaScript» главы 16.

Встроенные функции

Язык JavaScript и браузеры позволяют вам использовать различные команды для выполнения каких-либо действий в своих программах и на веб-страницах. Такие команды обычно называются *функциями* и похожи на глаголы в предложении. Они выполняют различные действия. Например, функция `alert()`, которую вы уже видели, заставляет браузер открыть диалоговое окно и отобразить сообщение.

Некоторые функции, например, `alert()` или `document.write()` (которые вы встретили в предыдущей главе), работают только на веб-страницах. Вы не найдете их при программировании в других средах, использующих язык JavaScript (например, в Node.js, Adobe Photoshop или ActionScript, основанном на JavaScript).

Другие функции универсальны и работают везде, где применяется этот язык. Например, команда `isNaN()`, проверяющая, является ли конкретное значение числовым, используется, если вы хотите узнать, ввел ли посетитель число, когда вопрос требует числового ответа (до-

пустим, «Сколько виджетов вам нужно?»). Подробнее о функции `isNaN()` и ее использовании вы узнаете в разделе «Работа с числами» главы 16.

Язык JavaScript имеет множество различных функций, о которых вы узнаете в процессе изучения данной книги. Команду в программе легко узнать по наличию скобок. Например, вы можете сказать наверняка, что `isNaN()` — это команда, поскольку за частью `isNaN` следуют круглые скобки.

JavaScript также позволяет вам создавать ваши собственные функции, поэтому диапазон операций, которые могут выполнять ваши сценарии, шире, чем стандартный набор команд языка JavaScript. Подробнее познакомиться с функциями вы сможете в главе 3.

ПРИМЕЧАНИЕ

Иногда вместо слова «функция» в JavaScript используется слово «метод».

Типы данных

Каждый день вы сталкиваетесь с различными видами информации. Ваше имя, стоимость обеда, адрес врача и дата вашего рождения — все это важные для вас сведения. Вы решаете, что делать и как жить, основываясь на имеющейся у вас информации. Компьютерные программы поступают точно также. Они тоже полагаются на имеющуюся информацию, чтобы выполнять задачи. Например, для вычисления стоимости заказа нужно знать цену и количество каждого заказанного товара. Чтобы добавить на веб-страницу имя посетителя («Рады снова тебя видеть, Катя!»), необходимо знать его имя.

Языки программирования обычно подразделяют информацию на различные типы и обращаются с каждым типом по-разному. В языке JavaScript выделяются три основных типа данных: `number` (число), `string` (строка) и `Boolean` (логический или булев тип).

Числа

Числа используются для подсчета и вычислений. Вы можете подсчитать количество дней до летнего отпуска или вычислить стоимость

двух билетов в кино. Числа играют важную роль в процессе программирования на языке JavaScript: подсчет используется для того, чтобы выяснить, сколько раз посетитель заходил на веб-страницу, а также узнать положение элемента на веб-странице с точностью до пиксела, или определить, сколько единиц товара хочет заказать посетитель.

В языке JavaScript число представляется с помощью цифры, например, *5*. Вы также можете использовать дробные числа: 5.25 или 10.3333333. Более того, JavaScript позволяет вводить отрицательные числа, например, −130 или −459.67.

Поскольку числа используются при вычислениях, ваши программы часто будут содержать математические *операции*. О них вы узнаете далее в разделе «Работа с типами данных и переменными», здесь же рассмотрим такой пример: допустим, что вам нужно записать на веб-странице результат сложения (5 + 15). Вы можете сделать это с помощью следующей строки кода:

```
document.write(5 + 15);
```

Этот фрагмент JavaScript-кода складывает два числа и отражает результат (20) на веб-странице. Существует много различных способов работы с числами, более подробно вы узнаете о них далее в этой главе.

Строки

Чтобы отобразить имя, предложение или любую последовательность букв, вы используете строки. *Строка* — это набор букв и других символов, заключенный в кавычки. Например, `'Привет, Галя!'` и `"Вы здесь"` — это строки. Вы уже использовали строку в предыдущей главе в команде оповещения: `alert('Привет, Мир!');`.

Открывающая кавычка сообщает интерпретатору JavaScript, что далее следует строка — серия символов. Интерпретатор воспринимает эти символы буквально, а не как нечто характерное для языка JavaScript, например, команду. Когда интерпретатор встречает закрывающую кавычку, он понимает, что достиг конца строки, и переходит к следующей части программы.

Для обозначения строк можно использовать как двойные (`"Привет, Мир"`), так и одинарные кавычки (`'Привет, Мир'`). Но следует

помнить, что в начале и в конце строки должен быть использован *один и тот же* тип кавычек (например, строка `"Это неправильно'` не является правильной, поскольку она начинается с двойной кавычки, а заканчивается одинарной).

ПРИМЕЧАНИЕ

При написании кода JavaScript, вы должны использовать обычные прямые кавычки: `""` и `''`.

Итак, чтобы открылось окно с оповещением «Внимание, внимание!», вы можете использовать следующий код:

```
alert('Внимание, внимание!');
```

или

```
alert("Внимание, внимание!");
```

В процессе программирования вы часто будете использовать строки, создавая оповещения, работая с данными, введенными пользователем в веб-форму, а также манипулируя содержимым веб-страницы. Вы более подробно узнаете о строках далее в разделе «Объединение строк».

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Кавычки в строках

Когда я пытаюсь создать строку с кавычками, программа не работает. Почему?

В языке JavaScript кавычки всегда означают начало и конец строки. Когда интерпретатор JavaScript встречает первую кавычку, он говорит себе: «Итак, здесь начинается строка». Когда он достигает соответствующей закрывающей кавычки, он понимает, что дошел до конца строки. Вот почему нельзя создать такую строку: `"Он сказал, "Привет""`. В данном случае первая кавычка (перед словом «Он») означает начало строки; как только интерпретатор JavaScript встречает вторую кавычку перед словом «Привет», он считает, что строка закончена, поэтому у вас получается строка `"Он сказал,"` и возникает ошибка JavaScript.

Есть несколько способов выйти из подобного положения. Самый простой — использовать одинарные кавычки для обозначения строки, внутри которой есть двойные кавычки. Например, `'Он сказал, "Привет"'` — это правильная строка. Одинарные кавычки обозначают строку, а двойные кавычки являются ее *частью*. Аналогично вы можете заключить в двойные кавычки строку, имеющую внутри одинарные кавычки, например, `"Д'Артаньян и три мушкетера"`.

Другой способ — это заставить интерпретатор JavaScript воспринимать кавычки в строке как ее часть, а не как ее окончание. Для этого необходимо использовать так называемый *знак переключения кода*. Если перед кавычкой поставить обратную косую черту (\), то кавычка будет считаться частью строки. Вы можете переписать приведенный выше пример так: `"Он сказал, \"Привет\""`. В некоторых случаях использование знака переключения кода — это единственно возможный выход. Например, `'Фильм называется "Д'Артаньян и три мушкетера"'`. Поскольку строка заключена в одинарные кавычки, в слове `Д'Артаньян` перед одинарной кавычкой необходимо поставить обратную косую черту: `Д\'Артаньян`.

Вы можете использовать знак переключения кода даже в том случае, когда это необязательно — просто чтобы одинарная кавычка воспринималась буквально. Например: `'Он сказал, \"Привет\"'`. Хотя здесь не обязательно использовать знак \ перед двойными кавычками (поскольку строку окружают одинарные кавычки), некоторые программисты делают так, чтобы показать, что кавычка — это просто кавычка.

Логический тип данных

Числа и строки могут принимать бесконечное множество значений, в то время как логическим (булевым) значением может быть либо *истина (true)*, либо *ложь (false)*. Вы встретитесь с этим типом данных при создании JavaScript-программ, реагирующих на действия пользователя и введенную им информацию. Например, вы хотите убедиться, что пользователь указал верный адрес электронной почты перед отправкой формы. Для этого добавьте на вашу страницу элемент логики, задав простой вопрос: «Ввел ли пользователь действительный почтовый адрес?». Ответ на этот вопрос является логическим значением: адрес электронной почты или действителен (истина), или нет (ложь). В зависимости от ответа код отреагирует по-разному. Например, если адрес электронной почты действителен (истина), то форма окажется отправлена, а если нет (ложь), то появится сообщение об ошибке, и форма отправлена не будет.

Логические типы данных настолько важны, что в языке JavaScript существуют два специальных ключевых слова для этих значений:

```
true
```

и

```
false
```

О логических типах данных вы узнаете подробнее в следующей главе.

Переменные

Вы можете внести числовое, строковое или логическое значение прямо в программу JavaScript, однако эти типы данных работают только в том случае, если в программе уже есть необходимая информация. Например, вы можете вызвать окно оповещения, содержащее строку `"Привет, Иван"` с помощью следующего фрагмента кода:

```
alert("Привет, Иван");
```

Однако эта фраза имеет смысл только в том случае, если всех посетителей данной страницы зовут Иван. Если вы хотите отобразить персональные сообщения для разных пользователей, имена должны быть разными — в зависимости от того, кто просматривает страницу: `'Привет, Мария'`, `'Привет, Петр'`, `'Привет, Ира'` и т. д. К счастью, во всех языках программирования предусмотрены так называемые *переменные*, предназначенные для разрешения ситуаций такого рода.

Переменная — это способ хранения информации для будущего использования. Например, рассмотрим созданную на языке JavaScript игру «Линии», цель игрока в которой — набрать как можно больше очков. Когда человек начинает играть, количество очков равно нулю, но когда игрок собирает линии из пяти шариков одного цвета, количество очков увеличивается. В данном случае *очки* являются переменной, изменяющейся в процессе игры. Другими словами, в переменной хранится информация, которая может *изменяться* (рис. 2.1).

Представьте, что переменная — это корзина: вы можете положить в нее что-нибудь, посмотреть, что в ней лежит, выбросить или заменить ее содержимое. Тем не менее при манипулировании содержимым сама корзина остается прежней.

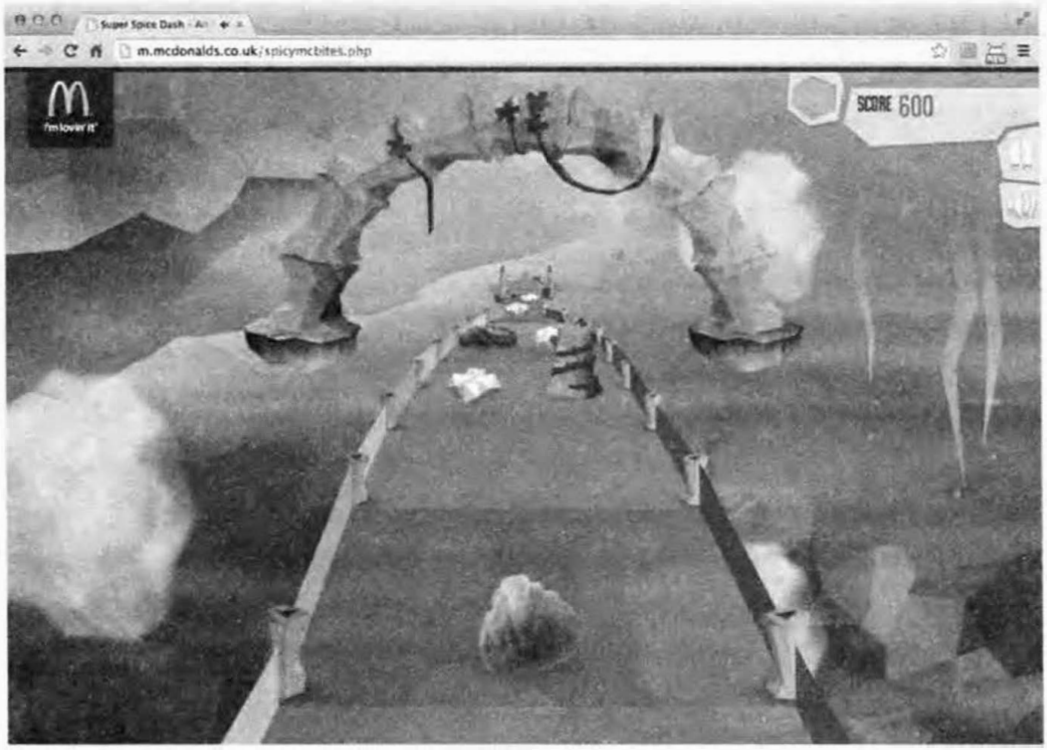

Рис. 2.1. На сайте **mcbites.sh75.net** код JavaScript используется для создания игры, способствующей продвижению продукции фирмы McDonalds. В этой игре вы управляете кусочком курицы (это не шутка), собирая золотые крошки, обходя препятствия и перепрыгивая через ямы. В процессе игры вы накапливаете очки, количество которых отображается в правом верхнем углу. Для ведения счета используется переменная, значение которой изменяется по мере вашего продвижения

Создание переменной

Создание переменной происходит в два этапа: *объявление* переменной и присвоение ей имени. В языке JavaScript для создания переменной с именем, к примеру, `score` нужно ввести:

```
var score;
```

Первая часть `var` — это ключевое слово языка JavaScript, которое создает, или, как говорят программисты, *декларирует* переменную. Вторая часть инструкции `score` — это имя переменной.

Как именовать переменные — решать вам, однако есть некоторые правила, которым необходимо следовать:

- **Имя переменной должно начинаться с буквы, символа $ или _.** Другими словами, имя переменной не может начинаться с цифры или знака препинания: имена `1thing` и `&thing` не будут работать, а имена `score`, `$score` и `_score` — будут.

- **Имена переменных могут содержать только буквы, цифры, символы $ и _.** Вы не можете использовать пробелы или другие специальные знаки в имени переменной: `fish&chips` и `fish and chips` — недопустимые имена, а `fish_and_chips` и `plan9` — допустимые. Не рекомендуется использовать символы, отличные от латиницы — переменные, написанные кириллицей, будут работать, но неустойчиво и могут быть непонятны вашим коллегам из других стран.
- **Имена переменных чувствительны к регистру.** Интерпретатор JavaScript по-разному воспринимает строчные и прописные буквы, то есть переменная `SCORE` отличается от переменной `score`, а также от переменных `sCoRE` и `Score`.

Табл. 2.1. Некоторые слова зарезервированы для использования языком JavaScript и браузером. Не следует называть этими словами переменные

Ключевые слова языка JavaScript	Слова, зарезервированные для использования в будущем	Слова, зарезервированные для браузера
break	abstract	alert
case	boolean	blur
catch	byte	closed
continue	char	document
debugger	class	focus
default	const	frames
delete	double	history
do	enum	innerHeight
else	export	innerWidth
false	extends	length
finally	final	location
for	float	navigator
function	goto	open
if	implements	outerHeight
in	import	outerWidth
instanceof	int	parent
new	interface	screen
null	let	screenX
return	long	screenY
switch	native	statusbar
this	package	window
throw	private	
true	protected	
try	public	
typeof	short	
var	static	
void	super	
while	synchronized	
with	throws	
	transient	
	volatile	
	yield	

- **Избегайте использования ключевых слов.** Некоторые слова в языке JavaScript зарезервированы, например, слово `var` используется для создания переменной, поэтому вы не можете назвать переменную `var`. Кроме того, некоторые слова, такие как `alert`, `document`, `window`, считаются особыми свойствами веб-браузера. Если вы попытаетесь использовать эти слова в качестве переменных, то получите сообщение об ошибке. Список зарезервированных слов представлен в табл. 2.1. Не все они инициируют ошибку в браузерах, но лучше воздержаться от их использования при наименовании переменных.

В дополнение к этим правилам вам следует присваивать своим переменным ясные и выразительные имена. Наименование переменных в соответствии с типом данных, которые вы в них храните, сильно облегчает чтение кода, сразу становится понятно, о чем идет речь. Например, `score` — это отличное имя переменной, в которой подсчитываются очки игрока. Конечно, вы можете присвоить переменной имя «`s`», но в этом случае будет непонятно, что за данные в ней хранятся.

Старайтесь, чтобы имена ваших переменных были удобны для чтения. Если вы используете в имени переменной более одного слова, то либо добавьте между словами символ нижнего подчеркивания, либо начинайте каждое слово после первого с заглавной буквы. Например, прочитать и понять имя `imagepath` сложнее, чем `image_path` или `imagePath`.

Использование переменных

Когда переменная создана, вы можете хранить в ней данные любого типа по своему усмотрению. Для того чтобы это сделать, нужно использовать символ =. Например, чтобы сохранить значение 0 в переменной `score`, введите такой код:

```
var score;
score = 0;
```

В первой строке кода вы создаете переменную, во второй — сохраняете в ней значение 0. Знак равенства называется *операцией присваивания*, поскольку он используется, чтобы присвоить переменной какое-либо значение. Вы также можете создать переменную и сохранить в ней значение с помощью одной инструкции JavaScript, например:

```
var score = 0;
```

В переменной вы можете сохранять числа, строки и логические значения:

```
var firstName = 'Питер';
var lastName = 'Паркер';
var age = 22;
var isSuperHero = true;
```

СОВЕТ

Чтобы сэкономить время, декларируйте несколько переменных одним ключевым словом `var`, например, так:

```
var x, y, z;
```

С помощью одной инструкции JavaScript вы можете объявить и сохранять значения в нескольких переменных:

```
var isSuperHero=true, isAfraidOfHeights=false;
```

После того как вы сохранили значение в переменной, вы можете получить доступ к этому значению, просто используя имя данной переменной.

Например, чтобы открыть диалоговое окно оповещения и отобразить значение, сохраненное в переменной `score`, введите код:

```
alert(score);
```

Обратите внимание, что в переменных не используются кавычки — они только для строк, поэтому в коде `alert('score')` будет отображаться слово `score`, а не значение, сохраненное в переменной `score`.

Теперь вы понимаете, почему строки следует заключать в кавычки: интерпретатор JavaScript воспринимает слова без кавычек либо как специальные объекты (например, команда `alert()`), либо как имена переменных.

ПРИМЕЧАНИЕ

Ключевое слово `var` можно использовать только один раз: когда вы создаете переменную. Далее вы можете присваивать ей новые значения, не используя слово `var`.

Работа с типами данных и переменными

Сохранение фрагмента информации, например, числа или последовательности символов в переменной, — это только первый шаг в создании программы. Большинство сценариев для получения новых результатов выполняет различные операции с данными. Например, добавление определенного количества очков для увеличения счета в процессе игры; умножение количества заказанных товаров на их стоимость; персонализация исходного сообщения путем добавления имени пользователя: «Мы рады снова видеть тебя, Игорь». Язык JavaScript предоставляет различные операции для работы с данными. Операция — это слово или символ, который может изменить одно или несколько значений. Например, используя символ + (операцию сложения), вы складываете величины. Для различных типов данных существуют различные типы операций.

Основные математические операции

Язык JavaScript поддерживает основные математические операции: сложение, деление, вычитание и т. д. В табл. 2.2 показаны основные математические операции и примеры их использования.

Табл. 2.2. Основные математические операции в языке JavaScript

Операция	Действие	Пример использования
+	Сложение двух чисел	`5 + 25`
–	Вычитание одного числа из другого	`25 – 5`
*	Перемножение двух чисел	`5 * 10`
/	Деление одного числа на другое	`15/5`

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Пробелы, табуляция и возврат каретки в языке JavaScript

Язык JavaScript, похоже, очень чувствителен к опечаткам. Как узнать, где от меня ожидается использование пробелов, а где их использовать нельзя?

Вообще, язык JavaScript довольно непритязателен в отношении пробелов, табуляции и возвратов каретки. Вы часто без проблем може-

те пропускать или добавлять лишние пробелы и возвраты каретки (переводы строки). Интерпретаторы JavaScript игнорируют лишние пробелы, поэтому вы можете свободно добавлять пробелы, табуляцию и возвраты каретки для форматирования кода. Например, вам не обязательно ставить пробелы с обеих сторон от операции присваивания, но вы можете это сделать, если думаете, что так будет легче читать код. Обе следующие строки кода будут работать:

```
var formName ='подписка';
var formRegistration = 'рассылка';
```

В рамках инструкции вы можете добавить любое количество пробелов и возвратов каретки. Так что обе следующие строки кода тоже будут работать:

```
var formName = 'подписка';
var formRegistration
=
 'рассылка';
```

Конечно же, возможность поставить лишний пробел не означает, что делать это необходимо. Последние два примера как раз труднее читать и понимать именно из-за лишних пробелов. Итак, главное эмпирическое правило — добавлять лишние пробелы только в том случае, если они облегчают чтение кода. Например, дополнительный возврат каретки упрощает чтение кода при одновременном объявлении нескольких переменных и присвоения им значений. Следующий код пишется в одну строку:

```
var score=0, highScore=0, player='';
```

Однако некоторые программисты предпочитают располагать каждую переменную в отдельной строке для облегчения чтения кода:

```
var score=0,
highScore=0,
player='';
```

Если вы считаете, что подобное расположение облегчает чтение кода, используйте его; интерпретатор JavaScript просто проигнорирует прерывания строк. Вы увидите, как пробелы облегчают восприятие кода в примерах с массивами и объектными литералами языка JavaScript.

Существуют два важных исключения из правил, приведенных выше. Например, нельзя возвращать каретку внутри строки. Другими словами, строку нельзя разбить на две строки кода, например, так:

```
var name = 'Иван
Иванов';
```

Подобное использование возврата каретки (путем нажатия клавиши **Enter**) инициирует ошибку в JavaScript-коде, и ваша программа работать не будет.

В дополнение к этому, вам следует добавлять пробел между ключевыми словами: например, `varscore=0` — это не то же самое, что `var score=0`. Последний пример создает новую переменную с именем `score`, в то время как первый пример сохраняет значение 0 в переменной с именем `varscore`. Интерпретатор JavaScript нуждается в пробеле между словами `var` и `score` для того, чтобы определить ключевое слово `var`: `var score=0`. В то же время пробел необязателен между ключевыми словами и такими символами, как операция присваивания (=) или точка с запятой, которая обозначает окончание инструкции.

Возможно, вы привыкли обозначать действие умножения символом × (например, 4 × 5), однако в языке JavaScript для перемножения двух чисел используется символ *.

В математических операциях вы также можете использовать переменные. Так как переменная — это всего лишь «хранилище» для какого-то значения, например числа или последовательности символов, то использование переменной тождественно использованию ее содержимого:

```
var price = 10;

var itemsOrdered = 15;

var totalCost = price * itemsOrdered;
```

В первых двух строках кода создаются две переменные (`price` и `itemsOrdered`), в каждой из которых хранится число. Третья строка кода создает следующую переменную (`totalCost`) и сохраняет в ней результат перемножения значений переменной `price` (10) и переменной `itemsOrdered` (15). В данном случае итог (150) сохраняется в переменной `totalCost`.

В этом примере демонстрируется удобство использования переменных. Допустим, вы пишете программу, являющуюся частью системы заказа товаров в электронном магазине. На протяжении всей программы вы должны использовать цену отдельно взятого товара для различных расчетов. Вы можете кодировать точную цену на протяжении всей программы (скажем, товар стоит 100 рублей, то есть вы вводите в программу значение 100 везде, где используется цена). Однако если цена когда-нибудь из-

менится, вам придется находить и изменять это значение в каждой строке кода, в которой присутствует цена. При использовании переменной вы зададите цену в начале программы. Потом, если цена когда-нибудь изменится, вы отредактируете всего одну строку кода, определяющую стоимость товара, чтобы обновить цену сразу во всей программе:

```
var price = 20;
var itemsOrdered = 15;
var totalCost = price * itemsOrdered;
```

Существует много других способов работы с числами (см. раздел «Работа с числами» главы 16), но наиболее часто вы будете пользоваться основными математическими операциями, перечисленными в табл. 2.2.

Порядок операций

Осуществляя несколько математических операций одновременно, например, складывая несколько чисел и умножая сумму на 10, вы должны помнить, в каком порядке интерпретатор JavaScript производит вычисления. Одни операции имеют приоритет над другими и производятся в первую очередь. Поэтому из-за невнимательности вы можете получить ошибочный результат. Например:

```
4 + 5 * 10
```

Вам может показаться, что эти операции производятся слева направо: 4 + 5 = 9, а 9 * 10 = 90. Но это не так. Умножение выполняется раньше сложения, поэтому результат этого равенства: 5 * 10 = 50 + 4 = 54. Операции умножение (*) и деление (/) имеют приоритет над действиями сложение (+) и вычитание (–).

Чтобы удостовериться в правильности результата, используйте круглые скобки для группирования операций. Приведенный выше пример может быть переписан таким образом:

```
(4 + 5) * 10
```

Все действия в скобках производятся в первую очередь, поэтому в данном случае, сначала выполняется сложение: 4 + 5, результат которого (9) умножается на 10. Если вы хотите сначала произвести умножение, то код будет яснее, если записать его так:

```
4 + (5*10);
```

Объединение строк

Объединение двух и более строк в одну — это обычная задача при программировании. Например, если на веб-странице есть форма, предусматривающая введение пользователем своего имени и фамилии в разные поля, то вам, чтобы получить его полное имя, следует скомбинировать данные в этих полях. Более того, если вы пожелаете отобразить сообщение, дающее пользователю знать, что заполненная им форма принята, вы должны объединить сообщение с именем этого посетителя: «Иван Петров, спасибо за ваш заказ».

Объединение строк называется *конкатенацией* и осуществляется с помощью операции +. Да, это та самая операция, которая используется для сложения числовых величин, однако со строками он ведет себя несколько иначе. Вот пример:

```
var firstName = 'Иван';
var lastName = 'Петров';
var fullName = firstName + lastName;
```

В последней строке кода содержимое переменной `firstName` комбинируется с содержимым переменной `lastName`. Обе переменные буквально соединяются, и результат помещается в переменную `fullName`. В результате получается строка «ИванПетров». Между именем и фамилией нет пробела, поскольку при объединении строки просто «склеиваются» друг с другом. Во многих случаях (например, в таком, как этот) вы должны добавить между объединяемыми строками пробел:

```
var firstName = 'Иван';
var lastName = 'Петров';
var fullName = firstName + ' ' + lastName;
```

Одинарные кавычки и пробел между ними `' '` — это просто строка, содержащая пробел. Помещенный между двумя переменными, этот код создает строку «Иван Петров». Данный пример также показывает, как вы можете объединить более двух строк одновременно, в данном случае — три строки.

ПРИМЕЧАНИЕ

Помните, что переменная — это просто контейнер, который может содержать любой тип данных, например, строку или чис-

ло. Поэтому объединение двух переменных, содержащих строки (`firstName + lastName`), равнозначно объединению этих двух строк: `'Иван'+'Петров'`.

Объединение чисел и строк

Большая часть математических операций применяется только к числам. Например, нет смысла умножать два на «яблоки». Если вы сделаете это, то получите особое значение языка JavaScript — *NaN*, которое означает «не число». Однако может случиться, что вам потребуется объединить строку и число. Например, вам нужно вывести сообщение, которое указывает, сколько раз посетитель был на вашей странице. Количество его визитов — это *число*, а сообщение — *строка*. В данном случае, используя операцию +, вы совершаете два действия: преобразуете число в строку и соединяете его с другой строкой:

```
var numOfVisits = 101;
var message = 'Вы посетили этот сайт ' ↵
+ numOfVisits +  ' раз.';
```

В данном случае переменная `message` содержит строку «Вы посетили этот сайт 101 раз». Интерпретатор JavaScript понимает, что здесь присутствует строка, поэтому не следует совершать математической операции (сложения). Вместо этого он воспринимает знак + как операцию объединения строк, а также понимает, что число должно быть преобразовано в строку. Этот способ может показаться подходящим для совмещения чисел и строк в одном сообщении. В данном случае, очевидно, что число является частью символьной последовательности, образующей законченное предложение, и каждый раз, когда вы используете операцию + по отношению к числу и строке, интерпретатор JavaScript преобразует число в строку.

Однако данная функция, известная как *автоматическое преобразование типов данных*, может вызывать проблемы. Например, если в специальной форме посетитель отвечает на вопрос («Сколько пар обуви вы хотите?»), набирая число (например, 2), то это число считается строкой — `'2'`. И вы можете получить следующее:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = numOfShoes + numOfSocks;
```

Вы ожидаете, что число, сохраненное в переменной `totalItems`, — 6 (2 пары обуви + 4 пары носков). Однако, поскольку значение переменной `numOfShoes` является строкой, интерпретатор JavaScript преобразует значение переменной `numOfSocks` опять же в строку, и в итоге вы получаете строку `'24'` в переменной `totalItems`. Есть пара способов для предотвращения подобной ошибки.

Во-первых, вы можете добавить знак + в начале *строки*, содержащей число:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = +numOfShoes + numOfSocks;
```

Добавляя знак + перед переменной (убедитесь, что между ними нет пробела), вы просите интерпретатор JavaScript попытаться преобразовать строку в числовое значение. Если строка содержит только числа, например, `'2'`, то в итоге вы получите строку, преобразованную в число. В данном примере вы получите 6 (2 + 4).

Во-вторых, можно использовать команду `Number()`:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = Number(numOfShoes) + numOfSocks;
```

Команда `Number()` преобразует строку в число, если это возможно. Если строка содержит только буквы, вы получите значение `NaN` в качестве сообщения о том, что преобразование букв в числа невозможно.

В большинстве случаев вам придется преобразовывать числа в строки при обработке пользовательского ввода, например, значения, введенного посетителем в поле формы. Поэтому, если вам необходимо совершить операцию сложения с использованием информации, введенной пользователем, не забудьте применить к полученным от пользователя данным команду `Number()`.

ПРИМЕЧАНИЕ

Подобная проблема появляется только тогда, когда вы пытаетесь объединить число со строкой, содержащей число. Если вы попробуете умножить переменную `numOfShoes` на переменную, содержащую число, например, `shoePrice`, то интерпретатор JavaScript пре-

образует строку `numOfShoes` в число, а затем умножит это число на значение переменной `shoePrice`.

Изменение значений в переменных

Переменные полезны, поскольку могут содержать значения, изменяющиеся в процессе работы программы, например, очки, которых в ходе игры может стать больше или меньше. Как же изменить значение переменной? Если вы просто хотите заменить содержимое переменной, присвойте этой переменной новое значение, например:

```
var score = 0;
score = 100;
```

Однако зачастую нужно сохранить прежнее значение переменной, но, скажем, добавить к нему какое-то число. Продолжим пример с очками в игре. Вы никогда не устанавливаете новый счет, а просто прибавляете или вычитаете из существующего количества очков какое-либо число. Чтобы увеличить значение переменной, используйте его имя в качестве слагаемого:

```
var score = 0;
score = score + 100;
```

На первый взгляд последняя строка кода может показаться непонятной, однако в ней используется совершенно обычная техника. Вот как она работает: все, что находится правее знака =, вычисляется в первую очередь. Это часть `score + 100`. Программа понимает данную команду следующим образом: «Возьми то, что сейчас содержится в переменной `score` (0), и прибавь к этому 100». Затем результат операции заново сохраняется в переменной `score`. Окончательный итог выполнения двух строк кода — переменная `score` теперь имеет значение 100.

Та же логика применима и к другим математическим операциям, таким как вычитание, деление и умножение:

```
score = score - 10;
score = score * 10;
score = score / 10;
```

На самом деле математическая обработка значения переменной и последующее сохранение результата в той же переменной — достаточ-

но обычные действия, поэтому для их осуществления есть специальные комбинированные операции (табл. 2.3).

Табл. 2.3. Комбинированные операции для математических действий над переменными

Операция	Действие	Как использовать	Альтернатива
+=	Складывает значение, находящееся справа от знака =, с переменной, находящейся слева от знака +	`score += 10`	`score = score + 10`
–=	Вычитает значение, находящееся справа от знака =, из переменной, находящейся слева от знака –	`score –= 10`	`score = score – 10`
*=	Умножает переменную, находящуюся слева от знака *, на значение, находящееся справа от знака =	`score *= 10`	`score = score * 10`
/=	Делит переменную, находящуюся слева от знака / на значение, находящееся справа от знака =	`score /= 10`	`score = score / 10`
++	Помещенный непосредственно после имени переменой, данный знак прибавляет к переменной 1	`score ++`	`score = score + 1`
––	Помещенный непосредственно после имени переменой, данный знак вычитает из переменной 1	`score – –`	`score = score – 1`

Те же правила работают и при объединении строки и переменной. Например, у вас есть переменная со строкой, и вы хотите добавить к этой переменной еще пару строк.

```
var name = 'Иван';
var message = 'Привет';
message = message + ' ' + name;
```

Как и в случае с числами, существует сокращенная операция для соединения строки и переменной. Операция += добавляет то, что располагается справа от знака =, в конец строки переменной. Таким образом, последнюю строку данного выше кода можно переписать так:

```
message += ' ' + name;
```

Вы будете часто встречать операцию += при работе со строками.

Использование переменных для создания сообщений на практике

В данном руководстве вы будете использовать переменные, чтобы отобразить сообщение на веб-странице.

ПРИМЕЧАНИЕ

Чтобы выполнить инструкции этой главы, вам следует использовать прилагаемые файлы примеров.

1. **В текстовом редакторе откройте файл *02_01.html* из каталога *глава02*.**

Это обычный HTML-файл с простым дизайном, усовершенствованным с помощью CSS. В нем еще нет кода JavaScript. Вы будете использовать переменные для создания сообщения на веб-странице.

2. **Найдите элемент `h1` (находится примерно в центре кода) и добавьте открывающий и закрывающий теги элемента `script`, чтобы код выглядел так:**

```
<h1>Использование переменной</h1>
<script>
</script>
```

Этот HTML-код уже должен быть знаком вам: он просто подготавливает место для сценария, который вы собираетесь добавить.

ПРИМЕЧАНИЕ

В данном примере используется HTML5. Если вы работаете с версиями XHTML 1.0 или HTML 4.01, добавьте код `type="javascript"` в тег `<script>` следующим образом:
`<script type="text/javascript">`.
Этот код не влияет на работу сценария; он нужен только для того, чтобы страница прошла проверку валидатором W3C.

3. **Между открывающим и закрывающим тегами элемента `script` введите:**

```
var firstName = 'Иван';
var lastName = 'Петров';
```

Вы только что создали две свои первые переменные (`firstName` и `lastName`) и сохранили в них два значения строкового типа. Теперь вам предстоит совместить две строки и отобразить результат на веб-странице.

4. Под объявлением переменных напечатайте:

```
document.write('<p>');
```

Как вы помните из главы 1, команда `document.write()` добавляет текст прямо на веб-страницу. В данном случае вы используете ее для записи HTML-тегов на вашей странице. Вы предоставляете команде строку — `'<p>'` и она выдает эту строку, как будто вы написали ее в HTML-коде. Использование HTML-тегов в качестве части команды `document.write()` совершенно обычное явление. В данном случае JavaScript добавляет открывающий тег абзаца, который будет содержать текст, отображаемый на странице.

ПРИМЕЧАНИЕ

Существуют более эффективные методы добавления HTML-кода на веб-страницу, чем команда `document.write()`. Вы узнаете о них в главе 4.

5. Нажмите клавишу Enter и напечатайте следующий код JavaScript:

```
document.write(firstName + ' ' + lastName);
```

Здесь вы используете значения, сохраненные в переменных на шаге 3. Операция + позволяет объединить несколько строк для создания одной длинной строки, которую команда `document.write()` затем записывает в HTML-код страницы. В данном случае к значению, сохраненному в переменной `firstName` — `'Иван'`, добавляется пробел, а затем — значение переменной `lastName` — `'Петров'`. В результате получается одна строка: `'Иван Петров'`.

6. Вновь нажмите клавишу Enter и наберите
`document.write('</p>');`

В итоге код сценария должен выглядеть так:

```
<script type="text/javascript">
var firstName = 'Иван';
var lastName = 'Петров';
```

```
document.write('<p>');
document.write(firstName + ' ' + lastName);
document.write('</p>');
</script>
```

7. **Просмотрите страницу в браузере, чтобы насладиться плодами своего труда (рис. 2.2).**

Имя и фамилия должны появиться под заголовком «Использование переменной». Если вы ничего не видите, то, возможно, допустили опечатку в коде.

Сравните сценарий из книги с тем, что создали вы, а также посмотрите советы по отладке сценария с использованием браузеров Firefox, Safari, Chrome или Internet Explorer в разделе «Отслеживание ошибок» главы 1.

8. **Вернитесь в текстовый редактор и измените вторую строку сценария, чтобы получилось следующее:**

```
var lastName = 'Сидоров';
```

Сохраните страницу и просмотрите ее в браузере. Сообщение изменилось на следующее: «Иван Сидоров». Файл *готовый_02_01.html* содержит рабочую копию данного сценария.

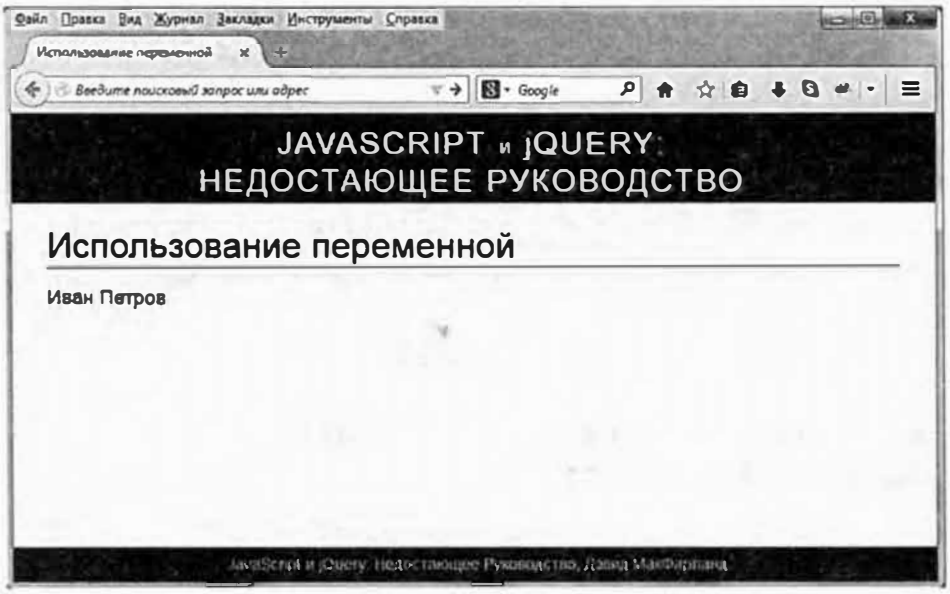

Рис. 2.2. Создание надписи «Иван Петров», возможно, не является вашей целью при чтении данной книги, но, тем не менее, этот сценарий продемонстрировал вам, как создавать и использовать переменные в JavaScript

Запрос информации на практике

В предыдущем руководстве мы разобрали, как создавать переменные, но не поговорили о том, как переменные могут реагировать на действия пользователя и создавать уникальный контент. В данном руководстве вы узнаете, как использовать команду `prompt()` для сбора введенных пользователем данных и изменения отображаемой на странице информации, на основе этого ввода.

1. **В текстовом редакторе откройте файл *02_02.html* каталога *глава02*.**

Чтобы ускорить процесс программирования, в этот файл уже добавлены элементы `script`. Вы увидите два набора тегов элемента `script`: один — в разделе заголовка, а другой — в теле документа. Код JavaScript, который вы собираетесь добавить, будет, во-первых, открывать диалоговое окно, которое попросит пользователя ответить на вопрос; во-вторых, создаст в теле веб-страницы уникальное сообщение, основываясь на ответе пользователя.

2. **Между первым набором тегов элемента `script`, в разделе заголовка документа, введите выделенный полужирным шрифтом код:**

```
<script>
var name = prompt('Как вас зовут?', '');
</script>
```

Функция `prompt()` создает диалоговое окно подобно функции `alert()`. Однако она не просто отображает сообщение, она запрашивает ответ (рис. 2.3). Для использования команды `prompt()` вы добавляете две строки в скобках, разделяя их запятой. На рис. 2.3 показано, что происходит с этими двумя строками: первая появляется в диалоговом окне (в данном примере — «Как вас зовут?»).

Вторая строка появляется в поле, в которое пользователь вводит информацию. В данном примере используется так называемая *пустая строка*, которая записывается с помощью двух одинарных кавычек (`''`) и создает в результате пустое текстовое поле. Однако вы можете снабдить ее полезной инструкцией, например, «Пожалуйста, введите ваши имя и фамилию», которая появится в текстовом поле. К сожалению, посетителю придется сначала удалить этот текст, а потом ввести в поле свои данные.

Рис. 2.3. Команда `prompt()` — это один из способов получать информацию от пользователя. Она предоставляет функции две строки: одна появляется в виде вопроса, а другая заполняет поле ввода текстом

Команда `prompt()` возвращает строку, в которой содержится информация, введенная пользователем в диалоговое окно. В данной строке кода JavaScript результат сохраняется в новой переменной `name`.

ПРИМЕЧАНИЕ

Многие команды *возвращают* значение. Проще говоря, выполнение команды предоставляет какую-либо информацию. Вы можете игнорировать ее или сохранить в переменной для последующего использования. В данном примере команда `prompt()` возвращает строку, которую вы сохраняете в переменной `name`.

3. Сохраните страницу и просмотрите ее в браузере.

Когда страница загрузится, вы увидите диалоговое окно. Обратите внимание, что больше ничего не происходит. Вы даже не увидите веб-страницы, пока не заполните диалоговое окно и не щелкнете по кнопке **OK**. После этого также ничего не произойдет, поскольку на данном этапе вы просто получили и сохранили ответ. Вы не использовали его на данной странице — вам только предстоит это сделать.

4. Вернитесь в текстовый редактор. Найдите второй набор тегов элемента `script` и добавьте код, выделенный полужирным шрифтом:

```
<script>
document.write('<p>Добро пожаловать, ' + name + '</p>');
</script>
```

Здесь вы используете информацию, введенную пользователем. Как и в предыдущем сценарии, вы комбинируете несколько строк: откры-

вающий тег абзаца, текст, значение переменной и закрывающий тег абзаца, а затем выводите результат на веб-страницу.

5. Сохраните страницу и просмотрите ее в браузере.

Когда появится диалоговое окно, введите имя и нажмите кнопку **OK**. Обратите внимание, что имя появилось на веб-странице (рис. 2.4). Обновите страницу и напечатайте новое имя — оно изменилось! Так и должна вести себя переменная.

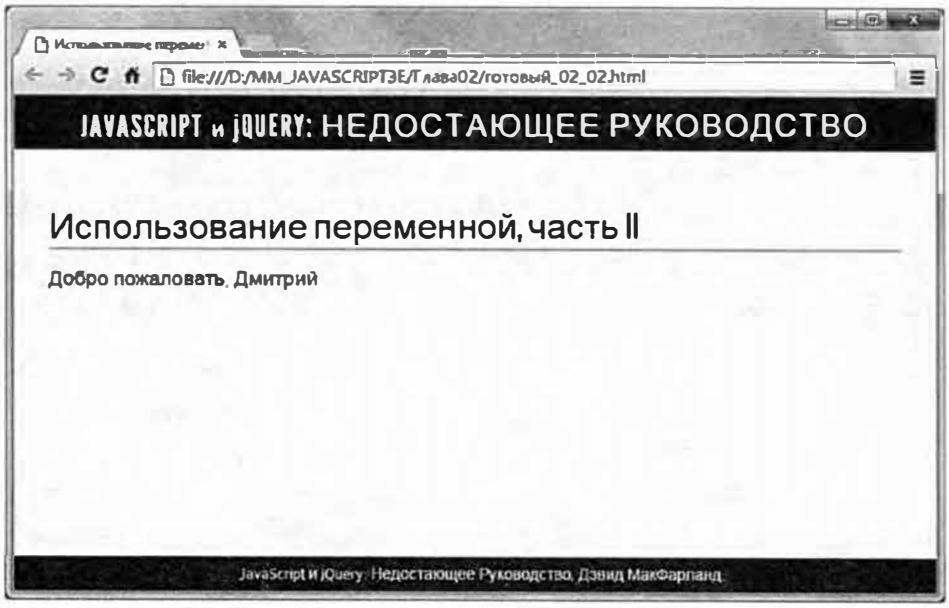

Рис. 2.4. Эта страница создает уникальное сообщение на основе ответа пользователя

Массивы

Простые переменные, как те, с которыми вы познакомились в предыдущем разделе, содержат только фрагмент информации, например, числовое значение или последовательность символов. Они отлично подходят для отслеживания одного показателя (счета, возраста или общей стоимости). Однако если вам приходится отслеживать несколько однородных показателей (дней недели или изображений на веб-странице), то обычные переменные не очень удобны.

Например, вы создали систему электронного заказа на языке JavaScript, отслеживающую товары, которые собирается приобрести посетитель. Если вы решите отслеживать все единицы товара, которые посетитель добавил в свою корзину, используя обычные переменные, то должны будете написать следующий код:

```
var item1 = 'Игровая приставка Xbox 360';
var item2 = 'Теннисные туфли';
var item3 = 'Подарочный сертификат';
```

Но что, если товаров будет больше? Вам придется добавить новые переменные: `var item4`, `var item5` и т. д. Поскольку вы не знаете, сколько товаров захочет заказать посетитель, вы не можете заранее угадать, сколько переменных потребуется создать.

К счастью, язык JavaScript предусматривает более удобный способ отслеживания списка элементов, называемый *массивом*. Массив — это способ хранения более одного значения в одном и том же месте. Представьте, что массив — это список покупок. Когда вам нужно сходить в магазин, вы садитесь и записываете то, что хотите купить. Если вы недавно ходили за покупками, в списке может быть всего несколько позиций. Однако если ваш холодильник пуст, то список может быть весьма длинным. Однако вне зависимости от количества позиций, список в любом случае один.

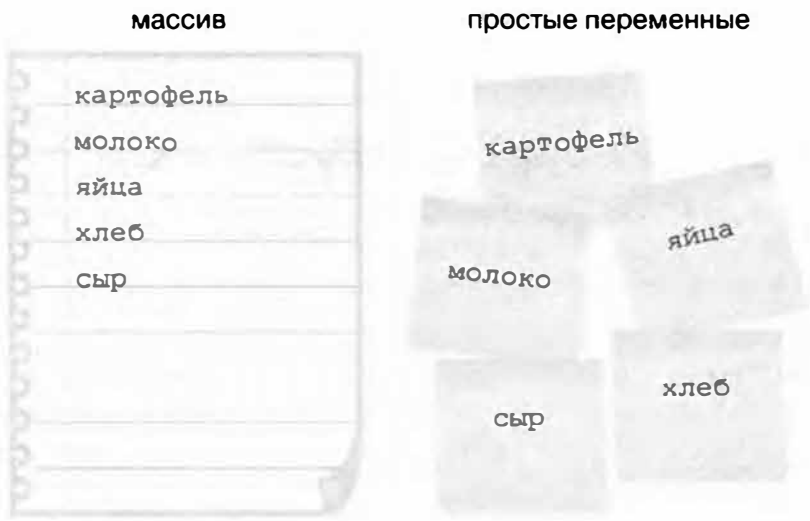

Рис. 2.5. Массив — это простой способ отслеживания списка взаимосвязанных позиций. Добавить в список новую позицию так же просто, как записать в конце обычного списка еще один пункт

Без использования массива вам придется создавать новую переменную для каждой позиции в списке. Представьте, например, что вы не могли бы составить список покупок на одном листе бумаги, а были бы вынуждены носить с собой множество клочков бумаги — по одному для каждого товара. Чтобы добавить новый товар, вам потребовался бы новый листок бумаги, далее, вам пришлось бы следить за тем, какие из товаров вы уже купили (рис. 2.5). Так работают простые переменные.

С помощью массива вы можете создать единый список позиций, и даже добавлять, удалять и изменять позиции в любое время.

Создание массива

Для создания массива и сохранения в нем каких-либо элементов сначала необходимо объявить его имя (как и в случае с переменной), а затем добавить к имени список значений, разделенных запятыми: каждое значение представляет собой один из элементов списка. Как и в случае с переменной вы можете выбрать любое имя, но вам следует придерживаться правил, перечисленных в разделе «Создание переменной». Чтобы создать массив, поместите список элементов в квадратные скобки — `[]`. Например, чтобы создать массив, содержащий аббревиатуры дней недели, следует написать такой код:

```
var days = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс'];
```

Квадратные скобки — `[]` — очень важны. Они сообщают интерпретатору JavaScript о том, что он имеет дело с массивом. Вы также можете создать пустой массив, не содержащий элементов:

```
var playList = [];
```

Создание пустого массива равнозначно декларированию переменной, описанному в разделе «Создание переменной». Вы создаете пустой массив, если не добавляете в него элементы до тех пор, пока программа не будет запущена. Например, пустой массив может быть использован для отслеживания названий песен, которые посетитель выберет из списка на веб-странице. Вы не можете знать заранее, какие песни будут выбраны, поэтому декларируете пустой массив, а позднее заполняете его элементами по мере того, как посетитель выбирает композиции (добавление элементов в массив описано далее в этой главе).

ПРИМЕЧАНИЕ

В программах на языке JavaScript и книгах по этой теме можно встретить и другой способ создания массива — с использованием ключевого слова `Array`:

```
var days = new Array('Пн', 'Вт', 'Ср');
```

Данный способ допустим, но метод, предлагаемый в данном пособии (называемый *литералом массива*), предпочитают многие профессионалы, потому что он требует меньше символов и считается более «простым».

Вы можете хранить в массиве любые значения. Другими словами, в одном и том же массиве могут присутствовать числа, строки и логические значения:

```
var prefs = [1, -30.4, 'www.eksmo.ru', false];
```

ПРИМЕЧАНИЕ

В качестве элементов массива вы можете использовать другие массивы и иные объекты. Этот способ позволяет хранить сложные данные.

Примеры массивов, рассмотренные выше, представляют собой одиночные строки кода. Однако если вам нужно добавить в массив много элементов или если элементы — это длинные последовательности символов, то, пытаясь набрать их одной строкой, вы усложните чтение программы. Многие программисты создают массив из нескольких строк кода:

```
var authors = [ 'Лев Толстой',
 'Николай Гоголь',
 'Иван Тургенев',
 'Антон Чехов'
 ];
```

Как уже говорилось, интерпретатор JavaScript пропускает дополнительные пробелы и разрывы строк, поэтому, несмотря на то, что данный фрагмент кода расположен на пяти строках, он является единой инструкцией, на что указывает точка с запятой в последней строке.

СОВЕТ

Чтобы оформить имена так, как показано выше, вы должны написать первую строку: `var authors = [ 'Лев Толстой'`, нажать клавишу **Enter**, а затем вставить столько пробелов, сколько необходимо, чтобы выровнять новое значение `'Николай Гоголь'` со значением из предыдущей строки.

Доступ к элементам массива

Вы можете получить доступ к значению переменной, просто используя ее имя. Например, команда `alert(lastName)` открывает окно оповещения со значением, сохраненным в переменной `lastName`. Однако, по-

скольку массив может содержать более одного значения, вы не получите доступ к его элементам, просто введя имя массива. Уникальный номер, называемый *индексом*, указывает позицию каждого элемента в массиве. Чтобы получить доступ к отдельно взятому элементу, вы должны использовать его индекс. Например, вы создали массив с сокращениями названий дней недели и хотите открыть окно оповещения, которое отобразит первый элемент этого массива. Вы можете сделать это следующим образом:

```
var days = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс'];
alert(days[0]);
```

Данный фрагмент кода открывает окно оповещения с элементом `'Пн'`. Массивы индексируются с нуля, это означает, что первый элемент имеет индекс 0, а второй — 1. Иными словами, вычтите единицу из номера элемента в списке — и вы получите его индексное значение. Индекс пятого элемента в списке — это его номер по порядку (5) минус 1, то есть **4**. Индексация с нуля может запутать, когда вы только начинаете программировать. В табл. 2.**4** показано, как индексируется массив `days` (из примера, приведенного выше), какие значения он содержит и как получить доступ к каждому из них.

Табл. 2.4. Получить доступ к элементам массива можно при помощи индекса, который равен порядковому номеру элемента в массиве минус 1

Значение индекса	Элемент	Доступ к элементу
0	Пн	days[0]
1	Вт	days[1]
2	Ср	days[2]
3	Чт	days[3]
4	Пт	days[4]
5	Сб	days[5]
6	Вс	days[6]

Вы можете изменить значение элемента массива, присвоив его позиции новое значение. Например, чтобы присвоить новое значение первому элементу массива `days`, укажите следующее:

```
days[0] = 'Понедельник';
```

Поскольку индексный номер последнего элемента массива всегда на единицу меньше их общего числа то, чтобы получить доступ к последнему элементу, вам достаточно знать, сколько всего элементов содержится в массиве. К счастью, это не сложная задача, поскольку среди

свойств массива есть его длина, показывающая общее число элементов. Чтобы получить доступ к свойству `length`, наберите после имени массива точку и слово `length`. Например, код `days.length` возвращает количество элементов в массиве `days` (если вы создали иной массив, например, `playList`, то узнаете число его элементов таким образом: `playList.length`). Теперь вы можете получить доступ к значению последнего элемента в массиве:

```
days[days.length-1]
```

Этот фрагмент кода показывает, что вам необязательно использовать конкретное число в качестве индекса (например, 0 в коде `days[0]`). Вы также можете использовать равенство, возвращающее необходимый номер: в данном случае часть `days.length-1` является равенством — оно сначала находит общее количество элементов массива `дни` (в нашем примере — 7) и вычитает из него единицу. Итак, инструкция `days[days.length-1]` преобразуется в инструкцию `days[6]`.

Вы также можете использовать в качестве индекса переменную, содержащую номер:

```
var i = 0;
alert(days[i]);
```

Последняя строка кода эквивалентна записи `alert(days[0]);`. Эта техника может быть полезна при работе с циклами, описанными в следующей главе.

Добавление элементов в массив

Допустим, вы создали массив для отслеживания объектов на вебстранице, по которым пользователь щелкает кнопкой мыши. При каждом щелчке в массив добавляется элемент. Язык JavaScript предусматривает несколько способов добавления содержимого в массив.

Добавление элемента в конец массива

Чтобы добавить элемент в конец массива, вы можете использовать значение индекса, которое на единицу больше, чем значение индекса последнего элемента массива. Допустим, вы создали массив с именем `properties`:

```
var properties = ['red', '14px', 'Arial'];
```

В данный момент этот массив содержит три элемента. Вы уже знаете, что индекс последнего элемента массива на единицу меньше общего числа элементов. Таким образом, последний элемент в данном массиве — `properties[2]`. Добавим еще один элемент в массив:

```
properties[3] = 'bold';
```

Данная строка кода помещает элемент `'bold'` на четвертую позицию, в результате получается массив из четырех элементов: `['red', '14px', 'Arial', 'bold']`. Обратите внимание, что, добавляя новый элемент, вы используете значение индекса, равное общему числу элементов, содержащихся в данный момент в массиве, поэтому, используя в качестве индекса свойство массива `length`, вы можете быть уверены, что всегда добавляете новый элемент в конец массива. Например, вы можете переписать код следующим образом:

```
properties[properties.length] = 'bold';
```

Вы также можете использовать команду `push()`, которая добавит в массив значение, указанное в круглых скобках. Рассмотрим этот способ добавления элемента в конец массива `properties`:

```
properties.push('bold');
```

Что бы вы ни указали в круглых скобках (в данном случае строка `'bold'`), это добавляется в качестве нового элемента в конец массива. Вы можете использовать любой тип значения, например, строковое, числовое, логическое значение и даже переменную.

Одно из преимуществ команды `push()` состоит в том, что она позволяет добавлять в массив более одного элемента. Например, если вы хотите добавить три значения в конец массива `properties`, используйте следующий код:

```
properties.push('bold', 'italic', 'underlined');
```

Добавление элемента в начало массива

Если вы хотите добавить элемент в начало массива, используйте команду `unshift()`. Вот как можно добавить значение `'bold'` в начало массива `properties`:

```
var properties = ['red', '14px', 'Arial'];
properties.unshift('bold');
```

В результате выполнения этого кода массив `properties` будет содержать четыре элемента: `['bold', 'red', '14px', 'Arial']`. Как и в случае с командой `push()`, вы можете использовать функцию `unshift()` для вставки нескольких элементов в начало массива:

```
properties.unshift('bold', 'italic', 'underlined');
```

ПРИМЕЧАНИЕ

Убедитесь, что вы используете имя массива, за которым следуют точка и метод, который вы собираетесь применить. Другими словами, команда `push('новый элемент')` не сработает. Сначала нужно указать имя массива (присвоенное ему при создании), за которым следует точка, а затем команда:

```
authors.push('Жюль Верн');
```

Выбор способа добавления элементов в массив

Итак, в этой главе рассказано о трех способах добавления элементов в массив. В табл. 2.5 приведено сравнение этих техник. Каждая из представленных команд решает похожие задачи, поэтому ваш выбор зависит от особенностей вашей программы. Если порядок, в котором хранятся элементы в массиве, не имеет значения, то подойдет любая команда. Например, у вас есть страница с изображениями товаров, и щелчок кнопкой мыши по картинке добавляет товар в корзину электронного магазина. Для хранения данных о выбранных товарах вы используете массив. Порядок, в котором элементы добавляются в корзину (или массив), не имеет значения, поэтому вы можете использовать любой из описанных способов.

Табл. 2.5. Способы добавления элементов в массив

Метод	Исходный массив	Пример кода	Итоговый массив	Объяснение
свойство `.length`	`var p = [0,1,2,3]`	`p[p.length]=4`	`[0,1,2,3,4]`	Одно значение добавляется в конец массива
команда `push()`	`var p = [0,1,2,3]`	`p.push(4,5,6)`	`[0,1,2,3,4,5,6]`	Один или несколько элементов добавляются в конец массива
команда `unshift()`	`var p = [0,1,2,3]`	`p.unshift(4,5)`	`[4,5,0,1,2,3]`	Один или несколько элементов добавляются в начало массива

Однако если вы создаете массив, учитывающий порядок добавления элементов, то метод имеет значение. Например, вы сделали страницу, позволяющую пользователям создавать список песен, щелкая кнопкой мыши по названию композиции на странице. Так как песни в списке перечислены в той последовательности, в которой они должны быть проиграны, порядок имеет значение. Поэтому чтобы при щелчке кнопкой мыши название песни попадало в конец списка (чтобы проигрываться в последнюю очередь), воспользуйтесь методом `push()`.

Команды `push()` и `unshift()` возвращают значение. То есть после выполнения своих задач команды `push()` и `unshift()` предоставляют число элементов в массиве:

```
var p = [0,1,2,3];
var totalItems = p.push(4,5);
```

После выполнения этого кода значение переменной `totalItems` будет равно 6, поскольку в массиве `p` теперь содержится шесть элементов.

Удаление элементов из массива

Если вы хотите удалить элемент из начала или из конца массива, используйте методы `pop()` или `shift()`. Первый удаляет элемент из конца массива, второй — из начала. В табл. 2.6 сравниваются два этих метода.

Табл. 2.6. Два способа удаления элемента из массива

Метод	Исходный массив	Пример кода	Итоговый массив	Объяснение
`pop()`	`var p = [0,1,2,3]`	`p.pop()`	`[0,1,2]`	Удаляет последний элемент из массива
`shift()`	`var p = [0,1,2,3]`	`p.shift()`	`[1,2,3]`	Удаляет первый элемент из массива

Аналогично методам `push()` и `unshift()`, команды `pop()` и `shift()` возвращают значения после выполнения своих задач — удаления элементов из массива. На самом деле они возвращают только что удаленное значение. Например, приведенный ниже код удаляет значение и сохраняет его в переменной `removedItem`:

```
var p = [0,1,2,3];
var removedItem = p.shift();
```

Значение `removedItem` после выполнения этого кода — `0`, а массив `p` теперь содержит элементы `[1,2,3]`.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Создание очереди

Методы добавления элементов в массив — `push()` и `unshift()` и методы удаления элементов из массива — `pop()` и `shift()` — часто используются вместе для доступа к элементам массива в том порядке, в котором они добавлялись. Классический пример — музыкальный плейлист. Вы создаете список, добавляя в него песни, затем, по мере проигрывания композиций, они удаляются из него. Песни проигрываются в том порядке, в котором они заносились в список, таким образом, сначала проигрывается первая песня, после чего она удаляется. Это похоже на очередь в кино. Вы занимаете место в конце очереди. Перед началом фильма двери открываются, и в зал заходит человек, который стоял в очереди первым.

В среде программистов данная концепция называется «первым пришел — первым ушел». Вы можете воспроизвести этот процесс, используя массивы и команды `push()` и `shift()`. Допустим, у вас есть массив `playlist`. Чтобы добавить новую песню в конец массива, вы используете метод `push()`:

```
playlist.push('Yellow Submarine');
```

Чтобы получить доступ к песне, которая должна проигрываться следующей, вы удаляете элемент, который является в списке первым на данный момент:

```
nowPlaying = playlist.shift();
```

Этот код удаляет из списка первый элемент массива и сохраняет его в переменной под названием `nowPlaying`. Концепция «первым пришел — первым ушел» полезна для создания очередей и управления ими, например, плейлистов, списков текущих дел или слайд-шоу.

ПРИМЕЧАНИЕ

В папке с учебными файлами есть веб-страница *массив_методы.html*, позволяющая тестировать различные команды массивов в интерактивном режиме. Она находится в каталоге *эксперименты*. Откройте файл в браузере и пощелкайте по кнопкам страницы, чтобы посмотреть, как работают методы массива. Кстати, своей интерактивностью данная страница обязана языку JavaScript и библиотеке jQuery.

Публикация текста на веб-странице с помощью массивов на практике

Вы будете использовать массивы для создания многих сценариев из этой книги. Чтобы получить представление о создании и использовании массивов, выполните следующие инструкции.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1, чтобы узнать, как скопировать файлы примеров.

1. **В текстовом редакторе откройте файл *02_03.html* из каталога *глава02*.**

Сначала вы создадите простой массив, содержащий четыре последовательности символов. Как и в случае с предыдущим руководством, в разделе заголовка и в теле данного документа уже содержатся элементы `script`.

2. **Между первой парой тегов элемента `script` напечатайте выделенный полужирным шрифтом код:**

```
<script>
var authors = [ 'Лев Толстой',
 'Николай Гоголь',
 'Иван Тургенев',
 'Антон Чехов'
];
</script>
```

В данном коде содержится одна инструкция JavaScript, но она разбита на пять строк.

Чтобы создать ее, напечатайте первую строку: `var authors = [ 'Лев Толстой'`, нажмите клавишу **Enter**, а затем вставьте столько пробелов, сколько необходимо, чтобы выровнять новое значение `'Николай Гоголь'` со значением из предыдущей строки.

ПРИМЕЧАНИЕ

Большинство HTML-редакторов использует *моноширинный* шрифт, например, `Courier` или `Courier New`, для HTML- или JavaScript-кода. В таких шрифтах каждый символ имеет ту же ширину, что и другие, с его помощью легко выравнивать столбцы (в данном примере — из имен авторов). Если в вашем текстовом редакторе нет подобного шрифта, то вам будет сложно выровнять имена.

Как уже говорилось, при создании массива с большим количеством элементов рекомендуется сделать код более удобным для чтения, разбив его на несколько строк. Вы можете с уверенностью сказать, что это единая инструкция, поскольку точка с запятой появляется только в конце пятой строки. Данный фрагмент кода создает массив `authors` и сохраняет в нем имена четырех авторов (четыре последовательности символов). Далее вам предстоит получить доступ к элементу массива.

3. **Найдите второй набор тегов элемента `script` и добавьте код, выделенный полужирным шрифтом:**

```
<script>
document.write('<p>Первый автор — <strong>');
document.write(authors[0] + '</strong></p>');
</script>
```

Первая строка начинает новый абзац небольшим текстом и открывающим тегом `<strong>` — это обычный HTML-код. В следующей строке используется первый элемент массива `authors`, а также закрывающие теги `</strong>` и `</p>` для завершения абзаца HTML. Для доступа к первому элементу массива вы используете в качестве индекса 0 — `authors[0]`, а не 1.

Теперь рекомендуется сохранить ваш файл и просмотреть его в браузере. Вы увидите на экране фразу «Первый автор — Лев Толстой». Если этого не произошло, то, возможно, вы допустили ошибку на шаге 2 или 3, когда создавали массив.

ПРИМЕЧАНИЕ

Не забудьте о возможности использования консоли ошибок вашего браузера для поиска ошибок в JavaScript-коде (см. раздел «Отслеживание ошибок» главы 1).

4. **Вернитесь в текстовый редактор и добавьте две следующие строки кода:**

```
document.write('<p>Последний автор — <strong>');
document.write(authors[4] + ' </strong></p>');
```

Этот шаг во многом дублирует предыдущий, за исключением того, что используется другой элемент массива. Сохраните страницу и просмотрите ее в браузере. Вместо имени автора вы увидите слово «undefined» (не определено) (рис. 2.6). Не беспокойтесь, так и должно быть.

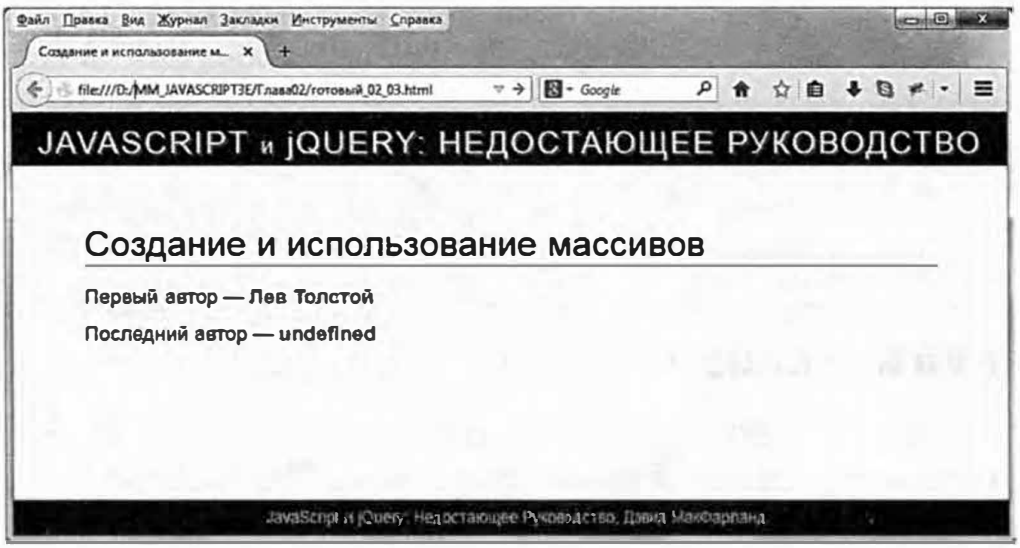

Рис. 2.6. При попытке запросить значение несуществующего элемента вы получите значение `undefined` (не определено)

Помните, что индексирование элементов массива начинается с 0, поэтому индекс последнего элемента определяется вычитанием 1 из общего числа элементов. В данном случае в массиве `authors` сохранены четыре строки, так что доступ к последнему элементу производится с помощью записи `authors[3]`.

ПРИМЕЧАНИЕ

Если вы попытаетесь запросить значение элемента, используя несуществующий индекс, то получите значение `undefined` (не определено). Это означает, что в данной позиции не сохранено никакое значение.

К счастью, существует простой способ извлечения последнего элемента массива вне зависимости от количества содержащихся в нем элементов.

5. **Вернитесь в текстовый редактор и исправьте только что введенный код. Удалите цифру «4» и добавьте вместо нее выделенный полужирным шрифтом код:**

```
document.write('<p>Последний автор — <strong>');
document.write(authors[authors.length-1] + '</↵
strong></p>');
```

Как вы помните из раздела «Добавление элементов в массив» свойство `length` содержит количество элементов массива. То есть общее количество элементов массива `писатели` можно выяснить с помощью кода `authors.length`. В данном случае мы получаем значение 4.

Зная, что индекс последнего элемента массива всегда на 1 меньше, чем общее количество элементов, вы просто вычитаете 1 из общего количества, чтобы получить значение индекса последнего элемента массива: `authors[authors.length-1]`.

ПРИМЕЧАНИЕ

Символ ↵, появляющийся в приведенном коде, означает, что данный фрагмент кода JavaScript — это одна строка. Поскольку ширина страниц этой книги иногда не позволяет напечатать часть кода в одну строку, символ ↵ используется, чтобы указать, что весь код должен находиться в одной строке. Если вы собираетесь написать этот код в текстовом редакторе, то напечатайте его в одну длинную строку (и опустите символ ↵).

Вы завершите данное упражнение, добавив еще один элемент в начало массива.

6. **Добавьте следующую строку кода после тех, которые вы создали на шаге 5:**

```
authors.unshift('Жюль Верн');
```

Как вы знаете, метод `unshift()` добавляет один или несколько элементов в начало массива. После выполнения данной строки кода массив `authors` будет выглядеть так: `['Жюль Верн', 'Лев Толстой', 'Николай Гоголь', 'Иван Тургенев', 'Антон Чехов']`.

Далее вы отобразите на странице только что добавленный элемент.

7. **Добавьте еще три строки (выделенные полужирным шрифтом), чтобы итоговый код выглядел так:**

```
document.write('<p>Первый автор — <strong>');
document.write(authors[0] + '</strong></p>');
document.write('<p>Последний автор — <strong>');
document.write(authors[authors.length-1] + ' </↵
strong></p>');
authors.unshift('Жюль Верн');
document.write('<p>Чуть не забыл, <strong>');
document.write(authors[0]);
document.write('</strong></p>');
```

Сохраните файл и просмотрите его в браузере. Вы должны увидеть нечто, похожее на рис. 2.7. Если что-то не получается, то не забывайте, что консоль ошибок может помочь обнаружить проблему.

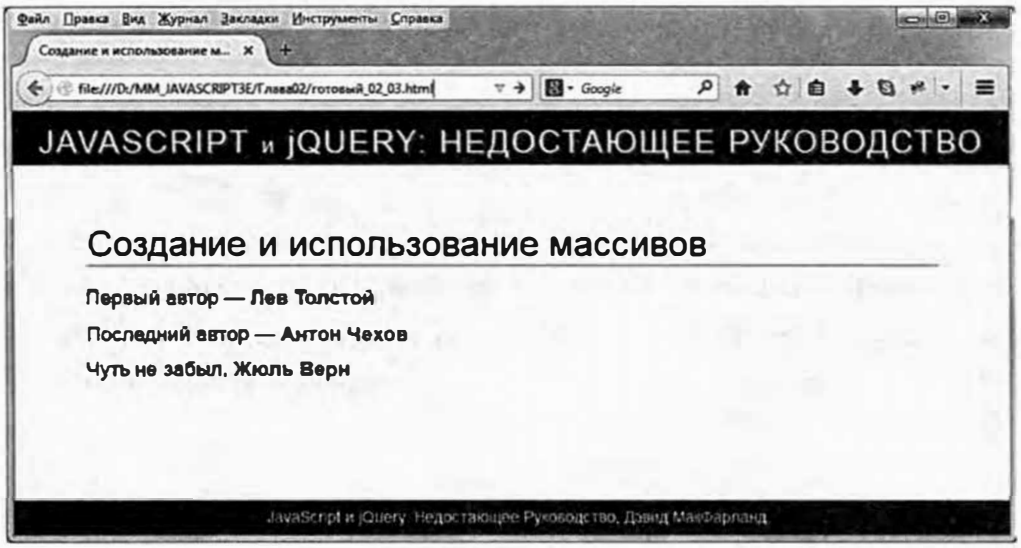

Рис. 2.7. Теперь вам должен быть понятен принцип работы массива

Вкратце об объектах

Вы уже знаете, что написать сообщение на веб-странице можно с помощью команды `document.write()`, а чтобы определить, сколько элементов содержит массив, необходимо указать его имя, за которым ввести точку и слово `length`, например, `days.length`. Возможно, вам интересно, для чего нужна эта точка. Вы уже сделали несколько упражнений, не задумываясь над особенностями этой части синтаксиса JavaScript. Пришло время о них поговорить.

Вы можете представить многие элементы языка JavaScript, а также элементы веб-страницы, как объекты. Реальный мир тоже наполнен

объектами, вроде машины или собаки. Большинство объектов состоят из различных частей: у собаки есть хвост, голова и четыре лапы; у машины — двери, колеса, фары, сигнал и т. д. Объект также может что-нибудь делать: машина перевозит пассажиров, собака лает. На самом деле даже часть объекта может что-то делать, например, хвост может вилять, а сигнал — сигналить. В табл. 2.7 показан один из способов наглядного представления отношений между объектами, их частями и действиями.

Табл. 2.7. Упрощенное восприятие мира

Объект	Часть	Действие
Собака		Лает
	Хвост	Виляет
Машина		Перевозит
	Сигнал	Сигналит

Мир JavaScript также полон объектов: окно браузера, документ, последовательность символов, дата и т. д. Как и объекты реального мира, объекты языка JavaScript состоят из разных частей. На языке программистов части объекта называются *свойствами*. Действия, которые может совершать объект, — это *методы*, которые являются функциями (например, `alert()`), характерными для данного объекта (табл. 2.8).

ПРИМЕЧАНИЕ

Вы всегда можете отличить метод от свойства, поскольку метод оканчивается круглыми скобками, например, `write()`.

Каждый объект JavaScript имеет собственный набор свойств и методов. Например, объект «массив» имеет свойство `length`, «документ» — метод `write()`. Чтобы получить доступ к свойству объекта или выполнить одну из его функций, используется *точечный синтаксис*, то есть просто точки. Точка объединяет объект с его свойством или методом. Например, команда `document.write()` означает «запусти метод `write()` объекта `document`». Если бы это работало в реальном мире, то собака виляла бы хвостом так: `собака.хвост.вилять()` (однако в реальном мире гораздо эффективнее действует угощение).

Табл. 2.8. Некоторые методы и свойства объектов «массив» и «документ»

Объект	Свойство	Метод
`document`	`title`	
	`url`	
		`write()`
`['Катя', 'Григорий', 'Семен']`	`length`	
		`push()`
		`pop()`
		`unshift()`

И так же, как в реальности, у вас может быть несколько собак, ваши программы JavaScript могут иметь несколько объектов одного и того же рода (экземпляров). Например, вы создаете две простые переменные:

```
var first_name = 'Валет';
var last_name = 'Червей';
```

Вы только что создали два различных объекта типа *строка*. Строки имеют собственный набор свойств и методов, которые отличаются от свойств и методов других объектов, например, дат (вы узнаете о некоторых из этих свойств и методов в разделе «Дата и время» главы 16). Когда вы создаете объект (это также называется созданием *экземпляра объекта*), то получаете доступ ко всем его свойствам и методам.

ПРИМЕЧАНИЕ

Вам уже знаком еще один объект — `window`, представляющий окно браузера. По сути это объект-контейнер для веб-страницы. Например, функции `alert()` и `prompt()` являются методами объекта `window` и могут записываться таким образом: `window.alert()` и `window.prompt()`. Однако, поскольку объект `window` всегда присутствует на веб-странице, вы можете не указывать его имя, то есть записи `alert('привет')` и `window.alert('привет')` — равнозначны.

Вводя новую переменную и сохраняя в ней значение, вы создаете новый экземпляр определенного класса объектов. Так, каждая из следующих строк создает экземпляр отдельного класса объектов JavaScript:

```
var first_name = 'Вова'; // строковый объект
var age = 32; // числовой объект
var valid = false; // логический объект
```

На самом деле, когда вы меняете тип данных, хранящихся в переменной, вы также изменяете тип объекта. Например, если вы создаете переменную `data`, сохраняющую массив, а потом сохраняете в этой переменной число, то вы изменяете тип переменной с «массива» на «число»:

```
var data = false; // логический объект
data = 32; // изменяется на числовой объект
```

Концепции объектов, свойств, методов и точечного синтаксиса на первый взгляд могут показаться немного странными. Однако, поскольку они являются основой того, как работает JavaScript, а также неотъемлемой частью использования библиотеки jQuery, вы очень быстро к ним привыкнете.

▶ **СОВЕТ**

Язык JavaScript предусматривает специальное ключевое слово для определения типа объекта (string, number, Boolean и т. д.). Это так называемая операция `typeof`, помещаемая перед переменной для указания типа объекта, хранящегося в этой переменной. Например:

```
var data = 32;
alert(typeof data); // в окне оповещения указано: "number"
data = 'Иван Иванов';
alert(typeof data); // в окне оповещения указано: "string"
```

В процессе чтения этой книги помните:

- В мире JavaScript существует большое количество разнообразных типов объектов.
- У каждого объекта есть собственные свойства и методы.
- Чтобы получить доступ к свойству объекта или выполнить его метод, вы используете точечный синтаксис. Например, `document.write()`.

Комментарии

Бывает, что вы захвачены процессом программирования и чувствуете, что понимаете все, что происходит в вашей программе. Каждая строка кода имеет смысл, и более того, она работает! Но через месяц-два, когда начальник или клиент просит вас изменить программу или добавить новую возможность в ваш классный сценарий, вы можете только почесать затылок, посмотрев на когда-то знакомый код JavaScript. Что это за переменная? Почему я написал именно так? Что происходит в этой части программы?

Достаточно легко забыть, как работает программа и почему вы написали код так, а не иначе. К счастью, большинство языков программирования позволяют оставлять заметки для себя или других программистов, которые могут просматривать ваш код. Язык JavaScript позволяет вставлять *комментарии* прямо в код. Если вы использовали комментарии при работе с языками HTML и CSS, то этот процесс будет вам знаком. Комментарий — это просто одна или несколько строк текста; интерпретатор JavaScript игнорирует его, но он может предоставить ценную информацию о том, как работает ваша программа.

Для создания однострочного комментария поставьте перед ним две косые черты:

```
// Это комментарий
```

Вы также можете добавить комментарий после инструкции JavaScript:

```
var price = 10; // установка начальной стоимости виджета
```

Интерпретатор JavaScript выполнит все, что записано в этой строке до символов //. Затем он перейдет к следующей строке.

Вы также можете добавить несколько строк, начав комментарий с сочетания символов /* и закончив его символами */. (Аналогичный тип комментариев используется в языке CSS.) Интерпретатор JavaScript игнорирует весь текст между этими наборами символов. Допустим, вы хотите описать работу программы в начале кода. Можете сделать это следующим образом:

```
/*
JavaScript слайд-шоу:
Эта программа автоматизирует вывод изображений
во всплывающем окне
*/
```

Вам не обязательно помещать символы /* и */ на отдельные строки. Вы можете использовать их для создания комментария JavaScript в одну строку:

```
/* это комментарий в одну строку */
```

Если вы хотите написать краткий однострочный комментарий, используйте //. Для создания комментария в несколько строк воспользуйтесь символами /* и */.

Использование комментариев

Комментарии — незаменимый инструмент в умеренно длинных и сложных программах, которые вы собираетесь использовать достаточно долго или изменять в будущем. Пока что вы научились создавать небольшие сценарии, состоящие из одной-двух строк кода, однако со временем вы начнете писать гораздо более объемные и сложные программы. Чтобы с легкостью узнать, что происходит в сценарии, добавьте в него комментарий для понимания общей логики программы и объяснения отдельных путаных или сложных фрагментов.

ПРИМЕЧАНИЕ

Если в сценарий добавить много комментариев, он станет больше и будет медленнее загружаться. В большинстве случаев объем комментариев не сильно влияет на размер файла, однако если вам необходимо его уменьшить, обратитесь к разделу «Оптимизация сценариев JavaScript» главы 16, чтобы узнать о способах, помогающих сделать файлы JavaScript меньше и быстрее.

Многие программисты добавляют блок комментариев в начало внешнего файла JavaScript. В таких комментариях объясняется, для чего предназначен сценарий, определяется дата его создания, приводится номер версии (для часто обновляемых сценариев) и информация об авторских правах.

Например, в начале файла из библиотеки JavaScript jQuery вы найдете такой комментарий:

```
/*!
* jQuery JavaScript Library v1.11.0
* http://jquery.com/
*
* Includes Sizzle.js
* http://sizzlejs.com/
*
* Copyright 2005, 2014 jQuery Foundation, Inc. and
other contributors
* Released under the MIT license
* http://jquery.org/license
```

```
*
* Date: 2014-01-23T21:02Z
*/
```

В начало сценария вы также можете поместить инструкции по использованию: переменные, которые нужно установить, что-то, что необходимо сделать с HTML-кодом, чтобы ваш сценарий работал и т. д.

Следует помещать комментарий перед серией сложных этапов программирования. Например, вы пишете сценарий, перемещающий изображение по экрану пользователя. Одна часть этого сценария определяет текущее положение картинки в окне браузера. Эта часть может содержать несколько строк кода, поэтому было бы неплохо поместить комментарий перед подобной частью программы, чтобы, взглянув на сценарий позже, вы могли точно вспомнить, что он делает:

```
// Определение позиций x и y изображения в окне
```

Как правило, следует добавлять комментарии везде, где они могут пригодиться позднее. Если строка кода очевидна, то комментарий, возможно, и не нужен. Например, нет необходимости комментировать простой код вроде `alert('привет')`, поскольку совершенно очевидно, для чего он предназначен (для открытия окна оповещения со словом «привет»).

Комментарии в этой книге

Комментарии также очень полезны при объяснении кода JavaScript. В этой книге комментарии часто рассказывают о том, что делает строка кода, или показывают результат выполнения определенной команды. Например, вы можете увидеть комментарий, описывающий результат выполнения команды оповещения:

```
var a = 'Иван';
var b = 'Иванов';
alert(a + ' ' + b); // 'Иван Иванов';
```

Третья строка с комментарием указывает на то, что вы должны увидеть, просматривая код в браузере. Если вы захотите протестировать код, приведенный в книге, добавив его на веб-страницу и просмотрев в браузере, то можете опустить комментарии, вводя код на веб-страницу.

Они предназначены просто для того, чтобы помочь вам понять, что происходит в коде.

Изучая более сложные команды JavaScript, вы начнете манипулировать данными, содержащимися в переменных. В этой книге вы будете часто видеть комментарии к коду, в которых указано, что должно быть сохранено в переменной после выполнения команды. Например, команда `charAt()` позволяет выбрать символ в указанном месте строки. Когда вы будете читать о том, как использовать эту команду, вы можете увидеть такой код:

```
var x = "Настало время хороших программистов.";
alert(x.charAt(2)); // 'c'
```

Комментарий `// 'c'`, находящийся в конце второй строки, показывает то, что вы должны увидеть в окне оповещения, если выполнить код в браузере. И, конечно, `c` — это правильно. При перечислении символов строки, первый — имеет индекс 0. Так что команда `charAt(2)` возвращает третий символ последовательности.

Глава 3
ДОБАВЛЕНИЕ В ПРОГРАММУ ЛОГИКИ И КОНТРОЛЯ

Итак, вы уже освоили некоторые основы языка JavaScript. Но просто создавая переменную и сохраняя в ней число или последовательность символов, вы не достигнете многого. А построение массива с длинным списком элементов будет не слишком полезным, если с ними сложно работать. В данной главе вы узнаете, как заставить вашу программу интеллектуально реагировать и работать более эффективно благодаря использованию управляющих инструкций, циклов и функций.

Интеллектуальная реакция программы

Жизнь постоянно ставит нас перед выбором: «Что мне сегодня надеть?», «Что съесть на завтрак?», «Что я делаю в пятницу вечером?» и т. д. Часто выбор зависит от сопутствующих обстоятельств. Например, вы решили сходить в пятницу вечером в кино. Вероятно, вы зададите себе ряд вопросов: «Будет ли в пятницу хороший фильм?», «Когда начинается сеанс?», «Хватит ли у меня денег на билет (и на воздушную кукурузу)?».

Допустим, именно в то время, когда вы хотите отдохнуть, идет хороший фильм. Тогда вы задаете себе простой вопрос: «Достаточно ли у меня денег?». Если ответ утвердительный, то вы направляетесь в кино. Если ответ отрицательный, то вы никуда не пойдете. Но в следующую пятницу у вас будет достаточно денег, и вы пойдете в кино. Этот сценарий — всего лишь простой пример того, как обстоятельства влияют на принимаемые нами решения.

Язык JavaScript располагает подобной возможностью принятия решений, называемой *«управляющие инструкции»*. Упрощенно, управляющая инструкция— это обычный вопрос, который предполагает ответ «да» или «нет». Если ответ на вопрос — «да», то программа выполняет одно действие; если ответ — «нет», то программа делает что-то другое.

Управляющие инструкции — это одна из наиболее важных концепций в программировании: они позволяют вашим программам реагировать на различные ситуации и вести себя интеллектуально. Вы будете пользоваться такими инструкциями бессчетное количество раз. Чтобы продемонстрировать их полезность, рассмотрим несколько примеров того, как их можно использовать.

- **Проверка форм.** Когда вы хотите убедиться, что кто-либо заполнил в веб-форме все необходимые поля («Имя», «Адрес», «Электронная почта» и т. д.), вы станете использовать управляющие инструкции. Например, если поле «Имя» пусто, отправить данные формы невозможно.
- **Перетаскивание.** Если вы добавляете возможность перетаскивания элементов в пределах вашей страницы, вы, возможно, захотите проверить, куда посетитель переместил элемент. Например, если он перетаскивает фотографию на изображение корзины, вы заставляете фотографию исчезнуть со страницы.

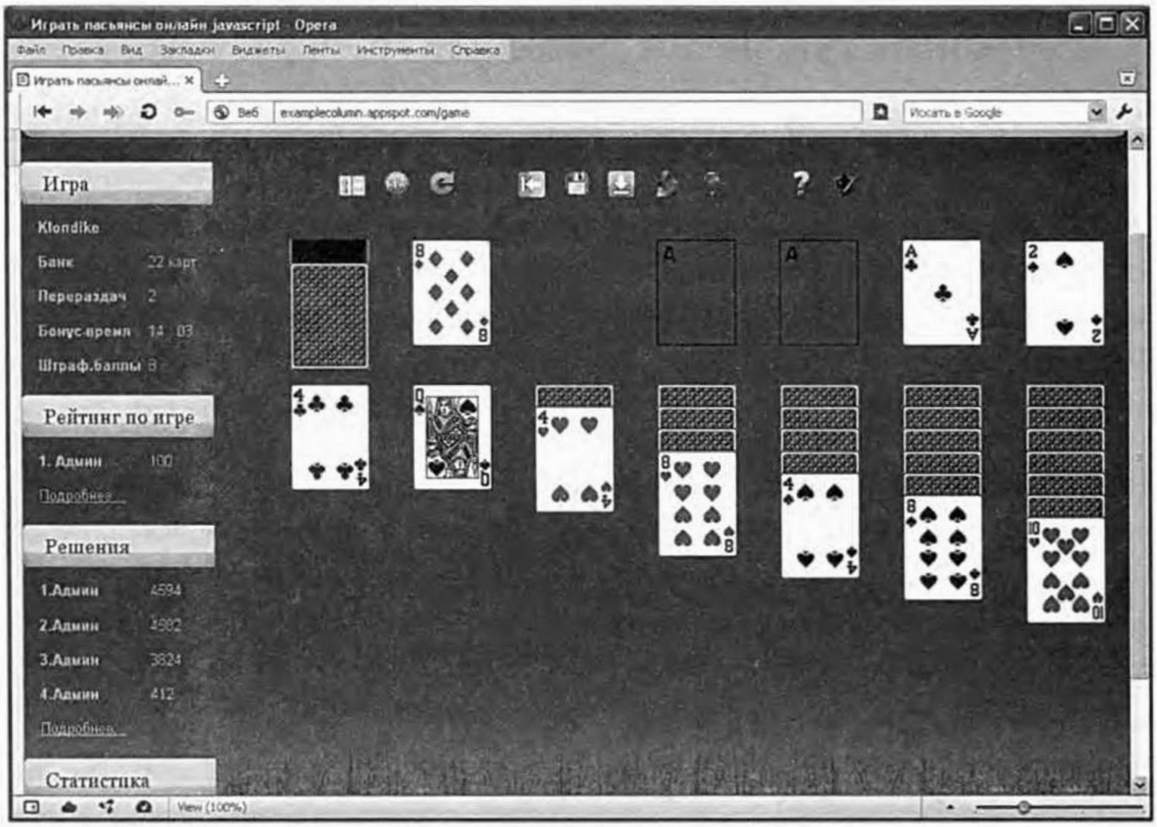

Рис. 3.1. Чтобы развлечься, нужно много поработать. Игра на основе языка JavaScript, такая как «Клондайк» (**examplecolumn.appspot.com/game**), показывает, как программа может по-разному реагировать в зависимости от условий. Например, когда игрок перетаскивает карту, программа должна решить, кладет ли он ее в допустимое место, а затем в каждом случае она действует по-разному

- **Оценка пользовательского ввода.** Если вы создаете всплывающее окно, в котором посетителю задается вопрос: «Не хотите ли вы ответить на пару вопросов о том, как великолепен этот сайт?», то реакция вашего сценария должна быть различной в зависимости от того, как ответит посетитель.

На рис. 3.1 продемонстрировано приложение, использующее управляющие инструкции.

Управляющие инструкции

Управляющие инструкции, также называемые выражениями *если/то*, выполняют задачу, только если ответ на вопрос — истина: «*Если* у меня достаточно денег, *то* я иду в кино». Структура управляющей инструкции выглядит так:

```
if ( условие ) {
// Здесь совершается некое действие
}
```

Инструкция состоит из трех частей: `if` сообщает о том, что программа, следующая далее, является управляющей инструкцией; в скобках находится вопрос, требующий ответа «да» или «нет», называемый условием (подробнее о нем ниже); фигурные скобки (`{}`) обозначают начало и конец JavaScript-кода, который должен выполняться, если инструкция истинна.

ПРИМЕЧАНИЕ

Строка `// Здесь совершается некое действие` — это комментарий. Это не код, который выполняется, а просто заметка, указывающая читателю, что должно происходить в этой части программы. Подробнее о комментариях рассказывалось в разделе «Комментарии» главы 2.

Во многих случаях условие — это сравнение двух величин. Например, вы написали игру, в которой игрок побеждает, если набирает более 100 очков. В данной программе вам нужна переменная, в которой сохраняются и обновляются очки игрока. В определенный момент вам придется проверить, превышает ли количество очков 100. Код JavaScript, позволяющий это сделать, выглядит так:

```
if (score > 100) {
alert('Вы победили!');
}
```

Важная часть — `score > 100`. Это условие, которое проверяет, превышает ли значение переменной `score` число 100. Если да, то появляется окно «Вы победили!»; если же количество очков меньше или равно 100, то интерпретатор JavaScript пропускает команду оповещения и переходит к выполнению следующей части программы. На рис. 3.2 наглядно представлен данный процесс.

Рис. 3.2. В присутствии управляющей инструкции код в фигурных скобках выполняется, только если условие — истина. Если условие — ложь, то этот фрагмент кода пропускается, и программа продолжает выполняться

Кроме символа > (больше), существуют и другие управляющие операции, используемые для сравнения чисел (табл. 3.1).

Табл. 3.1. Используйте эти операции сравнения в качестве части управляющих инструкций

Операция сравнения	Что означает
`==`	**Равно.** Сравнивает две величины на предмет идентичности. Может использоваться для сравнения чисел или строк
`!=`	**Не равно.** Сравнивает две величины на предмет неравенства друг другу. Может использоваться для сравнения чисел или строк
`===`	**Строго равно.** Сравнивает не только значения, но и типы данных. Другими словами, два значения должны быть одного типа (строкового, числового или логического), чтобы условие было истинным. Например, хотя выражение `'2'==2` истинно, выражение `'2'===2` — не истинно, так как первое значение — это строка (заключено в кавычки), а второе — число. Многие программисты предпочитают использовать данную операцию, поскольку она гарантирует сравнение информации одного

Операция сравнения	Что означает
	и того же типа. Однако при извлечении числового значения из формы (или диалогового окна `prompt()`) вы получите строковое значение `'2'`, а не число 2. Вам следует всегда преобразовывать строку в число перед проведением сравнения (см. раздел «Работа с типами данных и переменными» главы 2)
`!= =`	**Строго не равно.** Также как и операция «строго равно», сравнивает не только значения, но и типы данных. Например, хотя выражение `'2'!=2` ложно, выражение `'2'!==2` истинно, так как несмотря на равенство значений, типы данных отличаются
`>`	**Больше.** Сравнивает два числа и проверяет, является ли число слева большим числа справа от знака неравенства. Например, `2>1` истинно, так как число 2 больше, чем 1, но `2>3` ложно, так как 2 не больше 3
`<`	**Меньше.** Сравнивает два числа и проверяет, является ли число слева меньшим числа справа от знака неравенства. Например, `2<3` истинно, так как число 2 меньше, чем 3, но `2<1` ложно, так как 2 не меньше, чем 1
`>=`	**Больше или равно.** Сравнивает два числа и проверяет, является число слева большим или равным числу справа. Например, `2>=2` истинно, так как 2 равно 2, но `2>=3` ложно, так как 2 ни больше 3, ни равно 3
`<=`	**Меньше или равно.** Сравнивает два числа и проверяет, является число слева меньшим или равным числу справа. Например, `2<=2` истинно, так как 2 равно 2, но `2<=1` ложно, так как 2 ни меньше 1, ни равно 1

СОВЕТ

Введите два пробела (или один раз нажмите клавишу **Tab**) в начале каждой строки кода, находящегося в фигурных скобках. Пробелы (или табуляция) создают отступ для этих строк и помогают увидеть начальную и конечную фигурные скобки, а также определить, какой код относится к управляющей инструкции. Два пробела — обычная практика. Но если ваш код становится более удобным для чтения при вставке четырех пробелов, то используйте четыре пробела. В примерах данной книги строки кода в скобках набраны с отступом.

Чаще всего вы будете проверять, равны два значения или нет. Например, вы создали викторину на языке JavaScript, и один из ее вопросов: «Сколько лун у планеты Сатурн?». Ответ посетителя сохраняется в переменной `answer`. Вы можете написать такую управляющую инструкцию:

```
if (answer == 31) {
alert('Верно. У Сатурна 31 луна.');
}
```

Два знака равенства (==) — не опечатка. Это инструкция для интерпретатора JavaScript сравнить два значения и решить, равны ли они. Как вы помните, одинарный знак равенства в языке JavaScript называется *операцией присваивания*, который используется для сохранения значения в переменной:

```
var score = 0; // сохраняет значение 0 в переменной score
```

Поскольку интерпретатор JavaScript уже присвоил специальное значение одинарному знаку равенства, вы должны использовать два знака равенства, когда хотите сравнить два значения и узнать, равны они или нет.

Вы также можете использовать символы == (так называемую *операцию равенства*), чтобы проверить, одинаковы ли две строки. Например, вы разрешаете пользователю задавать в форме определенный цвет. Если вводится значение `'red'`, то цвет фона страницы изменяется на красный. Для осуществления подобного действия используйте управляющую операцию:

```
if (enteredColor == 'red') {
document.body.style.backgroundColor='red';
}
```

ПРИМЕЧАНИЕ

Пока не волнуйтесь насчет того, как именно изменяется цвет страницы. Вы изучите динамический контроль над свойствами CSS с использованием языка JavaScript в разделе «Чтение и изменение свойств CSS» главы 4.

Вы также можете проверить, являются ли значения неравными, используя *операцию неравенства*:

```
if (answer != 31) {
alert("Неверно! У Сатурна совсем не столько лун.");
}
```

Восклицательный знак переводится как «нет», то есть операция `!=` означает — «не равно». В данном случае, если значение, сохраненное в переменной `answer`, не равно 31, то незадачливый посетитель, проходящий тест, получит обидное сообщение.

Код, выполняемый при условии истинности инструкции, не ограничивается одной строкой, как в предыдущих примерах. Между открывающей и закрывающей фигурными скобками разрешается поместить столько угодно строк кода JavaScript. Продолжим пример с викториной, вы можете производить подсчет правильных ответов посетителя, проходящего тест.

То есть если на вопрос о Сатурне получен правильный ответ, вы можете добавить к сумме очков испытуемого 1. Допустимо сделать это с помощью следующей управляющей инструкции:

```
if (answer == 31) {
alert('Верно. У Сатурна 31 луна.');
numCorrect += 1;
}
```

ПРИМЕЧАНИЕ

Как отмечалось в главе 2, строка кода: `numCorrect += 1` просто прибавляет 1 к значению переменной `numCorrect`.

Вы можете добавить дополнительные строки кода JavaScript между скобками — любого кода, который необходимо выполнить в случае истинности условия.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Возвращение логических типов данных

В разделе «Типы данных» главы 2 вы узнали о логических значениях: *true* (истина) и *false* (ложь). Поначалу логические значения могут показаться не слишком полезными, но вы увидите, насколько они важны, когда начнете использовать управляющие инструкции. Поскольку условие является вопросом с ответом «да» или «нет», ответ на этот вопрос является логическим значением. Например, обратите внимание на следующий код:

```
var x = 4;
if ( x == 4 ) {
// некие действия
}
```

Первая строка кода сохраняет число 4 в переменной x. Условие в следующей строке — простой вопрос: равно ли 4 значение переменной x? В данном случае это так, поэтому выполняется код JavaScript, заключенный в фигурные скобки. А вот что происходит в круглых скобках: интерпретатор JavaScript превращает условие в логическое значение, говоря языком программирования, он *оценивает* условие. Если условие оценено как «истина» (то есть ответ на вопрос — «да»), то выполняется код в фигурных скобках. Если условие оценивается как «ложь», то код в фигурных скобках пропускается.

Часто логическое значение используется для создания того, что называется *флагом* — переменной, которая отмечает, что какое-то из условий не является истинным. Например, если для заполнения формы нужно большое количество сведений от посетителя, вы можете начать с создания «переменной-флага» `valid` с логическим значением `true`. Это значит, вы допускаете, что форма заполнена правильно. Потом вы проверяете данной переменной каждый ее элемент, и если в элементе формы не хватает информации или тип данных неверен, вы изменяете значение переменной `valid` на `false`. Проверив все элементы формы, вы смотрите, что сохранено в переменной `valid`, и если ее значение — по-прежнему `true`, то форма отправляется. Если значение переменной — `false` (то есть одно или более элементов не заполнены), вы отображаете какое-либо сообщение об ошибке, и данные формы отправлены не будут:

```
var valid = true;
// здесь производятся различные действия
// если элемент заполнен не правильно, тогда значение
//переменной valid меняется на false
if (valid) {
// форма отсылается
} else {
// отображаются сообщения об ошибках
}
```

Запасной план

Но что, если условие ложно? Простейшая управляющая инструкция в предыдущем разделе не имеет альтернативного плана действий на случай, если условие оказывается ложным. В реальной жизни, когда вы решаете, что делать в пятницу вечером, и у вас не хватает денег на кино, приходится выбрать что-то *другое*. Инструкция `if` имеет собственный

запасной план, называемый условием *else*. Допустим, в своей викторине вы хотите уведомить пользователя, проходящего тест, правильно он ответил на вопрос или нет. Вот, как вы можете это сделать:

```
if (answer == 31) {
alert('Верно. У Сатурна 31 луна.');
numCorrect = numCorrect + 1;
} else {
alert ("Неверно! У Сатурна совсем не столько лун. ");
}
```

Этот код устанавливает ситуацию «или-или»: может появиться только одно из двух сообщений (рис. 3.3). Если в переменной `answer` сохранено значение `31`, то появляется сообщение «верно», в ином случае появляется сообщение «неверно».

Рис. 3.3. При использовании условия `if/else` вы включаете два фрагмента кода, но выполнен будет только один. Если условие истинно, то выполняется код в скобках, следующий прямо за условием (слева); однако если условие ложно, то выполняется код в скобках, следующий за словом `else` (справа)

Для создания альтернативного условия добавьте после закрывающей скобки слово `else`, за которым следует еще одна пара скобок. В них вы вводите код, который должен выполняться, если условие оказывается ложным. Альтернативное условие может включать столько строк кода, сколько вы пожелаете.

Проверка более одного условия

Можно протестировать несколько условий и получить несколько результатов: представьте, что речь идет о телешоу, в котором ведущий

говорит: «Вы хотели бы приз, находящийся за дверью 1, дверью 2 или дверью 3?» Вы можете выбрать только одну дверь. В повседневных делах вы часто сталкиваетесь с выбором, подобным этому.

Например, вернемся к вопросу: «Что я буду делать в пятницу вечером?». Вы можете продумывать варианты своего отдыха в зависимости от того, сколько у вас денег и сколько вы готовы потратить. Например, вы начинаете рассуждать так: «Если у меня 1500 рублей и больше, то я хорошо поужинаю и схожу в кино (и на воздушную кукурузу мне тоже хватит)».

Если у вас нет 1500 рублей, вы можете провести другой тест: «Если у меня есть 1000 рублей и больше, то я хорошо поужинаю». Если у вас нет 1000 рублей, вы пробуете следующее: «Если у меня 500 рублей или больше, то я схожу в кино». И, наконец, если у вас нет 500 рублей, вы можете сказать: «Я останусь дома и посмотрю телевизор».

Язык JavaScript позволяет работать с подобными примерами многоуровневой логики, используя инструкции `else if` *(если нет... то)*. Принцип работы таков: вы начинаете с инструкции `if`, являющейся вариантом № 1, затем добавляете одно или более условий `else if` (если нет... то) для дополнительных вопросов, которые могут запускать дополнительные сценарии; наконец, вы используете альтернативное условие в качестве запасного варианта. Ниже представлена структура таких операций в JavaScript:

```
if (условие) {
// дверь #1
} else if (условие2) {
// дверь #2
} else {
// дверь #3
}
```

Такая структура — это все, что вам нужно для создания JavaScript-программы «Планирование вечера в пятницу». Она спрашивает посетителей, сколько у них денег, а затем определяет, что им делать в пятницу (звучит знакомо?). Вы можете использовать команду `prompt()`, о которой узнали в разделе «Запрос информации на практике» главы 2, для сбора ответов пользователя и серию инструкций `if/else if` *(если/если нет... то)* для определения того, что нужно делать:

```
var fridayCash = prompt('Сколько денег вы можете ↵
потратить?', '');
if (fridayCash >= 1500) {
alert('Вы можете и поужинать, и пойти в кино.');
} else if (fridayCash >= 1000) {
alert('Вы можете поужинать.');
} else if (fridayCash >= 500) {
alert('Вы можете пойти в кино.');
} else {
alert('Похоже, вы будете смотреть телевизор.');
}
```

Далее представлен пошаговый разбор этой программы: первая строка открывает диалоговое окно, спрашивающее посетителя о том, сколько он может потратить. То, что вводит посетитель, сохраняется в переменной с именем `fridayCash`. Следующая строка — это тест: величина, введенная посетителем, больше или равна 1500? Если ответ — «да», то появляется сообщение, предлагающее поужинать и посмотреть кино. В данном случае управляющая инструкция полностью выполняется. Интерпретатор JavaScript пропускает следующие две инструкции `else if` и последнюю инструкцию `else`. В случае с управляющей инструкцией может быть только один итог, то есть когда интерпретатор JavaScript сталкивается с условием, которое оценивается как истинное, запускается код JavaScript в скобках, соответствующий этому условию, и пропускает всю остальную информацию в пределах управляющей инструкции (рис. 3.4).

ПРИМЕЧАНИЕ

При оценке управляющей инструкции интерпретатор JavaScript пытается найти «истинное» значение. Язык JavaScript предусматривает концепцию «истинных» и «ложных» значений, которая выходит за рамки значений `true` и `false`. Звучит сложно? Не беспокойтесь, далее в книге вы узнаете об этой концепции подробнее.

Предположим, посетитель ввел значение *700*. Тогда первое условие не будет истинным, поскольку 700 меньше 1500. То есть интерпретатор JavaScript пропускает код в скобках, относящийся к первому условию, и переходит к инструкции `else if`: «700 больше или равно 1000»? Так

как ответ отрицательный, то он пропускает код, связанный с этим условием, и переходит к следующей инструкции `else if`: В данном случае выясняется, является ли 700 большим или равным 500. Ответ утвердителен, поэтому появляется окно с сообщением: «Вы можете пойти в кино», и программа завершается, пропуская конечное альтернативное условие.

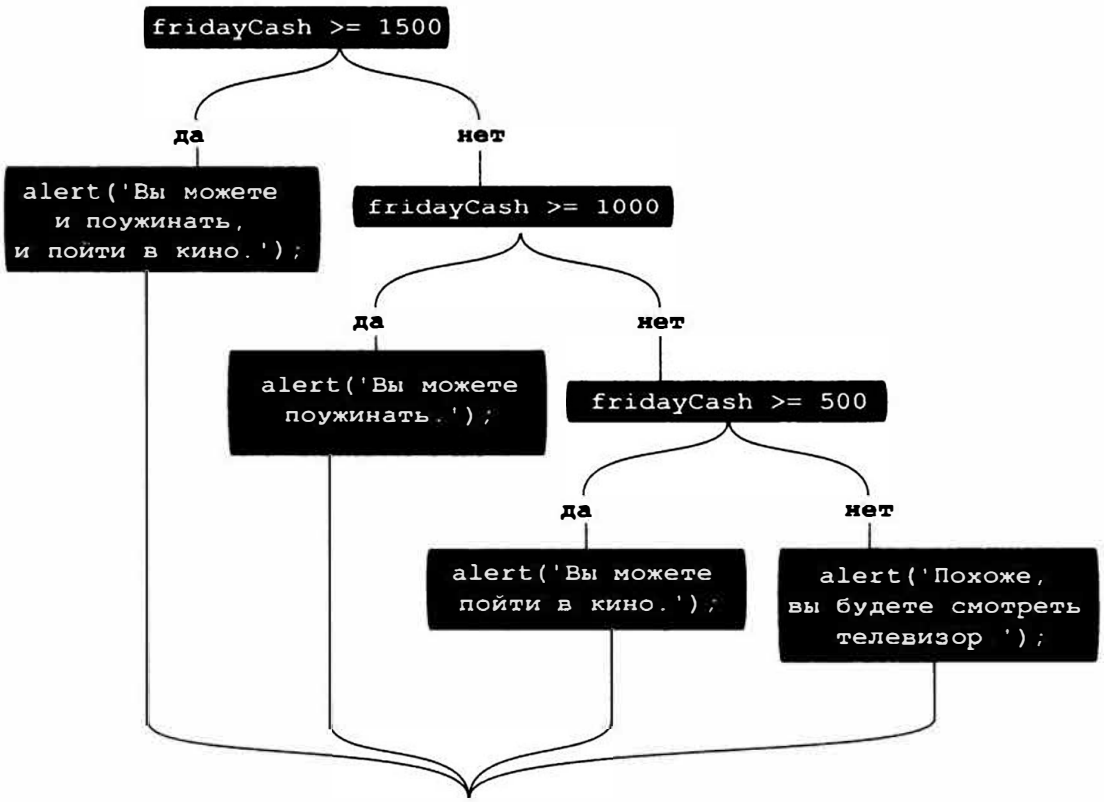

Рис. 3.4. В случае с элементарной управляющей инструкцией код, заключенный в фигурные скобки, выполняется, только если условие является истинным. Если условие является ложным, то этот код пропускается, и программа продолжает выполняться. Вы найдете рабочий пример этого кода в файле *вечер_пятницы.html* в папке *глава03* среди примеров к книге

Порядок следования ваших управляющих инструкций может повлиять на работу программ. Например, вы поменяли местами некоторые инструкции:

```
var fridayCash = prompt('Сколько денег вы можете ↵
потратить?', '');
if (fridayCash >= 500) {
alert('Вы можете пойти в кино.');
} else if (fridayCash >= 1000) {
```

```
alert('Вы можете поужинать.');
} else if (fridayCash >= 1500) {
alert('Вы можете и поужинать, и пойти в кино.');
} else {
alert('Похоже, вы будете смотреть телевизор.');
}
```

Вне зависимости от того, сколько у вас денег, вы никогда не дойдете до варианта «Вы можете поужинать» или «Вы можете и поужинать, и пойти в кино». Если у вас есть 3000 руб., программа сначала спросит: 3000 >=500?. Разумеется, ответ окажется утвердительным, поэтому вы увидите сообщение «Вы можете пойти в кино», а остальные условия просто будут проигнорированы, хотя у вас есть деньги на другие варианты.

Вы столкнетесь с этой проблемой, когда станете сравнивать численные значения, используя операцию < или >. Поскольку множество значений могут быть больше или меньше определенного числа, существует множество положительных ответов. Когда вы проверяете, не равно ли значение переменной определенному числу, существует только один возможный вариант, поэтому такие инструкции можно ставить в любом порядке в серии условий `if / else if /`.

СОВЕТ

Существует другой способ создания серий управляющих инструкций, которые проверяют одну и ту же переменную, как в приведенном примере. Инструкции `switch` производят ту же операцию (см. раздел «Оптимизация сценариев JavaScript» главы 16).

Более сложные условия

Если вы имеете дело с несколькими различными переменными, то часто необходимы еще более сложные управляющие инструкции. Например, если для отправки формы необходимо указать адрес электронной почты, вам потребуется убедиться и в том, что поле заполнено, и в том, что содержащаяся в поле информация — это адрес электронной почты (а не просто случайным образом набранные буквы). К счастью, язык JavaScript позволяет вам выполнить и такую проверку.

Проверка истинности более одного условия

Часто вам приходится принимать решение, основываясь на нескольких факторах. Например, вы хотите пойти в кино, если у вас достаточно денег *и* идет именно тот фильм, который вы хотели посмотреть. В данном случае вы пойдете в кино, только если выполняются оба условия. Если ложно хотя бы одно из них, то вы не пойдете в кино.

В JavaScript вы можете комбинировать условия, используя *операцию логического «И»* (обозначается символами `&&`). Вы можете добавлять ее между двумя условиями в рамках одной управляющей инструкции. Например, если вы хотите проверить, принадлежит ли число промежутку от 1 до 10, сделайте следующее:

```
if (a > 1 && a < 10) {
// Значение a находится между 1 и 10
alert("Значение " + a + " принадлежит промежутку ↵
от 1 до 10");
}
```

В данном примере — два условия: `a > 1` равнозначно вопросу: «Значение переменной `a` больше `1`?»; второе условие `a < 10` проверяет, является ли значение переменной `a` меньшим `10`. Код JavaScript, содержащийся в скобках, будет выполнен, только если верны *оба* условия. То есть, если в переменной `a` хранится значение `0`, то первое условие (`a < 10`) выполняется (0 меньше 10), а второе — нет (0 не больше 1).

Вы не ограничены всего двумя условиями. С помощью операции `&&` можно объединять сколько угодно условий:

```
if (b>0 && a>0 && c>0) {
// Все три переменные больше 0
}
```

Этот код проверяет три переменные, чтобы убедиться, что все они больше 0. Если хотя бы одна из них равна или меньше 0, код в скобках не будет выполнен.

Проверка истинности как минимум одного условия

Иногда может понадобиться проверить серию условий, но истинным должно быть только одно. Например, вы добавили для клавиатуры пользователя функцию перехода от картинки к картинке в фотогалерее.

Если посетитель нажимает клавишу **N**, появляется следующая фотография. В данном случае может потребоваться, чтобы переход к следующей фотографии происходил, если он нажимает **n** (нижний регистр) или **N** (верхний регистр, при нажатой клавише **CapsLock**). Здесь вы нуждаетесь в логике *или... или*: могут быть нажаты оба варианта. *Операция логического «ИЛИ»*, обозначаемая двумя символами **||**, работает следующим образом:

```
if (key == 'n' || key == 'N') {
// Переход к следующему фото
}
```

ПРИМЕЧАНИЕ

Чтобы напечатать знак |, нажмите сочетание клавиш Shift+\. Клавиша, печатающая обратный слеш и символ |, обычно расположена над клавишей Enter.

При использовании операции логического «ИЛИ», для выполнения кода JavaScript в скобках, необходимо, чтобы было верно хотя бы одно из условий.

Как и в случае с операцией логического «И», вы можете сравнивать больше двух условий. Например, вы написали на языке JavaScript игру в гонки. Игрок располагает ограниченным количеством времени, некоторым объемом бензина и заданным числом машин (в процессе игры он может разбить машину). Чтобы сделать игру более занимательной, вы можете запрограммировать конец игры, если заканчивается хотя бы один из этих ресурсов:

```
if (gas <= 0 || time <= 0 || cars <= 0) {
//игра окончена
}
```

При проверке множественных условий часто бывает сложно понять логику управляющих инструкций. Некоторые программисты группируют каждый набор условий в паре скобок, чтобы сделать логику более легкой для понимания:

```
if ((key == 'n') || (key == 'N')) {
//Переход к следующему фото
}
```

Чтобы прочитать этот код, просто считайте каждую группу отдельным тестом. Результат операции в скобках всегда должен оказаться либо истинным, либо ложным.

Отрицание условия

Если вы когда-либо увлекались Суперменом, то, возможно, помните Бизарро — антигероя, который жил на кубической планете Ялмез (Земля наоборот), имел костюм с перевернутой буквой S и был противоположностью Супермена во всех смыслах. Если Бизарро говорил «Да», это на самом деле означало «Нет», и наоборот.

В программировании на языке JavaScript есть аналогичный герой, называемый *операцией логического «НЕ»*, которая обозначается восклицательным знаком (`!`). Вы уже видели, как операция логического «НЕ» использовалась вместе со знаком равенства (не равно: `!=`). Но операция логического «НЕ» может использоваться и сама по себе, чтобы изменить результат управляющей инструкции на противоположный, другими словами, с его помощью можно превратить истинное выражение в ложное, а ложное в истинное.

Операцию логического «НЕ» следует использовать, если вы хотите выполнить какой-либо код на основании отрицательного условия. Например, вы создали переменную с именем `valid`, содержащую логическое значение `true` или `false` (см. врезку в начале данной главы). Вы используете эту переменную, чтобы проверить, правильно ли посетитель заполнил форму (например, поле не может быть пустым, в нем должен быть адрес электронной почты). Если возникает проблема (допустим, поле пустое), вы можете изменить значение переменной `valid` на «ложь»: `valid = false`.

Теперь, если вы хотите вывести на экран сообщение об ошибке и предотвратить отправку формы, то вы можете написать следующую управляющую инструкцию:

```
if (! valid) {
// Вывести на экран сообщение об ошибке и
//не отсылать форму
}
```

Условие `! valid` можно перевести как «если не действительно», что означает: если значение переменной `valid` ложно, то *условие ис-*

тинно. Чтобы выяснить результат условия, использующего операцию логического «НЕ», просто оцените условие без операции «НЕ», а потом измените его на противоположное. Другими словами, если результатом условия является `true`, то операция `!` изменяет его на значение `false`, поэтому управляющая инструкция не выполняется.

Как видите, понять операцию логического «НЕ» очень просто (переводя с языка Бизарро: очень сложно, но, попрактиковавшись в его использовании достаточно долго, вы привыкнете).

Вложение управляющих инструкций

По большому счету, программирование базируется на принятии решений с учетом информации, введенной пользователем, или на текущих условиях программы. Чем больше решений принимает программа, тем больше возможных результатов и тем «умнее» эта программа кажется. Вы можете обнаружить, что должны принимать дальнейшие решения *после* выполнения одной управляющей инструкции.

Допустим, в примере «Что делать в пятницу вечером?» вы хотите расширить программу до каждого из вечеров недели. В данном случае вы сначала должны определить текущий день недели, а затем решить, что делать в этот день. Итак, у вас может быть управляющая инструкция, в которой выясняется, пятница ли сегодня, и если она выполняется, у вас появляется серия других управляющих инструкций, с помощью которых определяется, что делать в этот день:

```
if (dayOfWeek == 'Пятница') {
var fridayCash = prompt('Сколько денег вы можете ↵
потратить?', '');
if (fridayCash >= 1500) {
alert('Вы можете и поужинать, и пойти в кино.');
} else if (fridayCash >= 1000) {
alert('Вы можете поужинать.');
} else if (fridayCash >= 500) {
alert('Вы можете пойти в кино.');
} else {
alert('Похоже, вы будете смотреть телевизор.');
}
}
```

В данном примере первое условие запрашивает, является ли значение, сохраненное в переменной `dayOfWeek`, строкой `'Пятница'`. Если ответ утвердительный, то появляется диалоговое окно, запрашивающее у посетителя некоторую информацию, и запускается следующая управляющая инструкция. Другими словами, первое условие (`dayOfWeek == 'Пятница'`) является переходом на другую серию управляющих инструкций. Однако если значение переменной `dayOfWeek` — это не `'Пятница'`, то условие является ложным, и вложенные управляющие инструкции пропускаются.

Советы по конструированию управляющих инструкций

Пример вложенной управляющей инструкции из предыдущего раздела может показаться немного страшным. Так много `()`, `{}`, `if` и `else`. А если вы вдруг допустите опечатку в одном из важных фрагментов управляющей инструкции, ваш сценарий не будет работать. Есть некоторые приемы, которые пригодятся вам при верстке кода JavaScript, содержащего управляющие инструкции.

- **Вводите обе фигурные скобки до набора кода в них.** Одна из наиболее распространенных ошибок программистов — пропуск закрывающей скобки в управляющей инструкции. Чтобы избежать этой ошибки, набирайте сначала условие и скобки, а затем код JavaScript, выполняемый, если условие верно. Например, начните написание управляющей инструкции так:

```
if (dayOfWeek == 'Пятница') {


}
```

Другими словами, напечатайте сначала условие `if` и первую скобку, дважды нажмите клавишу **Enter**, а затем поставьте закрывающую скобку. Теперь, когда основной синтаксис верен, вы можете установить текстовый курсор на пустую строку между скобками и добавить код JavaScript.

- **Делайте отступ для строк кода в скобках.** Так повышается наглядность структуры управляющей инструкции:

```
if (a < 10 && a > 1) {
alert("Значение " + a + " принадлежит промежутку ↵
  от 1 до 10");
}
```

Используя несколько пробелов (или клавишу **Tab**) для создания отступа строк кода в скобках, становится проще определить, какой фрагмент программы должен выполняться как часть управляющей инструкции. Если у вас есть вложенные управляющие инструкции, отделяйте отступом каждую из них:

```
if (a < 10 && a > 1) {
// Отступ первого уровня для первой
//управляющей инструкции
alert("Значение " + a + " принадлежит промежутку ↵
от 1 до 10");
if (a==5) {
// Отступ второго уровня для второй
//управляющей инструкции
alert(a + " половина от десяти.");
}
}
```

- **Используйте операцию == для определения равенства значений.** При проверке равенства значений не забудьте использовать операцию равенства:

```
if (name == 'Вова') {
```

Распространенная ошибка — использование одного знака равенства:

```
if (name = 'Вова') {
```

Одинарный знак равенства присваивает значение переменной, то есть в данном случае строка `'Вова'` будет сохранена в переменной `name`. Интерпретатор JavaScript считает этот этап истинным, поэтому код, следующий за условием, всегда будет выполняться.

Использование управляющих инструкций на практике

Управляющие инструкции станут частью вашего ежедневно используемого инструментария при работе с языком JavaScript. В данном руководстве вы на практике используете управляющие инструкции для контроля над работой сценария.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в главе 1 за подробной информацией по копированию файлов примеров.

1. **В текстовом редакторе откройте файл *03_01.html* из каталога *глава03*.**

Начнем с запроса номера у посетителя. В этом файле уже содержатся элементы `script` в разделе заголовка и в теле документа.

2. **Между первой парой тегов элемента `script` в разделе заголовка страницы введите выделенный полужирным шрифтом код:**

```
<script>
var luckyNumber = prompt('Какое ваше счастливое ↵
число?','');
</script>
```

Эта строка кода открывает диалоговое окно JavaScript, задает вопрос и сохраняет то, что ввел пользователь, в переменной `luckyNumber`. Далее вы добавите управляющую инструкцию, чтобы проверить, что пользователь напечатал в диалоговом окне запроса.

3. **Найдите вторую пару тегов элемента `script` ниже, в теле страницы, и введите выделенный полужирным шрифтом код:**

```
<script>
if (luckyNumber == 7 ) {
</script>
```

Не забудьте двойной знак «равно» (==), который сравнивает два значения. Этот фрагмент кода является началом управляющей инструкции; он просто проверяет, ввел ли посетитель число 7.

4. **Дважды нажмите клавишу Enter и поставьте закрывающую скобку, чтобы код выглядел так:**

```
<script>
if (luckyNumber == 7 ) {
}
</script>
```

Закрывающая скобка завершает управляющую инструкцию. Любой код JavaScript, который вы добавите в скобках, будет выполнен только в том случае, если условие истинно.

ПРИМЕЧАНИЕ

Как уже было указано, следует сначала поставить закрывающую скобку, а затем вводить код, являющийся частью управляющей инструкции.

5. **Щелкните мышью в пустой строке над закрывающей скобкой. Дважды нажмите клавишу Пробел и напечатайте:**

```
document.write("<p>Эй, 7 — это и мое счастливое ↵
число! </p>");
```

Два пробела перед кодом образуют отступ, поэтому вы можете легко понять, что этот код — часть управляющей инструкции. Код JavaScript должен быть вам знакомым — он просто выводит сообщение на страницу.

6. **Сохраните файл и просмотрите его в браузере. Введите число 7 в появившееся диалоговое окно.**

При загрузке страницы вы должны увидеть сообщение: «Эй, 7 — это и мое счастливое число!» под заголовком. Если вы его не видите, просмотрите код и убедитесь, что все напечатали правильно (за советами по обращению с неработающим сценарием обратитесь к разделу «Отслеживание ошибок» главы 1). Перезагрузите страницу, но введите другое число. На этот раз под заголовком не появляется ничего. Добавьте альтернативное условие, чтобы напечатать другое сообщение.

7. **Вернитесь в текстовый редактор и добавьте на страницу код, выделенный полужирным шрифтом:**

```
<script>
if (luckyNumber == 7 ) {
document.write("<p>Эй, 7 — это и мое счастливое ↵
число!</p>");
} else {
document.write("<p>Число " + luckyNumber + ↵
"является для вас счастливым!</p>");
}
</script>
```

Альтернативное условие обеспечивает запасное сообщение, поэтому, если посетитель напечатал не 7, он увидит иное сообщение, в котором записано его счастливое число. Чтобы завершить это упражнение, добавьте инструкцию `else if` для тестирования других значений и отображения других сообщений.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Два набора тегов элемента `script`

Зачем нужны два набора тегов элемента `script` в разделах заголовка и тела страницы?

При использовании метода `document.write()` для добавления на страницу контента вам необходимо поместить код `document.write()` в то место на странице, где должно появиться сообщение, в данном случае — в теле документа под элементом `h1`. Первая пара тегов элемента `script` находится в разделе заголовка документа, так как окно запроса должно появиться раньше. Если вы переместите метод `prompt()` в тело документа (кстати, попробуйте), вы увидите, что когда появляется окно запроса, отображена только часть страницы. Поскольку код JavaScript выполняется до того, как отобразятся другие части страницы, браузеру приходится ждать, пока посетитель введет данные в окно запроса, прежде чем он сможет отобразить оставшуюся часть страницы. Другими словами, страница выглядит странно.

Однако, помещая метод `prompt()` в раздел заголовка, при появлении окна запроса страница пуста — так лучше. В следующей главе вы узнаете, как добавить контент в любое место на странице без использования метода `document.write()`. Как только вы познакомитесь с этой техникой, вы сможете помещать весь свой JavaScript-код в одном месте на странице.

8. **Добавьте в ваш сценарий две строки, выделенные полужирным шрифтом:**

```
<script>

if (luckyNumber == 7 ) {

document.write("<p>Эй, 7 — это и мое счастливое ↵
число!</p>");

} else if (luckyNumber == 13 || luckyNumber == 24) {

document.write("<p>Ой. " + luckyNumber + "? ↵
Это — несчастливое число!");
```

```
} else {
document.write("<p>Число " + luckyNumber + " ↵
является для вас счастливым!</p>");
}
</script>
```

Сценарий сначала проверяет, сохранено ли в переменной `luckyNumber` число 7. Если в ней сохранено какое-то другое значение, выполняется код `else if`. Это управляющая инструкция состоит из двух условий: `luckyNumber == 13` и `luckyNumber == 24.` `||` (операция логического «ИЛИ») позволяет всей инструкции становиться истинным, если хотя бы одно из условий истинно. То есть если посетитель вводит числа 13 или 24, на странице отображается сообщение «Это — несчастливое число».

ПРИМЕЧАНИЕ

Вы можете ввести операцию логического «ИЛИ», дважды нажав сочетание клавиш **Shift+**\.

Просмотрите страницу в браузере и введите число 13 в диалоговое окно запроса. Обновите страницу и поэкспериментируйте с другими числами, а также буквами и другими знаками. Вы заметите что, если вводится слово или другой нечисловой символ, выполняется конечное альтернативное условие и появляется сообщение: «Число абв является для вас счастливым!». Поскольку в этом нет смысла, вам следует запрограммировать появление другого окна запроса, если посетитель не введет число.

9. Вернитесь в текстовый редактор, найдите первую пару тегов элемента `script` в разделе заголовка и добавьте код, выделенный полужирным шрифтом:

```
<script>
var luckyNumber = prompt('Какое ваше счастливое ↵
число? ','');
luckyNumber = parseInt(luckyNumber, 10);
</script>
```

Данная строка кода обрабатывает значение переменной `luckyNumber` с помощью функции `parseInt()`. Эта команда

JavaScript берет значение и пытается превратить его в целое число, например, –20, 0, 1, 5 или 100. Вы прочитаете об этой команде подробнее в главе 16. А сейчас просто представьте, что посетитель напечатал текст, например, «ха-ха». Команда `parseInt()` не сможет превратить этот текст в число, а вместо этого выдаст специальное значение JavaScript — *NaN*, означающее «не число». Вы можете использовать эту информацию для вызова нового диалогового окна, если в первое не было введено число.

10. Добавьте в ваш сценарий, выделенный полужирным шрифтом , код:

```
<script>
var luckyNumber = prompt('Какое ваше счастливое ↵
число? ','');
luckyNumber = parseInt(luckyNumber, 10);
if (isNaN(luckyNumber)) {
luckyNumber = prompt('Пожалуйста, сообщите мне ↵
свое счастливое число.','');
}
</script>
```

Здесь мы вновь работаем с управляющей инструкцией. Условие `isNaN(luckyNumber)` использует другую команду JavaScript, которая проверяет, является ли введенный символ числом. В частности, она проверяет, является ли значение переменной `luckyNumber` «не числом». Если это значение не является числом (например, посетитель вводит «абвгд»), появляется второе диалоговое окно и вновь задается вопрос. Если посетитель ввел число, код, инициирующий открытие второго диалогового окна, пропускается.

Сохраните страницу и вновь просмотрите ее в браузере. На этот раз напечатайте слово и нажмите кнопку **OK**, когда появится диалоговое окно с запросом. Вы должны увидеть второе окно с запросом. На этот раз введите число. Конечно же, сценарий считает, что посетитель в первый раз просто ошибся, напечатав не число, но он не повторит такую ошибку. К сожалению, если посетитель и во второй раз напечатает слово, вы столкнетесь с той же проблемой. О том, как ее исправить, вы узнаете в следующем разделе.

ПРИМЕЧАНИЕ

Полную версию этого примера вы найдете в файле *готовый_03_01.html* (папка *глава03*).

Работа с повторяющимися задачами с использованием циклов

Иногда необходимо снова и снова повторять одну и ту же серию шагов. Например, у вас есть веб-форма с 30 текстовыми полями. Если пользователь отправляет такую форму, вы хотите убедиться, что все поля заполнены. Другими словами, вы должны 30 раз выполнить один и тот же набор действий — проверить, заполнено ли каждое поле. Поскольку компьютеры хорошо выполняют повторяющиеся задачи, в язык JavaScript также включен инструмент, позволяющий быстро повторять один и тот же набор действий.

На языке программистов многократное выполнение одной и той же задачи называется *циклом*. Поскольку циклы очень часто встречаются в программировании, язык JavaScript предлагает несколько типов. Все они делают одно и то же, но немного разными способами.

Циклы `while`

Цикл `while` повторяет фрагмент кода, пока остается истинным определенное условие. Базовая структура цикла `while` такова:

```
while (условие) {
// повторяющийся код javascript
}
```

Первая строка вводит инструкцию `while`. Как и в случае с управляющей инструкцией, вы помещаете условие в скобки, следующие за ключевым словом `while`. Условие — это любой тест, который вы можете встретить в управляющей инструкции, например, `x > 10` или `answer == 'да'`. И как в управляющей инструкции, интерпретатор JavaScript выполняет весь код, находящийся в фигурных скобках, если условие истинно.

Однако в отличие от управляющей инструкции, когда интерпретатор JavaScript достигает закрывающей скобки инструкции `while`, он вместо перехода к следующей строке программы возвращается к началу цикла `while` и тестирует условие повторно. Если условие вновь оказывается верным, интерпретатор опять выполняет код JavaScript. Процесс продолжается до тех пор, пока условие не станет ложным; после этого программа переходит к выполнению инструкции, следующей за циклом (рис. 3.5).

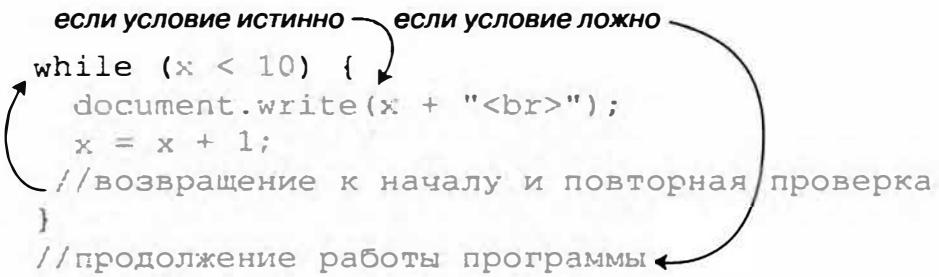

Рис. 3.5. Цикл `while` запускает код JavaScript в фигурных скобках, пока условие (в данном случае — `x < 10`) истинно

Допустим, вы хотите отобразить на странице числа от 1 до 5. Рассмотрим один из способов сделать это:

```
document.write('Число 1 <br>');
document.write('Число 2 <br>');
document.write('Число 3 <br>');
document.write('Число 4 <br>');
document.write('Число 5 <br>');
```

Обратите внимание, что строки кода почти идентичны: от строки к строке изменяется только число. В данной ситуации для достижения той же цели более эффективно использовать цикл:

```
var num = 1;
while (num <= 5) {
document.write('Число ' + num + '<br>');
num += 1;
}
```

Первая строка кода, `var num = 1;`, не является частью цикла `while`, она устанавливает переменную для хранения числа, отображаемого на странице. Вторая строка — это начало цикла. Она задает условие проверки. Пока число, сохраненное в переменной `num`, меньше или равно `5`, код в скобках выполняется.

Когда условие проверяется в первый раз, значение переменной `num` равно `1`, условие истинно (так как 1 меньше 5), поэтому выполняется команда `document.write()` — на странице отображается строка `'Число 1'` (`<br>` — это HTML-тег, означающий переход на новую строку, он предназначен для того, чтобы каждая строка кода отображалась на вебстранице на отдельной строке).

ПРИМЕЧАНИЕ

Выражение `num += 1` можно записать более компактно (оно означает, что к текущему значению переменной `num` добавляется единица):

```
num ++
```

Данный метод также добавляет 1 к значению переменной `num` (см. табл. 2.3 в разделе «Изменение значений в переменных» главы 2 для получения более полной информации).

Последняя строка цикла `num += 1` очень важна. Она не просто увеличивает значение переменной `num` на 1, чтобы на странице отображалось следующее число (например, 2), но также позволяет условию проверки стать ложным (если вам кажется странным сочетание символов +=, обратитесь к разделу «Изменение значений в переменных» главы 2). Поскольку код JavaScript в инструкции `while` повторяется до тех пор, пока условие истинно, вы должны изменить один из элементов условия так, чтобы условие стало ложным, цикл прекратился, и можно было перейти к следующей части сценария. Если условие теста никогда не становится ложным, имеет место так называемый *бесконечный цикл* — программа, выполнение которой никогда не закончится. Обратите внимание, что произойдет, если вы не включите эту строку в цикл:

```
var num = 1;
while (num <= 5) { // Это бесконечный цикл
document.write('Число ' + num + '<br>');
}
```

После первого выполнения цикла тест спросит: «1 меньше или равен 5?». Ответ утвердительный, поэтому выполняется команда `document.write()`. В конце цикла (последние скобки) интерпретатор JavaScript возвращается к началу цикла и вновь тестирует условие. Здесь значение переменной `num` по-прежнему равно 1, условие истинно — и выполняется команда `document.write()`. Интерпретатор JavaScript вновь возвращается к началу цикла и тестирует условие в третий раз. Вы уже догадались, к чему это приведет: к бесконечному числу строк `'Число 1'`.

Этот простой пример также демонстрирует гибкость циклов. Например, вы хотели записать числа не от 1 до 5, а от 1 до 100. Вместо добавления множества дополнительных строк с командой `document.write()` вы просто изменяете условие теста следующим образом:

```
var num = 1;
while (num <= 100) {
document.write('Число ' + num + '<br>');
num = num + 1;
}
```

Теперь цикл будет выполнен 100 раз и на веб-странице появится 100 строк.

Циклы и массивы

Вы поймете, что циклы удобны, работая с обычным элементом JavaScript — *массивом.* Как вы помните из раздела «Массивы» главы 2, массив — это набор данных. Вы можете представить себе массив как список планируемых покупок. В магазине вы совершаете нечто похожее на цикл: вы проходите по рядам и ищете товар, который присутствует в вашем списке. Находите его и кладете в тележку. Затем ищете следующий предмет в списке, находите его, кладете в тележку и т. д., пока не найдете все товары из списка. Тогда вы выходите из торгового зала (то же самое, что выйти из цикла) и идете к кассе (другими словами, переходите к следующему фрагменту программы).

В языке JavaScript разрешается использовать циклы для прохождения через элементы массива и выполнения действий над каждым из них. Например, вы пишете программу для составления календаря. Календарь сгенерирован, и вы хотите отобразить в нем названия дней недели. Вы можете начать с сохранения названий дней недели в массиве следующим образом:

```
var days = ['Понедельник', 'Вторник', 'Среда', ↵
'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];
```

Затем, используя цикл, вы можете пройти через каждый элемент массива и напечатать названия дней на странице. Помните, что вы получаете доступ к элементу массива, используя значение индекса, соответствующее этому элементу. Например, первый элемент массива `days` (`Понедельник`), вызывается с помощью кода `days[0]`. Второй элемент — `days[1]` и т. д.

Чтобы напечатать каждый из элементов этого массива, вы можете использовать цикл `while` следующим образом:

```
var counter = 0;
while (counter < days.length) {
document.write(days[counter] + ', ');
counter++;
}
```

Первая строка, `var counter = 0`, устанавливает (или, на языке программистов, *инициализирует*) переменную `counter`, используемую одновременно как часть тестового условия и как индекс для доступа к элементам массива. Условие `counter < days.length` просто проверяет, меньше ли значение переменной `counter` количества элементов массива (из раздела «Доступ к элементам массива» главы 2 вы узнали, что количество элементов массива хранится в свойстве `length`). В данном случае в условии проверяется, является ли значение переменной `counter` меньше, чем 7 (число дней в неделе). Если значение переменной меньше 7, начинается цикл: день недели отображается на странице (за ним следуют запятая и пробел), а значение переменной `counter` увеличивается на 1 (выражение `counter++` — это то же, что и `counter += 1`, или `counter = counter + 1` (см. предыдущее примечание)). После выполнения цикла происходит новая проверка. Цикл повторяется, пока условие не оценивается как «ложь» (рис. 3.6).

значение переменной `counter` до теста	условие	цикл	`days[counter]`	значение `counter` после `counter ++`
0	0 < 7	да	days[0]	1
1	1 < 7	да	days[1]	2
2	2 < 7	да	days[2]	3
3	3 < 7	да	days[3]	4
4	4 < 7	да	days[4]	5
5	5 < 7	да	days[5]	6
6	6 < 7	да	days[6]	7
7	7 < 7	нет		

Рис. 3.6. В данном цикле условие проверялось 8 раз. В последнем тесте спрашивается, является ли число 7 меньшим 7. Получается значение «ложь», поэтому цикл завершается, и интерпретатор JavaScript пропускает его и переходит к следующей части сценария. Окончательный результат этого сценария: «Понедельник, Вторник, Среда, Четверг, Пятница, Суббота, Воскресенье,». Обратите внимание на запятую после слова «Воскресенье». Чтобы предотвратить появление этой запятой, вы можете использовать метод `join()`. Эта более сложная техника описана в разделе «Выводы» главы 16

Циклы for

Язык JavaScript предлагает еще один тип цикла — for. Он немного компактнее (и чуть сложнее). Циклы for обычно используются для повторения серии шагов определенное количество раз. Поэтому они часто включают особую переменную цикла, условие и способ изменения переменной цикла.

Во многих случаях цикл for помогает достичь тех же целей, что и цикл while, но с использованием меньшего количества строк кода. Вот, например, цикл while:

```
var num = 1;
while (num <= 100) {
document.write('Число ' + num + ' <br>');
num += 1;
}
```

Вы можете достичь такого же результата с помощью всего трех строк кода:

```
for (var num = 1; num <= 100; num++) {
document.write('Число ' + num + ' <br>');
}
```

На первый взгляд циклы for могут показаться несколько непонятными, но, когда вы вычлените части инструкции for, они окажутся несложными. Каждый цикл for начинается с ключевого слова for, за которым следуют круглые скобки (их содержимое подразделяется на три части), а также фигурные скобки.

Как и в случае с циклом while, информация в фигурных скобках (document.write('Число ' + num + '
')) в данном примере) — это код JavaScript, который выполняется в качестве части цикла.

В табл. 3.2 объясняются три части цикла из круглых скобок. В первой части (var num = 1;) инициализируется переменная цикла. Это происходит только один раз в самом начале работы цикла. Вторая часть — это условие, предназначенное для проверки того, нужно ли в очередной раз выполнять код. Третья часть — это действие, происходящее в конце каждого цикла (обычно это изменение значения переменной цикла) до тех пор, пока условие теста не станет ложным и цикл не закончится.

Табл. 3.2. Части цикла `for`

Часть цикла	Что означает	Когда применяется
`for`	Начинает цикл `for`	
`var num = 1;`	Устанавливает значение переменной `num` равное `1`	Только один раз — в самом начале инструкции
`num <= 100;`	Значение `num` меньше или равно `100`? Если это так, то цикл повторяется. Если нет, то цикл пропускается и выполняется следующая часть сценария	В начале инструкции и перед каждым повторным выполнением цикла
`num++`	Добавляет `1` к переменной `num`. То же, что `num = num + 1` и `num+=1`.	В конце каждого прохождения цикла

Поскольку циклы `for` являются простым способом повторять серию шагов определенное количество раз, они действительно хороши для работы с элементами массива.

Цикл `while` на рис. 3.5, записывающий каждый из элементов массива на страницу, может быть переписан с использование цикла `for` следующим образом:

```
var days = ['Понедельник', 'Вторник', 'Среда', ↵
'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];
for (var i=0; i< days.length; i++) {
document.write(days[i] + ', ');
}
```

СОВЕТ

Опытные программисты часто используют очень короткое имя для переменной в цикле `for`. В коде, приведенном выше, таким именем является буква `i`. Однобуквенное имя (обычно i, j и z) быстро печатается; а поскольку эта переменная не используется за пределами цикла, нет необходимости присваивать ей описательное имя, например, `counter`.

В приведенных примерах значение переменной изменялось до определенного числа, а затем цикл останавливался, но возможен и обратный процесс. Например, вы хотели бы отобразить элементы массива в обратном порядке (иначе говоря, последний элемент массива отображается первым). Вы можете сделать это так:

```
var example = ['первый','второй','третий','последний'];
for (var j = example.length - 1 ; j >= 0; j--) {
document.write(example[j] + '<br>');
}
```

В данном примере начальное значение переменной цикла `j` — общее количество единиц в массиве минус 1 (4–1=3). (Почему минус 1? Дело в том, что индекс элемента массива всегда на 1 меньше его порядкового номера: индекс первого элемента — `0`, индекс второго — `1`, а индекс последнего — значение свойства массива `length` минус 1. Другими словами, для получения доступа к последнему элементу массива необходимо использовать код `example[3]`.)

После каждого прохождения цикла вы проверяете, является ли значение переменной `j` большим 0. Если это так, то выполняется код в фигурных скобках. Затем из значения переменной `j` вычитается 1 (`j--`), и проверка проводится снова.

Таким образом, данный цикл выполняется в обратном направлении, начиная с конца массива (с элемента с индексом 3) к началу массива (к элементу с индексом 0).

Циклы do/while

Есть еще один, менее распространенный тип циклов, известный как `do/while`. Они работают практически так же, как циклы `while`. Базовая структура выглядит так:

```
do {
// повторяющийся код javascript
} while (условие) ;
```

В циклах данного типа проверка условия происходит *в конце*, после того как цикл выполнен. В результате выполнение кода JavaScript, находящегося в фигурных скобках, происходит *хотя бы один раз*. Даже если условие никогда не бывает истинным, проверка условия происходит только после выполнения кода.

Случаев, когда такой цикл может пригодиться, не много. Однако он полезен, когда вы хотите получить информацию от пользователя. Руководство, выполненное вами выше в этой главе, — хороший тому пример.

Сценарий просит посетителя ввести число. Если кто-то не введет число, сценарий еще раз об этом попросит. К сожалению, если посетитель не введет число и во второй раз, на страницу выводится бессмысленное сообщение.

Однако используя цикл `do/while`, вы можете напоминать посетителю о необходимости ввести число, пока он, наконец, этого не сделает. Чтобы посмотреть, как работает подобный код, отредактируйте пример из предыдущего руководства:

1. **В текстовом редакторе откройте файл *03_01.html*, с которым вы работали ранее в этой главе:**

   Если вы не выполнили прошлое задание, откройте файл *готовый_03_01.html*. Вам предстоит заменить код в верхней части страницы циклом `do/while`.

2. **Найдите фрагмент кода между открывающим и закрывающим тегами элемента `script` в разделе заголовка страницы и удалите код, выделенный полужирным шрифтом:**

```
var luckyNumber = prompt('Какое ваше счастливое ↵
число?','');

luckyNumber = parseInt(luckyNumber, 10);

if (isNaN(luckyNumber)) {

luckyNumber = prompt('Пожалуйста, сообщите мне ↵
свое счастливое число.','');

}
```

   Удаленный код вызывал второе диалоговое окно с запросом. Оно вам больше не понадобится. Вместо этого вставьте код цикла `do/while`.

3. **Поместите текстовый курсор перед первой строкой кода (которая начинается с `var luckyNumber`) и напечатайте следующее:**

```
do {
```

   Этот код создает начало цикла. Далее завершим цикл и добавим тестовое условие.

4. **Щелкните мышью в конце последней строки кода JavaScript в разделе и напечатайте: `} while (isNaN(luckyNumber));`. Законченный блок кода должен выглядеть так:**

```
do {
var luckyNumber = prompt('Какое ваше счастливое ↵
число?','');
luckyNumber = parseInt(luckyNumber, 10);
} while (isNaN(luckyNumber));
```

Сохраните этот файл и просмотрите его в браузере. Напечатайте в диалоговом окне текст или другие нечисловые символы. Надоедливое диалоговое окно будет появляться до тех пор, пока вы, наконец, не введете число.

Вот как это работает: ключевое слово `do` сообщает интерпретатору JavaScript, что начинается цикл `do/while`. Затем выполняются следующие две строки кода, поэтому появляется окно с запросом и ответ посетителя превращается в целое число. Только теперь проверяется условие. Это условие не отличается от того, которое присутствует в предыдущем сценарии: просто выполняется проверка того, не является ли значение, введенное посетителем, «нечислом». Если введено «нечисло», цикл повторяется. Другими словами, запрос станет появляться снова и снова до тех пор, пока не будет введено число. Здесь особенно хорошо то, что цикл может закончиться и после первого раза, то есть если в ответ на вопрос посетитель введет число, цикла не будет.

Рабочий сценарий сохранен в файле *готовый_do-while* в *папке03*.

Функции: многократное использование кода

Представьте, что у вас на работе появился новый ассистент, который готов помочь вам в любом деле (уже можно отнести эту книгу к фантастике). Например, вы проголодались и захотели кусочек пиццы, но, поскольку помощник еще не знаком с планом здания, вы должны дать ему подробные инструкции: «Выйди из кабинета, поверни направо к лифту...». В следующий раз ассистент уже будет знать, где готовят пиццу, поэтому вы просто говорите: «Принеси мне кусочек пиццы», он идет в пиццерию и приносит пиццу. Другими словами, вам нужно дать подробные инструкции только *один раз*. Ваш ассистент запомнит необходимые шаги. И если вы скажете: «Принеси мне кусочек», он мгновенно исчезнет, а через некоторое время появится с пиццей. В языке JavaScript есть подобный механизм, называемый функцией. *Функция* — это серия

шагов программирования, которые вы задаете в начале сценария — то же, что и подробные инструкции для ассистента. Эти шаги не выполняются при создании функции, вместо этого они сохраняются в памяти браузера, и вы вызываете их, когда это необходимо.

Функции бесценны для выполнения нескольких шагов программы с определенной периодичностью. Например, вы создали на веб-странице фотогалерею, состоящую из 50 эскизов фотографий. Если кто-то щелкнет мышью по маленькой фотографии, вам нужно, чтобы страница потускнела, отобразился заголовок, и экран заполнила большая версия изображения. Каждый раз, когда кто-нибудь щелкнет кнопкой мыши по картинке, процесс повторится. Таким образом, на странице, содержащей 50 маленьких фотографий, ваш сценарий выполнит одну и ту же серию шагов 50 раз. К счастью, вам не придется писать один и тот же код 50 раз, чтобы ваша фотогалерея работала. Вместо этого вы можете записать функцию со всеми необходимыми шагами, а затем эта функция будет выполняться при каждом щелчке по эскизу. Вы пишете код только один раз, но выполняете его, когда хотите.

Основная структура функции выглядит так:

```
function имяфункции() {

// код JavaScript, который необходимо выполнить

}
```

Ключевое слово `function` дает интерпретатору JavaScript знать, что вы создаете функцию, это похоже на то, как если бы вы использовали слово `if`, чтобы начать инструкцию `if/else`, или `var`, чтобы создать переменную. Далее вы указываете имя функции; как и в случае с переменной, вы вольны выбрать любое имя. Следуйте тем же правилам, что и при наименовании переменных (см. раздел «Создание переменной» главы 2). Кроме того, обычной практикой считается включение в имя функции глагола, например, `calculateTax`, `getScreenHeight`, `updatePage` или `fadeImage`. Глагол дает ясное представление о том, что делает функция, и помогает отличить функцию от переменной.

После имени добавьте пару скобок, являющихся другой характеристикой функции. После скобок — пробел, за которым следует фигурная скобка, одна или более строк кода JavaScript и, наконец, закрывающая фигурная скобка. Как и в случае с инструкциями `if`, фигурные скобки определяют начало и конец фрагмента кода функции JavaScript.

ПРИМЕЧАНИЕ

Как и в случае с инструкциями `if/else`, код функции удобнее читать, если вы создадите отступы для строк кода JavaScript, расположенного в фигурных скобках. Часто для этого используются два пробела (или табуляция).

Рассмотрим очень простую функцию, выводящую на экран дату в формате «День_недели Месяц Число Год»:

```
function printToday() {

var today = new Date();

document.write(today.toDateString());

}
```

Имя функции — `printToday`. Она содержит всего две строки кода JavaScript, возвращающих текущую дату, затем преобразующих ее в понятный нам формат (часть `toDateString()`), после чего выводящих на страницу результат с применением нашего старого друга — команды `document.write()`. Пока не задумывайтесь о том, как работает весь этот код с датами, об этом мы поговорим в разделе «Дата и время» главы 16.

Функции, которые будут использованы в ходе выполнения сценария, программисты обычно помещают в начало программы. Помните, что функция не запускается сразу после создания, — это все равно, что рассказать ассистенту, где находится пицца, но не послать его за ней. Код JavaScript просто сохранен в памяти браузера, ожидая, что позднее, когда он понадобится, его запустят.

Но как запустить функцию? Говоря языком программистов, вы *вызываете* функцию, когда желаете, чтобы она выполнила свою задачу. Вызвать функцию — значит записать ее имя, за которым следует пара скобок. Например, вы запускаете функцию `printToday`, набрав следующее:

```
printToday();
```

Как видно, для запуска функции не нужно много печатать. Будучи однажды созданной, она не требует дополнительного кода, чтобы выдать результат.

ПРИМЕЧАНИЕ

Вызывая функцию, не забудьте поставить в конце скобки, которые ее запускают. Например, запись `printToday` ни к чему не приведет, а запись `printToday()` выполнит функцию.

Практика

Функции очень важны. Приведенная здесь инструкция поможет вам на практике научиться создавать функции шаг за шагом.

1. **В текстовом редакторе откройте файл *03_02.html*.**

   Начните с добавления функции в раздел заголовка документа.

2. **Между открывающим и закрывающим тегами элемента `script` в разделе заголовка страницы введите следующий код:**

```
function printToday() {
var today = new Date();
document.write(today.toDateString());
}
```

   Функция на месте, но она пока не выполняет никаких действий.

3. **Сохраните файл и просмотрите его в браузере.**

   Ничего не происходит. И все же кое-что происходит, просто вы этого не видите. Браузер считывает инструкции функции, запоминает их и ждет, когда вы вызовете функцию, что вы сейчас и сделаете.

4. **Вернитесь к текстовому редактору и файлу *03_02.html*. Найдите элемент `p` в начале строки «Сегодня », и между открывающим и закрывающим тегами элемента `strong` добавьте код, выделенный полужирным шрифтом:**

```
<p>Сегодня <strong>
<script>
printToday();
</script>
</strong></p>
```

Сохраните страницу и просмотрите ее в браузере (рис. 3.7). Вы увидите текущую дату. Если вы хотите отобразить дату в нижней части вебстраницы, все, что вам нужно, — еще раз вызвать функцию.

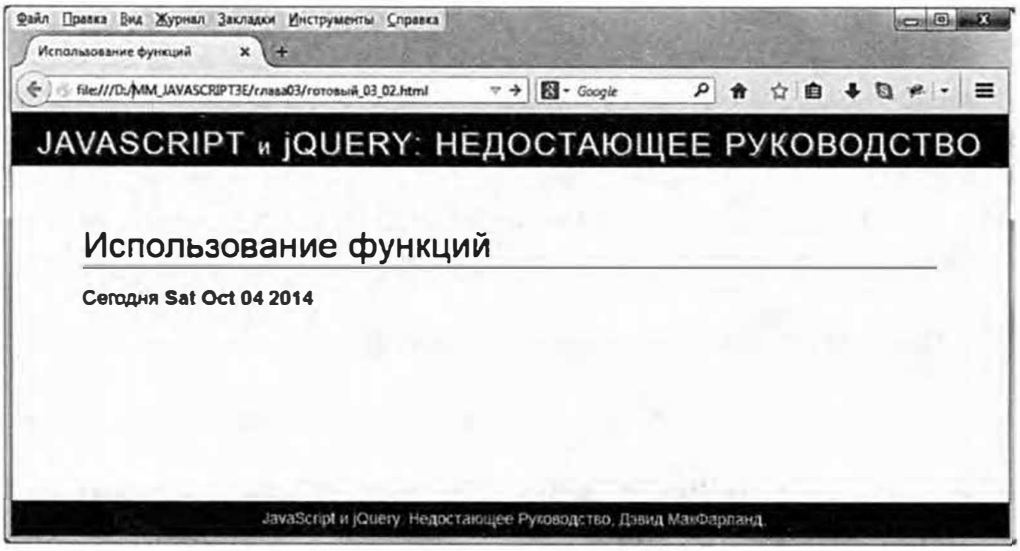

Рис. 3.7. Результат работы вашей первой функции. Дата, которую вы увидите на своей странице, будет отличаться от той, которая изображена на рисунке. Это связано с динамической природой функций — они выполняются тогда, когда вы их вызываете, поэтому при каждом вызове функции `printToday` будет отображаться другая дата

Передача данных функциям

Функции еще более полезны, если вы сообщаете им необходимую информацию. Вернемся к вашему ассистенту — парню, приносящему пиццу. Первичная «функция», описанная ранее, заключалась в том, чтобы отправиться в пиццерию, купить порцию и вернуться в офис. Пожелав пиццы, вы «вызвали» функцию, попросив ассистента: «Принеси мне кусочек пиццы!». Конечно же, в зависимости от настроения, вы можете захотеть кусочек пряной, сырной или оливковой пиццы. Чтобы ваши инструкции были более точными, расскажите ассистенту, чего именно вы желаете. Каждый раз, заказывая пиццу, вы можете выбрать иной тип.

Функции JavaScript также способны воспринимать информацию, называемую *параметрами*, которые они используют для выполнения своих действий. Например, вы хотите создать функцию для расчета общей стоимости заказа посетителя, поэтому должны сообщить ей, сколько стоит товар и сколько единиц заказано.

Для начала, создавая функцию, поместите имя новой переменной в круглые скобки — это параметр. Основная структура будет выглядеть так:

```
function имяфункции(параметр) {
// код JavaScript, который необходимо выполнить
}
```

Параметр — это просто переменная, поэтому вы можете выбрать любое допустимое для нее имя (см. раздел «Создание переменной» главы 2). Например, вы хотите сэкономить несколько нажатий на клавиши каждый раз, когда набираете код для отображения чего-либо на странице. Вы создаете простую функцию, позволяющую заменить функцию браузера `document.write()` более кратким именем:

```
function print(message) {
document.write(message);
}
```

Имя функции — `print`, у нее есть один параметр — `message`. Когда вы вызываете функцию, она получает информацию (сообщение, которое необходимо напечатать), а затем с использованием команды `document.write()` выводит сообщение на страницу. Конечно, функция ничего не делает, пока вы ее не запустите, поэтому где-нибудь в другом месте вашего кода вы должны вызвать эту функцию таким образом:

```
print('Привет, мир.');
```

Во время выполнения этого кода вызывается функция `print` и ей посылается строка `'Привет, мир.'`, которую она выводит на страницу. Технически процесс посылания информации функции называется *передачей аргумента*. В данном примере текст `'Привет, мир.'` — это аргумент. Аргументы — это значения, передаваемые функции, которые соответствуют параметрам, определенным при создании функции.

Даже в такой простой функции, как эта, логика действий и принцип работы могут вас запутать, особенно если вы новичок в программировании. Далее приведено пошаговое объяснение процесса, показанного на рис. 3.8.

1. **Интерпретатор JavaScript считывает функцию и сохраняет ее в памяти. Этот шаг просто подготавливает браузер к выполнению функции, которое произойдет позднее.**
2. **Функция вызывается, ей передается информация `'Привет, мир.'`.**

3. **Информация, переданная функции, сохраняется в новой переменной под именем `message`. Этот шаг эквивалентен коду: `var message = 'Привет, мир';`.**
4. **Наконец, функция выполняется, выводя на веб-страницу значение, сохраненное в переменной `message`.**

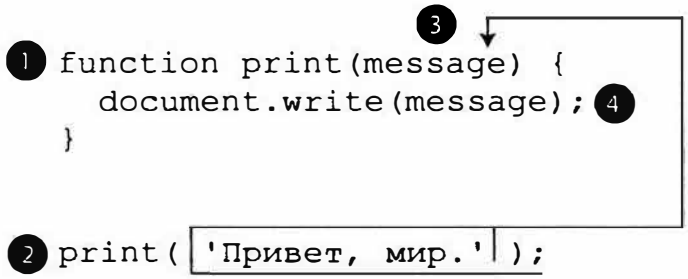

Рис. 3.8. Обычно вы сначала создаете функцию, а потом используете ее. Функция `print()` в этом примере создана в первых трех строках, но код внутри нее не выполняется вплоть до последней строки

Функция не ограничена одним параметром. Вы можете присвоить ей любое количество аргументов. Вам просто нужно указать каждый параметр в функции следующим образом:

```
function имяфункции(параметр1, параметр2, параметр3) {
// код JavaScript, который необходимо выполнить
}
```

А затем вам нужно вызвать функцию с тем же количеством аргументов, в том же порядке:

```
имяфункции(аргумент1, аргумент2, аргумент3);
```

В данном примере при вызове функции *аргумент1* сохранен в переменной *параметр1*, *аргумент2* — в *параметр2* и т. д. Расширяя функцию, приведенную выше, вы, допустим, хотите не только напечатать сообщение на веб-странице, но и задать HTML-элемент (тег) для выделения его в остальном тексте. Так вы можете оформить сообщение как заголовок абзаца. Новая функция будет выглядеть следующим образом:

```
function print(message,tag) {
document.write('<'+ tag +'>'+ message +'</'↵
+ tag +'>');
}
```

Вызов функции будет выглядеть следующим образом:

```
print('Привет, мир.', 'p');
```

В данном случае вы присваиваете функции два аргумента: `'Привет, мир.'` и `'p'`. Эти значения сохранены в двух переменных: `message` и `tag`. В результате на странице появляется новый абзац: `<p>Привет, мир.</p>`.

Вы можете присваивать функциям не только последовательности символов, но и любой тип переменной JavaScript, массив, число или логическое значение.

Запрос данных от функций

Иногда функция просто делает что-то, например, выводит на страницу сообщение, перемещает объект по экрану, проверяет элементы формы. Но иногда необходимо получить от функции какие-то данные как результат ее работы, ведь от функции «Принеси мне кусочек пиццы» будет мало толку, если в результате вы ничего не получите. Точно также, функция, вычисляющая общую стоимость товаров в корзине, не будет слишком полезной, если не сообщит вам окончательную стоимость.

Некоторые функции, встроенные в JavaScript, как мы уже видели, возвращают значения. Например, команда `prompt()` (см. раздел «Запрос информации на практике» главы 2) выводит диалоговое окно с текстом, и то, что пользователь вводит в него, возвращается. Как вы уже знаете, можно сохранить возвращенное значение в переменной и что-нибудь с ним сделать:

```
var answer = prompt('В каком месяце вы родились?', '');
```

Ответ посетителя, введенный в диалоговое окно, сохраняется в переменной `answer`. Вы можете проверить значение этой переменной, используя управляющие инструкции или произвести над ней любые действия, предусмотренные в языке JavaScript.

Для возвращения значений от функций используйте ключевое слово `return`, за которым следует величина, которую вы хотели бы вернуть:

```
function имяфункции(параметр1, параметр2) {
// код JavaScript, который необходимо выполнить
 return value;
}
```

Например, вы хотели бы подсчитать общую стоимость покупки, включая НДС. Напишите следующий сценарий:

```
var TAX = 0.18; // 18% НДС
function calculateTotal(quantity, price) {
var total = quantity * price * (1 + TAX);
var formattedTotal = total.toFixed(2);
return formattedTotal;
}
```

Первая строка сохраняет процент налога в переменной, называемой `TAX` (что позволяет вам изменить процент, просто обновив эту строку кода). Следующие три строки определяют функцию. Пока не задумывайтесь о том, что происходит внутри функции. Работа с числами будет подробно рассмотрена в главе 16. Важная часть функции находится в четвертой строке — это инструкция возврата. Он возвращает значение, сохраненное в переменной `formattedTotal`. Чтобы воспользоваться возвращенным значением, вы должны сохранить его в переменной. В данном примере вы можете вызвать функцию вот так:

```
var saleTotal = calculateTotal(2, 16);
document.write('Общая стоимость: ' + saleTotal);
```

В данном случае значения 2 и 16 являются аргументами функции. Первое число обозначает число купленных единиц товара, а второе — его цену. Результат возвращается от функции и сохраняется в новой переменной `saleTotal`, которая используется как часть команды `document.write()` для вывода общей стоимости проданного товара, включая налог.

ПРИМЕЧАНИЕ

Ключевое слово `return` должно быть последней инструкцией в функции, так как интерпретатор JavaScript выходит из функции сразу после того, как встречает инструкцию `return`. Строки кода, следующие после данной инструкции, никогда не будут выполнены.

Однако вам необязательно сохранять возвращенное значение в переменной. Вы можете сразу использовать его в другой инструкции, например, так:

```
document.write('Итого: ' + calculateTotal(2, 16));
```

В данном случае вызывается функция, а ее возвращенное значение добавляется к строке 'Итого: ', которая затем отображается в документе. На первый взгляд этот способ использования функции может показаться трудным. Вы можете сделать дополнительный шаг — просто сохранить результат функции в переменной, а затем использовать ее в сценарии.

ПРИМЕЧАНИЕ

Функция может возвращать только одно значение. Если вы хотите вернуть несколько элементов, сохраните их в массиве и затем возвращайте этот массив. Подробнее о массивах вы можете узнать в главе 2.

Предупреждение конфликта переменных

Одно из серьезных преимуществ функций состоит в том, что они уменьшают объем работы при программировании. Допустимо использовать какую-либо функцию в разных проектах. Например, функция, рассчитывающая стоимость доставки и НДС, пригодится в любой форме заказа. Поэтому вы можете скопировать и вставить эту функцию в другие сценарии на вашем сайте или использовать в других проектах.

Проблема может возникнуть, если вы просто вставите функцию в уже созданный сценарий. Давайте посмотрим, что случится, если в сценарии есть переменная с тем же именем, что и в функции:

```
var message = 'За пределами функции';
function warning(message) {
alert(message);
}
warning('Внутри функции'); // 'Внутри функции'
alert(message); // 'За пределами функции'
```

Обратите внимание, что переменная message появляется и вне функции (первая строка сценария), и в функции (в качестве параметра). Параметр — это ведь просто переменная с данными, которая используется при вызове функции. В данном случае вызов функции warning('Внутри функции'); передает ей строку и сохраняет эту строку в переменной message. Похоже, что теперь переменных

message две. Что же происходит со значением переменной message, созданной в первой строке сценария?

Вы можете подумать, что первоначальное значение ('За пределами функции') заменено новым — строкой ('Внутри функции'), но это не так. Когда вы запустите сценарий, то увидите два окна оповещения: 'Внутри функции' и 'За пределами функции'. Это и есть две переменные с именами message, но они существуют в разных местах (рис. 3.9),

Рис. 3.9. Параметр функции виден только внутри нее, поэтому первая строка функции function warning (message) создает переменную message, доступ к которой возможен только внутри функции. Когда функция выполнена, переменная исчезает

Интерпретатор JavaScript обращается с переменными внутри функции не так, как с переменными, созданными вне функции. Как говорят программисты, каждая функция имеет собственную *область видимости*. Область видимости функции как стена, которая ее окружает. Переменные «за стеной» не видны в оставшейся части сценария, которая находится «снаружи стены». При первом знакомстве концепция области видимости может запутать, но, если разобраться в ней, она окажется очень полезной. Поскольку функция располагает собственной областью видимости, вы можете не беспокоиться, что имена, используемые для параметров функции, будут заменены переменными из другой части сценария или вступят с ними в конфликт.

До сих пор единственной рассмотренной нами ситуацией было использование переменных в качестве параметров. Но как насчет переменных, которые созданы внутри функции, но не являются параметрами, например:

```
var message = 'За пределами функции';
function warning () {
var message ='Внутри функции';
```

```
alert(message);
}
warning(); //'Внутри функции'
alert(message); //'За пределами функции'
```

В данном случае код создает переменную `message` дважды: в первой строке сценария и вновь в первой строке внутри функции. Ситуация та же, что и с параметрами, — набрав строку кода `var message` внутри функции, вы создали новую переменную в пределах области видимости функции. Этот тип переменной называется *локальной переменной*, поскольку она видна только в пределах функции, основной сценарий и другие функции не видят этой переменной и не могут получить к ней доступ.

Однако переменные, созданные в основной части сценария (за пределами функции), существуют в *глобальном контексте*. Все функции сценария имеют доступ к переменным, созданным в основной его части. Например, в коде, приведенном ниже, переменная `message` создана в первой строке сценария — это *глобальная переменная*, и функция имеет к ней доступ:

```
var message = 'Глобальная переменная';
function warning() {
alert(message);
}
warning(); // 'Глобальная переменная'
```

Эта функция не имеет параметров и не определяет переменную `message`, поэтому, если выполняется часть `alert(message)`, функция ищет глобальную переменную с именем `message`. В данном случае, поскольку искомая переменная существует, появляется диалоговое окно с текстом `'Глобальная переменная'`.

Существует потенциальная проблема, связанная с локальными и глобальными переменными: переменная существует в области видимости функции только в том случае, если является параметром этой функции или если создана в пределах функции с использованием ключевого слова `var`.

На рис. 3.10 изображена подобная ситуация. Верхний фрагмент — пример того, как глобальная переменная `message` и локальная переменная одной из функций `message` могут существовать бок о бок. Ключевое

значение имеет первая строка в функции `var message = 'Внутри функции;`. Используя слово `var`, вы создаете локальную переменную.

Сравните этот пример с кодом в нижней части рис. 3.10. В данном случае в функции не использовано ключевое слово `var`. Строка кода `message = 'Внутри функции;` не создает новую локальную переменную, она просто сохраняет в глобальной переменной `message` новое значение. Результат? Функция переписывает глобальную переменную, изменяя ее начальное значение.

Локальная переменная в функции

```
var message ='За пределами функции';

function warning() {
  var message ='Внутри функции';

  alert( message ); //'Внутри функции'
}
warning();
alert( message ); //'За пределами функции'
```

Глобальная переменная в функции

```
var message = 'За пределами функции';

function warning() {
  message ='Inside the function';
  alert( message ); //'Внутри функции'
}
warning();
alert( message ); //'Внутри функции'
```

Рис. 3.10. Есть тонкое, но критически важное различие при присвоении значений переменным внутри функции. Если вы хотите, чтобы переменная была доступна только для кода внутри функции, убедитесь, что используете ключевое слово `var` для создания переменной внутри функции (пример в верхней части). Если вы не используете слово `var`, то просто сохраняете новое значение в глобальной переменной (пример в нижней части)

Понятие области видимости функции может поначалу сильно запутать, поэтому, возможно, приведенное выше рассуждение не имеет для вас большого смысла. Но это только пока. Просто запомните одну вещь:

если переменные, создаваемые в сценарии, содержат, как кажется, не те значения, которые вы ожидаете увидеть, может иметь место проблема с областью видимости функции. Если подобное происходит, перечитайте данный раздел еще раз.

Создание простой викторины на практике

Пришло время обобщить информацию этой главы и написать целую программу. Вы создадите простую викторину, задающую испытуемому вопросы и оценивающую его ответы. Мы рассмотрим несколько способов решения этой задачи и обсудим эффективные техники программирования.

Как всегда, первый шаг — определение того, что именно должна делать программа. Есть несколько задач, которые мы хотим решить с ее помощью:

- **Задать вопросы.** Если вы хотите сделать викторину, необходимо придумать, как задавать вопросы. Вам уже знаком простой способ получения обратной связи от посетителя — команда `prompt()`. Вам также нужен список вопросов. Поскольку массивы хорошо подходят для хранения списков, станем сохранять вопросы викторины с их помощью.
- **Сообщить испытуемому, прав он или нет.** Для начала вам будет необходимо определить, дал ли испытуемый правильный ответ. Эту задачу можно возложить на управляющую инструкцию. Чтобы сообщать о правильном ответе или об ошибке, можно использовать команду `alert()`.
- **Вывести результаты викторины.** Вы должны оценивать, насколько хорошо испытуемый справляется с викториной. Поэтому необходима переменная, которая отслеживает количество правильных ответов. Затем для сообщения конечного результата вы можете воспользоваться либо командой `alert()`, либо методом `document.write()`.

Существует много способов решения подобной задачи. Некоторые начинающие программисты могут избрать метод «грубой силы» и повторять один и тот же код для каждого вопроса. Например, код JavaScript, дающий возможность задать два первых вопроса викторины, может выглядеть так:

```
var answer1=prompt('Сколько лун у планеты Земля?','');
if (answer == 1 ) {
alert('Верно!');
} else {
alert('Неверно, правильный ответ: 1');
}
var answer2=prompt('Сколько лун у планеты ↵
Сатурн?','');
if (answer2 == 31) {
alert('Верно!');
} else {
alert('Неверно, правильный ответ: 31');
}
```

Этот подход кажется логичным, поскольку цель программы — задавать один вопрос за другим. Однако подобный путь неэффективен. Если в программе многократно выполняются одни и те же шаги, время подумать о цикле или функции. Мы создадим программу, которая использует цикл для прохождения через каждый из вопросов викторины, и функцию, которая осуществляет задавание вопросов.

1. **В текстовом редакторе откройте файл *03_03.html*.**

Начните с создания переменных, которые будут отслеживать количество вопросов и верных ответов в викторине.

2. **Между открывающим и закрывающим тегами элемента `script` в разделе заголовка страницы введите следующий код:**

```
var score = 0;
```

Эта переменная сохраняет количество вопросов, на которые испытуемый дал правильный ответ. В начале викторины, пока не дан ответ ни на один вопрос, переменная имеет значение 0. Далее мы создадим список вопросов и ответов на них.

3. **Нажмите клавишу Enter, чтобы добавить новую строку, и напечатайте в ней: `var questions = [`.**

Вы сохраните все вопросы в одном массиве, являющемся, по сути, переменной, в которой может содержаться несколько элементов. Только что записанный код — это первая часть инструкции массива. Далее

вы введете массив в виде нескольких строк кода, как описано в разделе «Создание массива» главы 2.

4. **Дважды нажмите клавишу Enter, чтобы добавить две новые строки, и введите символы] ;. Сейчас ваш код должен выглядеть так:**

```
var score = 0;

var questions = [

];
```

Поскольку викторина — это набор вопросов, имеет смысл сохранить каждый из них как элемент массива. Затем, чтобы задать вопросы, нужно будет просто пройти через все элементы массива. Однако каждый вопрос имеет ответ, поэтому вам нужен способ отслеживания ответов.

Одно из возможных решений — создать еще один массив, например `answers[]`, в котором будут содержаться ответы. Чтобы задать первый вопрос, вам нужно обратиться к первому элементу массива `answers[]`, а чтобы узнать, верен ли ответ — к первому элементу массива `answers[]`. Однако в данном случае возможна проблема, связанная с согласованием двух массивов. Например, вы добавляете вопрос в середину массива `questions[]`, а ответ на него помещаете в начало массива `answers[]`. Тогда первые элементы массивов вопросов и ответов не будут подходить друг другу.

Лучшая альтернатива — это использование *вложенного*, или (если вы хотите, чтобы это звучало устрашающе) *многомерного* массива. Это просто означает, что вы создаете массив, который содержит и вопрос, *и* ответ, то есть вы сохраняете массив `answers[]` как элемент массива `questions[]`. Иначе говоря, вы создаете список, каждый элемент которого — еще один список.

5. **Щелкните мышью по пустой строке между символами [и] и добавьте код, выделенный ниже полужирным шрифтом:**

```
var questions = [

['Сколько лун у планеты Земля?', 1],

];
```

Код `['Сколько лун у планеты Земля?', 1]`, является массивом из двух элементов. Первый элемент — вопрос, второй — ответ.

Этот массив — первый элемент массива `questions[]`. Вы не присваиваете этому массиву имя, поскольку он вложен в другой. Запятая в конце строки отмечает конец первого элемента в массиве `questions[]` и означает, что далее будет следовать другой элемент.

6. **Нажмите клавишу Enter, чтобы создать пустую строку, и добавьте в сценарий две строки, выделенные ниже полужирным шрифтом:**

```
var questions = [
['Сколько лун у планеты Земля?', 1],
['Сколько лун у планеты Сатурн?', 31],
['Сколько лун у планеты Венера?', 0]
];
```

Вы только что добавили в викторину еще два вопроса. Обратите внимание, что после последнего элемента массива *нет* запятой. Размещение всех вопросов в одном массиве делает вашу викторину гибкой. Если вы хотите добавить в нее еще один вопрос, просто вставьте очередной вложенный массив с вопросом и ответом.

Когда заданы основные переменные для викторины, пришло время решить, как будет задаваться каждый вопрос. Вопросы сохранены в массиве. Вы хотите задавать их последовательно. Как вы помните, цикл — отличный способ пройти через каждый из элементов массива (см. раздел «Циклы и массивы»).

7. **Щелкните мышью после символов `];` (конец массива `questions[]`), нажмите клавишу Enter, чтобы создать новую пустую строку, и добавьте следующий код:**

```
for (var i=0; i<questions.length; i++) {
```

Эта строка — часть цикла `for` (см. раздел «Циклы `for`»). Она делает следующее: создает новую переменную `i` и сохраняет в ней значение 0. Это переменная-счетчик, которая отслеживает, сколько раз был выполнен цикл. Вторая часть `i<questions.length` — это условие, как в инструкции `if/else`. Она проверяет, меньше ли переменная `i`, чем количество элементов в массиве `questions[]`. Если это так, цикл выполняется снова. Когда значение переменной `i` становится равной общему числу элементов массива или превышает его, цикл завершается. Наконец, фрагмент `i++` изменяет значение переменной `i` при каждом прохождении цикла (добавляет к нему 1).

Теперь пришло время создания ядра цикла — кода JavaScript, выполняемого при каждом прохождении цикла.

8. **Нажмите клавишу Enter, чтобы создать новую пустую строку, и добавьте следующий код:**

```
askQuestion(questions[i]);
```

Вместо того чтобы записывать весь программный код, вы просто запускаете функцию, задающую вопросы. Функция, которую вы сейчас создадите, называется `askQuestion()`. Каждый раз, выполняя цикл, вы посылаете функции один из элементов массива — часть `questions[i]`. Помните, что вы получаете доступ к элементу массива, используя значения индекса, то есть `questions[0]` — это первый элемент массива, `questions[1]` — второй и т. д.

Создавая функцию для задавания вопросов, вы пишете более гибкую программу. Вы можете перемещать функцию и повторно использовать ее в других программах по своему усмотрению. Наконец, вы завершаете код цикла.

9. **Нажмите клавишу Enter, чтобы создать новую пустую строку, и напечатайте символ } для обозначения окончания цикла. В результате код цикла должен выглядеть следующим образом:**

```
for (var i=0; i<questions.length; i++) {
askQuestion(questions[i]);
}
```

Да, вот он — простой цикл, который вызывает функцию для каждого из вопросов викторины. Сейчас мы создадим «сердце» викторины — функцию `askQuestion()`.

10. **Вставьте пустую строку перед только что добавленным циклом `for`.**

Другими словами, вы добавляете функцию между инструкциями, которые определяют основные переменные, и началом цикла. Функции в сценарии можно определять где угодно, но большинство программистов размещают их в начале программы. В этом сценарии глобальные переменные, такие как `score` и `questions`, определяются первыми, чтобы вам было проще находить и изменять их; далее идут функции, поскольку они образуют ядро большинства сценариев, затем следуют пошаговые инструкции (как цикл, описанный выше).

11. Добавьте следующий код:

```
function askQuestion(question) {


}
```

Он обозначает тело функции. В начале и в конце всегда следует вводить открывающую и закрывающую фигурные скобки и только потом добавлять сценарий. Это делается для того, чтобы случайно не забыть вставить закрывающую фигурную скобку.

Данная функция получает один аргумент и сохраняет его в переменной `question`. Обратите внимание, что это не та переменная, которая использовалась для обозначения массива `questions[]`, созданного вами в шаге 8. В данном случае переменная `question` хранит один из элементов массива `questions[]`, который, в свою очередь, является массивом, содержащим два элемента — вопрос и ответ.

12. Добавьте код, выделенный полужирным шрифтом:

```
function askQuestion(question) {
var answer = prompt(question[0], '');
}
```

Вы опять встретились со «старым другом» — командой `prompt()`. Часть, которая вам еще неизвестна, — `question[0]`. С ее помощью вы получаете доступ к первому элементу массива `question`. В данном примере функция получает в качестве аргумента массив, который содержит вопрос и ответ. Например, первый массив будет таким: `['Сколько лун у планеты Земля?', 1]`. Итак, `question[0]` получает доступ к первому элементу: `'Сколько лун у планеты Земля?'`, который используется как аргумент команды `prompt()` в качестве вопроса (он станет отображаться в диалоговом окне).

Все, что испытуемый записывает в диалоговом окне, ваша программа сохраняет в переменной `answer`. Далее вы сравниваете ответы испытуемого с правильными ответами на вопросы.

13. Завершите функцию, добавив код, выделенный полужирным шрифтом:

```
function askQuestion(question) {
var answer = prompt(question[0], '');
```

```
if (answer == question[1]) {
alert('Верно!');
score++;
} else {
alert('Неверно. Правильный ответ ' + question[1]);
}
}
```

Этот код является простой инструкцией `if/else`. Условие `answer == question[1]` проверяет, совпадает ли то, что ввел пользователь (`answer`) со значением, сохраненным в качестве второго элемента массива (`question[1]`). Если это так, отображается сообщение о том, что ответ правильный и добавляется одно очко (`score++`). Конечно же, если дан неправильный ответ, появляется окно с правильным ответом.

Теперь викторина полностью функциональна. Если вы сохраните файл и загрузите его в браузер, то сможете пройти викторину. Однако вы еще не сообщили испытуемому результаты тестирования, и он не знает, сколько раз ответил правильно.

14. Найдите вторую пару тегов элемента `script` в нижней части веб-страницы и введите:

```
var message = 'Вы ответили правильно на ' + score;
```

Вы создаете новую переменную и сохраняете в ней строку `'Вы ответили правильно на '` плюс счет испытуемого. То есть если он все сделал правильно, то переменная `message` будет иметь вид: `'Вы ответили правильно на 3'`. Чтобы упростить чтение сценария, можно разбить длинное сообщение на несколько строк.

15. Нажмите клавишу Enter и введите:

```
message += ' из ' + questions.length;
```

Вы добавили строку `' из '`, а также общее число вопросов к строке `сообщение`, поэтому теперь оно станет выглядеть примерно так: `'Вы ответили правильно на 3 из 3'`. Теперь завершите сообщение и выведите его на экран.

16. Добавьте в сценарий код, выделенный полужирным шрифтом:

```
var message = 'Вы ответили правильно на ' + score;
```

```
message += ' из ' + questions.length;
message += ' вопросов.';
document.write('<p>' + message + '</p>');
```

Сохраните страницу и откройте ее в браузере. Пройдите викторину и оцените ее (рис. 3.11). Если сценарий не работает, вспомните о способах устранения ошибок, описанных в разделе «Отслеживание ошибок» главы 1. Вы также можете сравнить ваш сценарий с полной работающей версией из файла *готовый_03_03.html*.

Попробуйте добавить вопросы к массиву `questions[]`, чтобы сделать викторину длиннее.

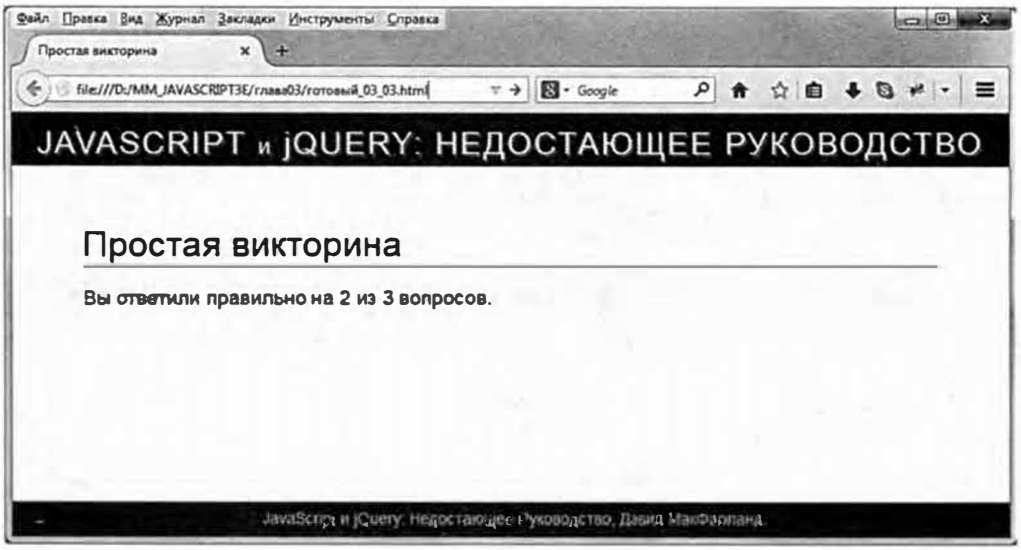

Рис. 3.11. Результаты вашей простой программы-викторины. Когда вы более подробно узнаете о том, как манипулировать веб-страницей (раздел «Добавление контента на страницу» главы 4), реагировать на события (раздел «Что такое события» главы 5) и работать с веб-формами (глава 8) попробуйте переписать эту программу-викторину так, чтобы вопросы появлялись прямо на веб-странице, а счет динамически обновлялся после каждого ответа. Другими словами, скоро вы изучите, как избавиться от этой неуклюжей команды `prompt()`

Теперь, когда вы разобрались со скучными деталями языка JavaScript, пришло время для настоящего веселья. В следующей части вы узнаете о том, что такое jQuery, как ее использовать и самое важное, как эффективно программировать на языке JavaScript и получать при этом огромное удовольствие.

Часть II

НАЧАЛО РАБОТЫ С JQUERY

Глава 4. Введение в jQuery

Глава 5. Действие/реакция: интерактивные страницы с помощью событий

Глава 6. Анимация и эффекты

Глава 7. Распространенные задачи, решаемые с помощью jQuery

Глава 8. Улучшение веб-форм

Глава 4
ВВЕДЕНИЕ В JQUERY

В первых главах книги освещены многие фундаментальные основы языка программирования JavaScript: ключевые слова, концепции, а также синтаксис. Многие из этих концепций довольно просты для понимания («переменная подобна коробке, в которую вы помещаете значение»), но некоторые темы, возможно, заставили вас отправиться на поиски аспирина (например, циклы `for` в предыдущей главе). Правда в том, что большинству людей сложно программировать на языке JavaScript. В самом деле, даже книга, содержащая 1000 страниц, не в состоянии включить в себя всю информацию о языке JavaScript и о том, как он работает в различных браузерах.

Процесс программирования сложен, вот почему это издание посвящено как JavaScript, так и jQuery. Как вы увидите в первом разделе данной главы, jQuery — это библиотека JavaScript, которая позволяет вам приступить к программированию и берет на себя заботу о самых скучных деталях этого процесса. jQuery, чей девиз: «Пиши меньше, делай больше», делает процесс программирования увлекательным, быстрым и стоящим потраченных усилий. С помощью одной строки кода библиотека jQuery позволяет достичь того же результата, что и 100 строк кода исключительно на языке JavaScript. После того, как вы освоите эту и следующую главы, вы сможете делать со своими веб-страницами больше, чем если бы вы изучили книгу, состоящую из 1000 страниц, посвященную одному лишь языку JavaScript.

О библиотеках JavaScript

Во многих программах на языке JavaScript приходится иметь дело с одними и теми же задачами, которые необходимо решать снова и снова: выбор элемента, добавление нового содержимого, показ и сокрытие контента, модифицирование атрибутов тега, определение значения элемента формы и реакция программы на различные действия пользователя. Детали этих простейших действий могут быть достаточно сложными, особенно если вы хотите, чтобы программа работала во всех основных

браузерах. К счастью, *библиотеки* JavaScript позволяют быстро решить многие трудоемкие задачи программирования.

Библиотека JavaScript — это коллекция кодов, предоставляющая простые решения многих повседневных задач программирования. В ней содержатся готовые функции языка JavaScript, которые вы можете добавлять на свои веб-страницы. Эти функции облегчают выполнение распространенных действий. Во многих случаях вы можете заменить большое количество строк вашей программы (и часы тестирования) одной-единственной функцией из библиотеки JavaScript. Существует много библиотек JavaScript, и элементы многих из них используются на ведущих сайтах, таких как Yahoo, Amazon, CNN, Apple и Twitter.

В этой книге используется популярная библиотека jQuery (**www.jquery.com**). Есть и другие библиотеки JavaScript (см. врезку далее в этой главе), но jQuery имеет много преимуществ:

- **Относительно небольшой размер файла.** Сжатая версия библиотеки занимает 96 Кб (версия 1.11) и около 83 Кб (версия 2.1). (Если ваш веб-сервер использует метод сжатия gzip, то вы можете уменьшить размер файла до 38 Кб!)
- **Дружественность веб-дизайнерам.** jQuery не подразумевает, что вы являетесь компьютерным гением. Она лишь предполагает, что большинство веб-дизайнеров знакомы с CSS.
- **Испытанность и надежность.** Библиотека jQuery используется на миллионах сайтов, включая очень популярные, с большим трафиком: Pinterest, MSN.com, Amazon, Microsoft.com, Craigslist и ESPN. На самом деле, она используется более чем на 57% сайтов*. Популярность jQuery — признак ее высокого качества.
- **Бесплатность.** Здесь даже рассуждать не о чем!
- **Большое сообщество разработчиков.** Пока вы читаете эту книгу, множество людей работают над проектом jQuery: пишут код, устраняют ошибки, добавляют новые возможности и размещают на сайте новую документацию и руководства. Библиотека JavaScript, созданная программистом-одиночкой, может очень легко исчезнуть, если автор устанет от своего проекта. jQuery, напротив, проживет долго, потому что поддерживается усилиями программистов со всего мира. Даже крупные компании вроде Microsoft и Adobe вносят свою леп-

* **w3techs.com/technologies/details/js-jquery/all/all**

ту, предоставляя программный код. Пользоваться библиотекой jQuery — это все равно, что иметь команду программистов JavaScript, которая бесплатно работает на вас.

- **Плагины, плагины и плагины.** jQuery позволяет другим программистам создавать плагины — дополнительные программные модули на языке JavaScript, работающие в союзе с jQuery над выполнением определенных задач, созданием эффектов, расширением возможностей, причем их очень просто добавить на веб-страницу. В этой книге вы познакомитесь с плагинами, предназначенными для проверки форм, добавления выпадающих навигационных меню и построения интерактивных слайд-шоу за полчаса, вместо двухнедельной работы. В jQuery, без преувеличения, доступны сотни плагинов.

Работая с данным пособием, вы уже пользовались библиотекой jQuery. Выполняя руководство в главе 1 (см. раздел «Прикрепление внешнего файла JavaScript»), вы добавили всего несколько строк кода JavaScript для создания эффекта проявления.

УСКОРЯЕМ РАБОТУ

Другие библиотеки

jQuery — это не единственная библиотека JavaScript. Существует много других. Некоторые разработаны для решения специфических задач, некоторые направлены на решение всевозможных общих задач, возникающих в работе. Вот некоторые наиболее популярные библиотеки:

- **Yahoo User Interface Library** (**yuilibrary.com**). Это проект сервиса Yahoo, повсеместно применяемый на сайте компании. Программисты Yahoo, постоянно улучшают и дополняют библиотеку, а также обеспечивают хорошей документацией.
- **Dojo Toolkit** (**dojotoolkit.org/**) — еще одна давно используемая библиотека. Располагает очень большим собранием файлов JavaScript, применимых в решении практически любой задачи. Она используется такими крупными компаниями, как ADP, IBM и VMware. Однако она довольно сложна и рассчитана на опытных программистов.
- **Mootools** (**mootools.net/**) — популярная библиотека, направленная на создание сложной анимации и визуальных эффектов.

Существуют и другие библиотеки, обеспечивающие базу для создания веб-приложений, например, Ember.js (**emberjs.com**), AngularJS (**angularjs.org**) и Backbone.js (**backbonejs.org**).

Некоторые библиотеки отличаются небольшим размером и предусматривают простые, но полезные утилиты для программирования на языке JavaScript. Например, небольшая библиотека Underscore.js (**underscorejs.org**) предлагает множество дополнений для языка программирования JavaScript, но не включает визуальные эффекты jQuery и ее функции, относящиеся к технологии AJAX или манипулированию элементами HTML.

Существуют также библиотеки, предназначенные для решения конкретных задач, например, Raphaël (**raphaeljs.com**), единственной целью которой является облегчение процесса рисования векторных изображений в браузере.

Другими словами, существует множество библиотек JavaScript. Тем не менее начинать знакомство с ними лучше всего с jQuery. Затем по мере роста вашего профессионализма вы можете воспользоваться возможностями, предлагаемыми другими библиотеками.

Получение библиотеки jQuery

Библиотека jQuery представляет собой внешний JavaScript-файл, содержащий программный код на языке JavaScript. Как и в случае с любым другим внешним файлом JavaScript (см. раздел «Внешние файлы JavaScript» главы 1), вам необходимо прикрепить его к своей веб-странице. Однако в связи с популярностью jQuery у вас есть несколько способов ее добавления на веб-страницу: вы можете либо использовать версию, размещенную на ресурсах Google, Microsoft или jQuery.com (рис. 4.1), либо скачать файл jQuery на собственный компьютер и добавить его на свой сайт.

Первый метод использует *сеть доставки (и дистрибуции) контента* (CDN, content distribution network), это означает, что файл jQuery расположен на другом веб-сайте, который отправляет данный файл любому, кто его запрашивает. У данного подхода есть пара преимуществ: во-первых, вы можете снизить нагрузку на ваш веб-сервер, позволив ресурсам Google, Microsoft или jQuery.com распространять файл среди посетителей вашего сайта. К тому же, CDN имеет дополнительное преимущество, выражающееся в наличии серверов, расположенных по всему миру. Поэтому если кто-то в Сингапуре, например, просматривает ваш сайт, то он получит файл jQuery с сервера, который, вероятно, расположен гораздо ближе, чем ваш веб-сервер, это означает, что посетитель получит файл скорее, и ваш сайт будет отображаться быстрее. И, наконец, самое главное, так как другие дизайнеры также использу-

ют сети CDN, есть довольно высокая вероятность того, что у какого-нибудь посетителя вашего сайта уже есть файл jQuery, сохраненный в кэше браузера. Так как он уже скачал jQuery-файл с ресурса Google при посещении другого сайта, ему не нужно загружать его снова при посещении вашего сайта, что приводит к значительному увеличению скорости.

У использования CDN также есть пара недостатков: во-первых, чтобы воспользоваться данным методом, посетители должны быть подключены к Интернету. Это становится проблемой, если вам нужно, чтобы ваш сайт работал в автономном режиме, например, в музее или во время демонстрации процесса программирования в классе. В этом случае, вам необходимо скачать файл jQuery с сайта jQuery.com (далее вы узнаете, как это сделать) и добавить его на ваш сайт. Добавление собственного файла jQuery также гарантирует то, что ваш сайт будет продолжать работать, даже если CDN-серверы будут отключены. (Конечно, если серверы Google когда-либо перестанут работать, то в мире возникнут более серьезные проблемы, чем сложности с работой вашего веб-сайта.)

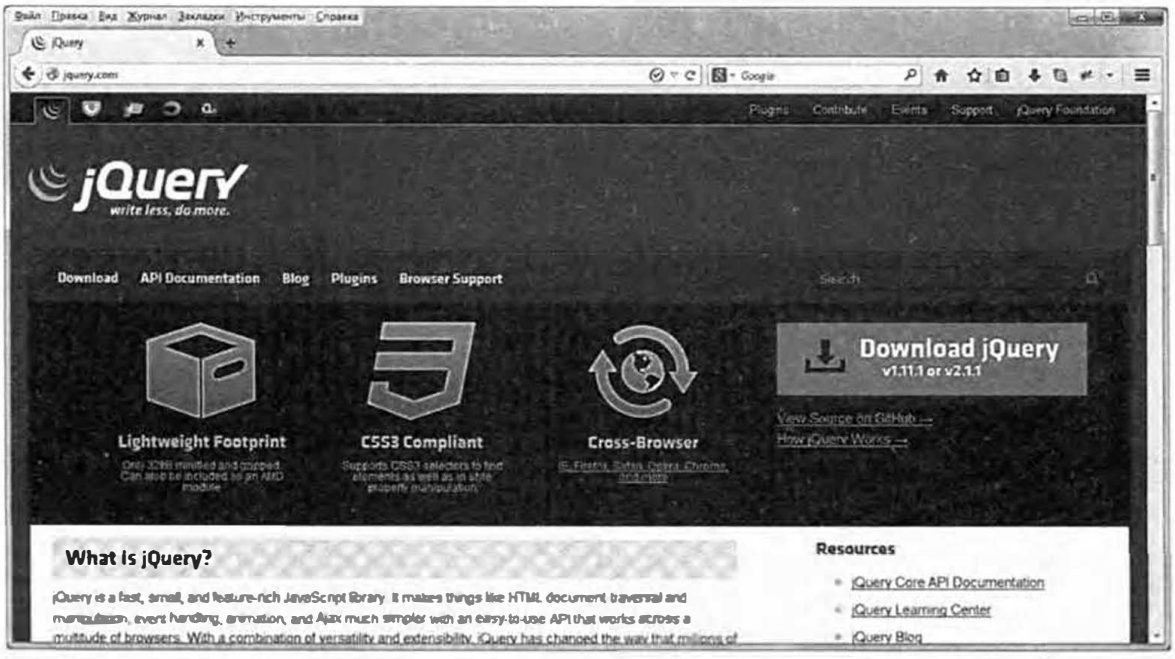

Рис. 4.1. На домашней странице jQuery.com вы можете загрузить библиотеку jQuery и изучить ее API-интерфейс (нечто вроде словаря, в котором перечислены все встроенные в jQuery функции). Значки в левой верхней части страницы позволяют перейти к другим проектам, например, jQuery UI (о котором вы узнаете в части III данной книги), jQuery Mobile (для создания веб-сайтов, которые можно просматривать с помощью мобильных устройств), Sizzle (встроенная в jQuery библиотека JavaScript, которая облегчает выбор и изменение частей веб-страницы) и QUnit (для тестирования программ JavaScript)

Ссылки на файл jQuery, расположенный на сервере CDN

Ресурсы Microsoft, jQuery и Google позволяют разместить файл jQuery на одной из ваших веб-страниц, используя их серверы. Например, для ссылки на версию jQuery 1.11.0 с использованием CDN корпорации Microsoft следует добавить такую строку кода в раздел заголовка — `head` — вашей веб-страницы (прямо перед закрывающим тегом `</head>`), например, так:

```
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/↵
jquery-1.11.0.min.js">
</script>
```

При использовании CDN jQuery вы можете использовать следующий код:

```
<script src="http://code.jquery.com/↵
jquery-1.11.0.min.js"></script>
```

А код при использовании CDN Google выглядит так:

```
<script src="//ajax.googleapis.com/ajax/libs/↵
jquery/1.11.0/jquery.min.js"></script>
```

Вам нужно использовать всего лишь одну из этих строк на странице, в зависимости от выбранной сети CDN. CDN Google, по-видимому, является самой популярной сетью, так что если вы не уверены относительно того, какую сеть CDN использовать, используйте серверы ресурса Google. Если вы решите использовать версию jQuery 2 (см. врезку далее в этой главе), просто измените значение `1.11.0` в приведенных выше строках кода на `2.1.0` (или другой номер версии библиотеки jQuery, которые перечислены на странице **jquery.com/download/**). Например, чтобы использовать CDN Google для загрузки версии jQuery 2.1.13, добавьте следующую строку кода:

```
<script src="//ajax.googleapis.com/ajax/libs/↵
jquery/2.1.13/jquery.min.js">
</script>
```

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Версии библиотеки jQuery 1 и 2

На сайте jQuery.com доступны две версии библиотеки — 1 и 2. Какую мне следует использовать?

На момент написания этой книги версии jQuery 1.11 и 2.1 функционально не отличались друг от друга. Большая разница и причина выпуска версии jQuery 2 заключается в том, что в ней отсутствует поддержка версий 6, 7 и 8 браузера Internet Explorer. Работа более ранних версий Internet Explorer часто отличается от работы поздних версий браузера, поэтому для обеспечения функционирования новых возможностей необходим дополнительный код JavaScript. Поддержка этих браузеров требует большего объема кода, что приводит к увеличению размера файла библиотеки jQuery.

В надежде на то, что браузеры Internet Explorer 6, 7 и 8 в один прекрасный день перестанут использоваться, команда jQuery создала облегченную версию библиотеки, в которой отсутствует поддержка этих браузеров. Тем не менее версии Internet Explorer 6, 7 и 8 все еще используются, поэтому версия 1 библиотеки jQuery по-прежнему доступна. На самом деле, программа Internet Explorer 8 по-прежнему является наиболее широко используемой версией браузера Internet Explorer. Поэтому в этой книге вы будете использовать версию jQuery 1.11. Она имеет те же функции, что и jQuery 2, но поддерживает старые версии браузера Internet Explorer. Пока посетители вашего сайта используют Internet Explorer 8 вам следует применять последнюю версию jQuery 1 (на момент написания данной книги это версия 1.11).

В дальнейшем любые новые функции, которые команда jQuery добавит в библиотеку, будут относиться к версии 2. В версию 1 окажутся включены только исправления ошибок. Однако не волнуйтесь, все, что вы узнаете из этого издания, будет работать в обеих версиях. Тем не менее, если после прочтения книги в версию jQuery 2 добавится какая-нибудь новая фантастическая функция, вы сможете рассмотреть возможность переключения на эту версию.

ПРИМЕЧАНИЕ

Вы могли заметить, что ссылка на CDN-сервер Google выглядит несколько необычно. Она не начинается со значения *http://*, как ссылка на CDN-сервер Microsoft или jQuery. Это URL-адрес относительно протокола. Это означает, что при загрузке файла браузер применит используемый в настоящее время протокол. Например, если ваша страница отправляется через защищенный сервер по протоколу https, то при отправке файла jQuery будет использоваться этот же протокол. Более подробную информацию вы можете найти на сайте **www.paulirish.com/2010/the-protocol-relative-url/**. С такими URL-адресами есть одна проблема: они работают только при просмотре страницы с веб-сервера. Если вы используете URL-адреса относительно протокола в локальных файлах, которые вы просматриваете, открывая их через свой веб-браузер, они не станут работать.

Загрузка файла jQuery

Вы можете легко загрузить файл jQuery и добавить его на свой сайт вместе со всеми остальными веб-страницами и файлами. Файлы примеров, которые вы загрузили с сайта издательства, включают файл библиотеки jQuery, но так как команда jQuery регулярно обновляет библиотеку, вы можете найти последнюю версию по адресу: **jquery.com/download/**.

Чтобы загрузить последнюю версию библиотеки jQuery:

1. **Посетите сайт jquery.com/download/.**

   Эта страница содержит информацию о коде, список сетей CDN, упомянутых выше, и предыдущие версии библиотеки jQuery.

2. **Выберите версию 1.x или 2.x.**

   В данной книге используется версия 1.11, за более подробной информацией обратитесь к приведенному выше тексту в рамке. Если аудитория вашего сайта использует программу Internet Explorer 8, выберите версию 1.11. jQuery-файл поставляется в двух версиях на сайте загрузки — минимизированный и несжатый. Размер несжатого файла очень большой (более 280 КБ), и предоставляется он для того, чтобы вы смогли больше узнать о библиотеке jQuery, изучая ее код. Код включает множество комментариев (см. раздел «Комментарии» главы 2), которые помогают разобраться в том, для чего предназначены различные части файла. (Однако для того, чтобы понять комментарии, вам нужно узнать *очень многое* о языке JavaScript.)

   Минимизированную версию файла библиотеки jQuery следует использовать для веб-сайтов. Этот файл гораздо меньше обычного JavaScript-файла: все комментарии и ненужные пробелы убраны (табуляция, разрывы строк и т. д.), что делает файл быстро загружаемым, но более сложным для чтения.

ПРИМЕЧАНИЕ

Обычно вы можете опознать минимизированный файл JavaScript по наличию части *.min* в имени файла, например, имя *jQuery-1.11.0.min.js* указывает на то, что данный файл содержит минимизированный вариант библиотеки jQuery версии 1.11.0.

3. **Щелкните правой кнопкой мыши по ссылке на сжатый файл и выберите пункт Сохранить по ссылке как (Save Link As) в появившемся контекстном меню.**

Если вы просто щелкните по ссылке, вы не сможете скачать файл: вместо этого веб-браузер отобразит весь код в окне программы, так что вы должны использовать метод «Сохранить как».

4. **Перейдите в папку на компьютере, в которой находится ваш сайт, и сохраните файл.**

   Вы можете сохранить файл jQuery, где угодно на вашем сайте, но многие веб-дизайнеры хранят свои внешние файлы JavaScript в специальной папке. Обычно эта папка называется *scripts*, *libs*, *js* или *_js*.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Версии библиотеки jQuery, используемые в этой книге

Я вижу, что в данной книге используется версия jQuery 1.11.0, но текущая версия на сайте jQuery — 1.11.x. Является ли это проблемой?

Библиотека jQuery постоянно развивается. Часто обнаруживаются новые ошибки и команда jQuery старательно работает над их исправлением. Кроме того, по мере выхода новых веб-браузеров с новыми возможностями и улучшенной поддержкой действующих стандартов, команда jQuery обновляет библиотеку, чтобы она могла более эффективно работать с этими браузерами. Наконец, иногда в библиотеку jQuery добавляются новые возможности, чтобы сделать ее более полезной для веб-программистов. Поэтому вполне вероятно, что вы можете найти более новые версии библиотеки jQuery, чем та, которая используется в данной книге. Если существует более новая версия, то, конечно, используйте ее.

Библиотека jQuery совершенствовалась в течение многих лет и теперь ее основная функциональность практически не изменяется. Хотя jQuery-программисты часто что-то придумывают, чтобы сделать jQuery быстрее, заставить лучше работать в различных браузерах и исправить ошибки, то, как вы ее используете, как правило, немногое меняет. Другими словами, хотя программисты могут изменить jQuery-функцию, чтобы она работала лучше, то, как вы *используете* эту функцию — имя функции, аргументы, которые вы ей даете и значения, которые она возвращает, не часто меняются. Это означает, что то, что вы узнаете в этой книге, скорее всего, будет работать с новой версией jQuery, но только быстрее и лучше.

Однако это не всегда так. Например, спустя шесть месяцев после выпуска последнего издания данной книги, вышла версия jQuery 1.9, из которой исчезла одна из команд, используемых в примерах, поэтому читатели, применяющие последнюю версию jQuery, обнаружили, что некоторые руководства не работают.

Вы можете определить, насколько сильно одна версия jQuery отличается от другой, используя систему нумерации. Первая цифра указывает на значительную новую версию, например, jQuery версия 1 или версия 2. (Как говорилось выше, версия 2 имеет те же функции, что и версия 1.11, однако не поддерживает программу Internet Explorer 8 или более ранние версии данного браузера.)

Далее через точку идут релизы, например, jQuery 1.1, 1.2, 1.3 и т. д. Каждая из этих цифр, как правило, указывает на новые функции, переписанные старые функции для достижения большей эффективности в работе и т. д. Наконец, последняя цифра, такая, как 0 в версии jQuery 1.11.0, обычно относится к исправлению какой-либо ошибки (в данном случае в версии jQuery 1.11). Так что если вы используете версию jQuery 1.11.0 и выходит версия 1.11.3, то следует загрузить новый файл, так как новая версия, вероятно, будет включать исправления ошибок, обнаруженных в версии 1.11.0.

Чтобы выяснить, что изменилось в новой версии, просто перейдите на страницу загрузки (**jquery.com/download/**). Там вы найдете ссылку на страницу «Release notes» («Заметки о выпуске»). Например, на странице **blog.jquery.com/2014/01/24/jquery-1-11-and-2-1-released/** обсуждаются версии 1.11.0 и 2.1. Заметки о выпуске содержат перечисление изменений, внесенных в эту версию. После прочтения списка изменений вы сможете решить для себя, стоит ли делать обновление. Например, если изменения касаются функций, которые вы не используете на своем сайте, вы, вероятно, можете пропустить это обновление; однако, если изменения являются исправлением ошибок, связанных с функциями, которые вы используете, то следует загрузить новый файл библиотеки. Если вы используете плагины jQuery на своем сайте, то прежде, чем обновлять библиотеку до последней версии, вам следует убедиться, что плагин работает с новой версией jQuery.

Добавление библиотеки jQuery на страницу

Если вы используете одну из CDN-версий jQuery, то вы можете указать на это с помощью одного из фрагментов кода, перечисленных в разделе «Ссылки на файл jQuery, расположенный на сервере CDN». Например, чтобы использовать CDN-версию сервиса Google, нужно добавить элемент `script` в раздел заголовка страницы следующим образом:

```
<script src="//ajax.googleapis.com/ajax/libs/ ↵
jquery/1.11.0/jquery.min.js"></script>
```

СОВЕТ

При использовании CDN Google вы можете не указывать часть номера версии. Если вы используете в ссылке номер 1.11 вместо 1.11.0 (`<script src="ajax.googleapis.com/ajax/libs/jquery/1.11/jquery.min.js"></script>`), то сервис Google загрузит последнюю версию семейства 1.11, например, 1.11.2. Если библиотека jQuery обновлена до версии 1.11.9, то Google загрузит данную версию. Это умная техника, поскольку (согласно тексту в рамке «Версии библиотеки jQuery») незначительные изменения версий с 1.11.0 по 1.11.2 часто являются исправлениями ошибок, что может улучшить функционирование вашего сайта.

После того, как вы загрузили библиотеку jQuery на свой компьютер, необходимо прикрепить данный файл к веб-странице, на которой вы предполагаете его использовать. Библиотека jQuery представляет собой внешний файл *.js*, так что вы можете прикрепить ее тем же способом, который описан в разделе «Внешние файлы JavaScript» главы 1. Например, вы сохранили файл *jquery.js* в папке с именем *js* в корневом каталоге вашего сайта. Чтобы прикрепить файл к вашей домашней странице, вы можете добавить следующий код в ее раздел заголовка:

```
<script src="js/jquery-1.11.0.min.js"></script>
```

После прикрепления файла jQuery вы можете добавить свои собственные сценарии, которые используют расширенные функции этой библиотеки. Следующим шагом будет добавление второго набора тегов элемента `script` и небольшого фрагмента jQuery-кода между ними:

```
<script src="js/jquery-1.11.0.min.js"></script>
<script>
$(document).ready(function() {
    //программный код
});
</script>
```

Второй набор тегов `<script>` включает фрагмент кода, который вы хотите добавить на определенную веб-страницу, однако, вам, наверное, интересно, что означает часть `$(document).ready()`. Функция `$(document).ready()` — это встроенная функция библиотеки jQuery, которая ожидает, пока HTML-код страницы полностью загрузится, прежде чем запустить ваш сценарий.

Зачем вам может понадобиться данная функция? Дело в том, что большая часть кодов на языке JavaScript пишется с целью манипулирования содержимым веб-страницы: например, проявление изображения, выпадение меню при наведении указателя мыши на ссылку и т. д. Чтобы сделать что-нибудь интересное и интерактивное с элементом страницы, необходимо выбрать его. Тем не менее JavaScript не может выбрать HTML-тег, пока веб-браузер не загрузит его. Так как веб-браузер запускает код JavaScript непосредственно после его обнаружения, остальная часть страницы не загружается еще некоторое время. (Вы наблюдали этот эффект в руководстве по созданию викторины из предыдущей главы. При загрузке викторины страница пуста. Содержимое страницы появляется только после того, как вы закончите отвечать на вопросы — это потому, что сначала выполняется JavaScript-код викторины, а затем веб-браузер отображает HTML-теги.)

Другими словами, для того, чтобы делать интересные вещи с HTML-кодом на странице, нужно подождать пока она загрузится. Вот, зачем нужна функция `$(document).ready()`: она просто ждет, пока загрузится HTML-код, а затем выполняет код JavaScript. Если все это кажется вам очень запутанным, просто имейте в виду, что при помещении кода JavaScript в раздел заголовка страницы вы всегда должны включать функцию `.ready()` и то, что вам необходимо помещать свой код между строкой `$(document).ready(function() {` и последней комбинацией символов `});`.

В дополнение к этому, вот, что вам следует иметь в виду:

- Ссылка на файл jQuery должна предшествовать любому коду, который использует библиотеку. Иначе говоря, не размещайте другие элементы `script` перед элементом `script`, который загружает библиотеку jQuery.
- Помещайте ваш JavaScript-код *после* любых таблиц стилей CSS (как связанных внешних, так и внутренних). Поскольку jQuery-программирование часто ссылается на стили CSS, вам следует поместить свой JavaScript-код после кода, загружающего стили. Хорошим эмпирическим правилом является помещение вашего JavaScript-кода (все элементы `script`) после любого другого контента в разделе заголовка, но до закрывающего тега `</head>`.
- Добавьте комментарий JavaScript, например, `//окончание ready`, после сочетания символов `});`, которое отмечает окончание функции `ready()`. Например:

```
$(document).ready(function() {
      // здесь находится ваш код
}); // окончание ready
```

Добавление комментария после окончания функции позволяет легко определить конец программы. Как вы увидите позже, библиотека jQuery часто требует использования сочетания трех символов: фигурной скобки, круглой скобки и точки с запятой. Добавляя комментарий после них, будет намного легче определить, к какой части вашей программы относится конкретная группа знаков препинания.

СОВЕТ

jQuery обеспечивает упрощенный способ записи функции

```
$(document).ready(function() { }:
        $(function() {
              // здесь находится ваш код
        }); // окончание ready
```

Модифицирование веб-страниц

Язык JavaScript позволяет вам изменять веб-страницы прямо у вас на глазах. Используя JavaScript, вы можете моментально добавлять изображения и текст, скрывать контент или изменять вид элемента на странице. На самом деле, динамическое изменение веб-страницы — это отличительная черта нового поколения веб-сайтов, созданных с помощью языка JavaScript. Например, сервис Google Maps (**maps.google.com**) предоставляет доступ к карте мира; по мере изменения масштаба страница обновляется без необходимости загрузки новой веб-страницы. Похожим образом при наведении указателя мыши на название программы на сайте сервиса Now (**www.now.ru**) появляется всплывающее «облачко», содержащее ее детальное описание (см. рис. 4.1). В обоих приведенных примерах язык JavaScript изменяет HTML-код, изначально загруженный браузером.

В данной главе вы узнаете, как видоизменять веб-страницы, используя язык JavaScript. Вы будете добавлять новое содержимое, элементы и атрибуты HTML, а также корректировать уже существующие элементы. Другими словами, вы научитесь использовать язык JavaScript для генерирования нового и модификации уже имеющегося на странице HTML-кода.

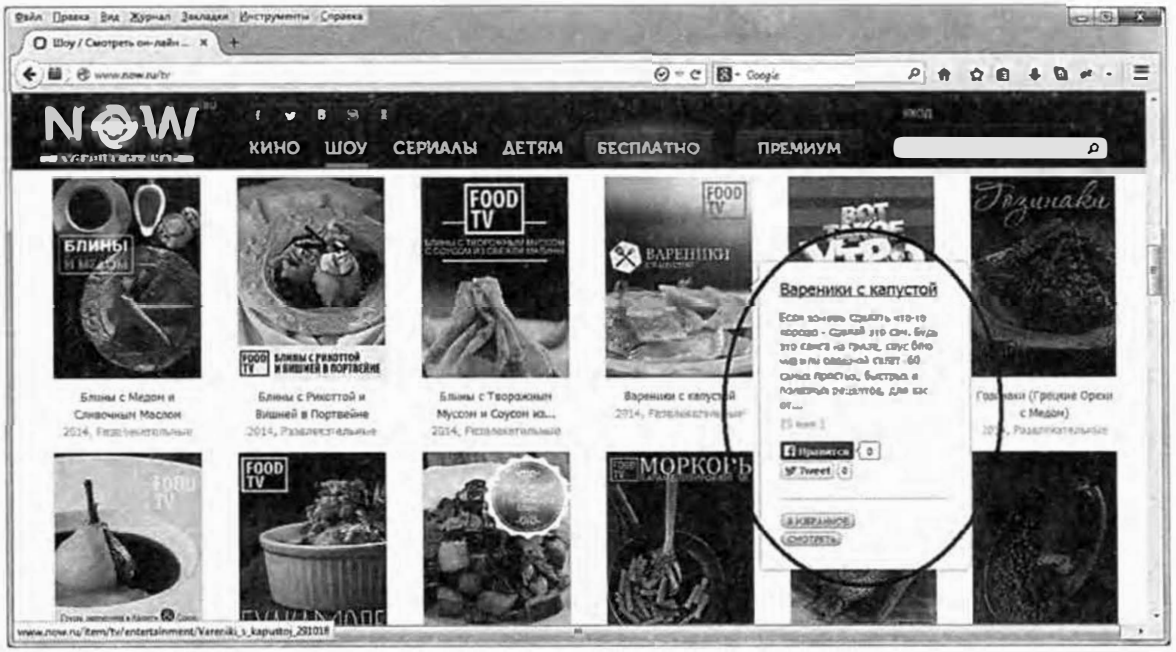

Рис. 4.2. Язык JavaScript может упростить чтение и просмотр веб-страниц, показывая контент только тогда, когда он нужен. На сайте Now.ru описания фильмов и программ скрыты и отображаются, когда посетитель наводит указатель мыши на их заголовок или эскиз

Сложно поверить, но если вы умеете создавать веб-страницы, используя HTML и CSS, то вы знаете уже многое из того, что необходимо знать для эффективного использования языка JavaScript при создании по-настоящему интерактивных веб-сайтов. Например, популярный плагин Datepicker для проекта jQuery UI облегчает посетителям процесс выбора даты при заполнении формы (при планировании перелета или события). Когда посетитель щелкает кнопкой мыши по специально отмеченному текстовому полю, появляется календарь (рис. 4.2). Несмотря на зрелищный эффект и удобство календаря, язык JavaScript отвечает только за интерактивность — сам календарь создан с помощью старых добрых HTML и CSS, с которыми вы уже знакомы.

Если вы загляните «под капот» календаря, вы увидите серии HTML-элементов, таких как `div`, `table` и `td` со специальными классами и идентификаторами (`ui-datepicker-month`, `ui-datepicker-div` и т. д.), примененными к ним. Таблица со стилями класса и идентификатора добавляет цвет, оформление и форматирование. Другими словами, вы могли бы сами создать такой же календарь, используя HTML и CSS. Язык JavaScript просто делает его интерактивным, заставляя календарь появляться, когда посетитель щелкает по элементу формы и исчезать после выбора даты.

Рис. 4.3. Проект jQuery UI (**jqueryui.com**) предоставляет удобные виджеты для веб-приложений. Например, виджет Datepicker предоставляет простой, дружественный пользователю способ выбора даты. Подробнее об этом вы узнаете в главе 10

Таким образом, современное JavaScript-программирование, особенно применительно к дизайну пользовательского интерфейса, следует рассматривать в качестве способа автоматизации создания HTML и применения CSS. В примере с ресурсом Now.ru (см. рис. 4.2) язык JavaScript заставляет появляться всплывающее «облачко» при наведении указателя мыши на эскиз или название фильма, однако самое интересное (дизайн этого «облачка») — это просто результат использования HTML и CSS — то, что вы уже умеете!

В основном вы будете использовать язык JavaScript для манипулирования веб-страницей, добавляя новый контент, изменяя HTML-код страницы или применяя CSS к элементу. Когда бы вы ни меняли содержимое, HTML-код или CSS на странице (добавляя панель навигации с всплывающими меню, создавая слайд-шоу на основе JavaScript или просто заставляя страницу проявляться (как в руководстве в главе 1)), вы выполняете два основных действия.

1. **Выбор элемента на странице.**

Элемент — это любой существующий тег, и прежде чем что-либо с ним сделать, вы должны *выбрать* его, используя JavaScript (что вы и научитесь делать в этой главе). Например, чтобы заставить страницу проявиться, вы сначала должны выбрать контент страницы (элемент

body); чтобы заставить появиться всплывающее меню при наведении указателя мыши на кнопку, нужно выбрать кнопку. Даже если вы хотите использовать язык JavaScript только для того, чтобы поместить текст в нижней части веб-страницы, вы должны выбрать элемент, чтобы вставить текст внутрь, до или после него.

2. **Выполнение действия над элементом.**

Ладно, «выполнить действие» — это не самая точная формулировка. Дело в том, что для изменения вида или поведения веб-страницы существует практически бесконечное количество действий. Цель этой книги — научить вас совершать различные действия над элементами веб-страницы. Вот несколько примеров:

- **Изменение свойства элемента.** При анимации элемента div вы изменяете положение соответствующего элемента на странице.
- **Добавление нового содержимого.** Если посетитель неправильно заполнил элемент веб-формы, обычно появляется сообщение об ошибке, например: «Пожалуйста, введите адрес электронной почты». В данном случае вы добавляете содержимое там, где необходимо заполнить соответствующий элемент.
- **Удаление элемента.** В примере с ресурсом Now.ru, показанном на рис. 4.2, всплывающее «облачко» исчезает, если убрать указатель мыши с названия фильма. В данном случае его удаляет JavaScript.
- **Извлечение информации об элементе.** Случается, что вам необходимо узнать больше о выбранном элементе. Например, для проверки текстового поля вы должны выбрать его, затем выяснить, что за текст был в него введен, другими словами, узнать значение этого поля.
- **Добавление и удаление атрибута класса.** Иногда вам нужно, чтобы элемент на странице изменял свой вид, например, чтобы цвет текста абзаца становился синим или цвет фона текстового поля становился красным, указывая на ошибку. Хотя язык JavaScript может произвести данные изменения, часто проще всего применить класс и позволить браузеру изменить вид элемента на основании стиля CSS. Например, чтобы изменить цвет текста абзаца на синий, вы можете создать стиль класса с синим цветом текста и использовать JavaScript для динамического применения этого класса к абзацу.

Часто вы будете одновременно производить несколько действий, описанных выше. Например, вы хотите удостовериться, что посетитель

не забыл ввести свой адрес электронной почты в поле формы. Если он попытается отправить форму без указания адреса, вы можете уведомить его об этом. Эта задача может потребовать выяснения того, ввел ли посетитель в текстовое поле хоть что-нибудь (извлечение информации об элементе), вывода сообщения об ошибке (добавление нового содержимого), если посетитель ничего не ввел и подсвечивания элемента формы (путем назначения класса текстовому полю).

Выбор элемента страницы — это первый шаг. Чтобы понять, как идентифицировать и изменять часть страницы, используя язык JavaScript, вы должны познакомиться с объектной моделью документа.

Объектная модель документа (DOM)

Когда браузер загружает HTML-файл, он отображает содержимое этого документа на экране (конечно, должным образом оформленное с использованием каскадных таблиц стилей). Но это не все, что браузер делает с элементами, атрибутами и содержимым файла, он также создает и запоминает «модель» HTML этой страницы. Другими словами, браузер запоминает HTML-элементы, их атрибуты и порядок, в котором они появляются в файле. Такое представление страницы называется *объектной моделью документа* (DOM, Document Object Model).

DOM предоставляет информацию, необходимую JavaScript для коммуникации с элементами на веб-странице. Объектная модель документа также предоставляет инструменты, необходимые для навигации по веб-странице, изменения и добавления на страницу нового HTML-кода. Сама по себе объектная модель документов не является частью JavaScript — это стандарт консорциума W3C, к которому производители большинства браузеров привели свои программы. Объектная модель документов позволяет JavaScript обмениваться информацией с веб-страницей и изменять ее HTML-код.

Чтобы понять принцип работы объектной модели документов, давайте рассмотрим пример очень простой веб-страницы:

```
<!DOCTYPE HTML>
<html>
<head>
      <meta charset="UTF-8">
      <title>Веб-страница</title>
```

```
</head>
<body class="home">
<h1 id="header">Заголовок</h1>
<p>Какой-нибудь <strong>важный</strong> текст</p>
</body>
</html>
```

На этом и на всех остальных сайтах одни элементы заключены в другие, например, элемент `html` включает все остальные, элемент `body` — элементы и контент, отображаемые в окне браузера. Вы можете представить отношения между элементами в виде модели, напоминающей генеалогическое древо (рис. 4.4). Элемент `html` — это корень дерева (прапрапрадедушка всех элементов страницы), остальные элементы — это ветви, например, `head` и `body`, каждый из которых содержит собственный набор элементов.

Браузер также отслеживает текст, появляющийся в элементе (например, «Заголовок» в элементе `h1` на рис. 4.4), а также *атрибуты*, присвоенные каждому элементу (на рис. 4.4 атрибут класса присваивается тегу `<body>`, а атрибут идентификатора — тегу `<h1>`). На деле объектная модель документов считает элементы (также называемые *тегами*), атрибуты и текст отдельными единицами — *узлами*.

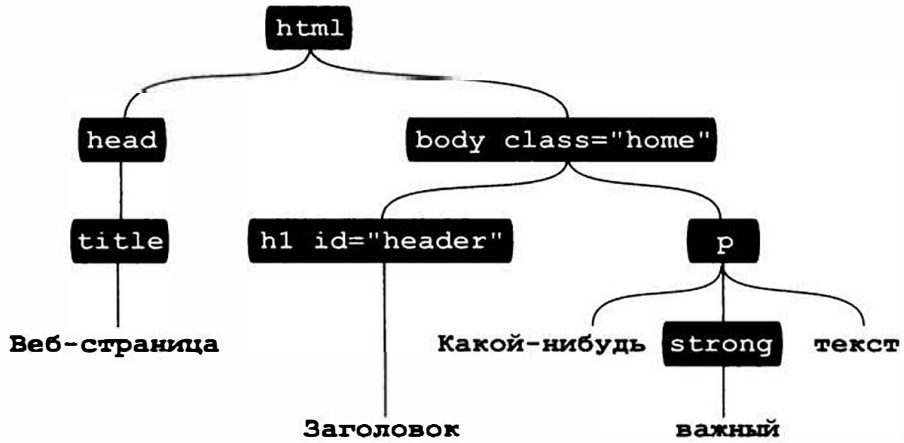

Рис. 4.4. Простейшая структура HTML-страницы часто изображается в виде генеалогического древа, где одни элементы включают другие и называются «родительскими», а вложенные элементы — «дочерними»

Язык JavaScript предоставляет несколько способов выбора элементов на странице, вы можете сделать с ними что-нибудь, например, заставить их исчезнуть из поля зрения или анимировать их перемещение по странице. Метод `document.getElementById()` позволяет выбрать

элемент с определенным идентификатором, примененным к его HTML-коду. Так что если у вас есть элемент `div` с примененным к нему идентификатором: `<div id="banner">`, то вы можете выбрать этот элемент `div` следующим образом:

```
document.getElementById('banner');
```

Подобным образом, метод `document.getElementsByTagName()` выбирает каждый экземпляр конкретного элемента (например, `document.getElementsByTagName('a')`, выбирает все элементы привязки (ссылки) на странице); а некоторые браузеры поддерживают методы выбора всех элементов с определенным классом или с использованием CSS-селектора для выбора элементов страницы.

Последние версии браузеров предусматривают способ выбора элементов DOM на основе селекторов CSS. Например, код `document.getElementsByClassName()` извлекает все элементы с одним именем класса. Чтобы выбрать все элементы с классом `author`, вы можете использовать следующий код:

```
document.getElementsByClassName('author');
```

Более понятной функцией является метод `querySelectorAll()`, который позволяет использовать любой селектор CSS для выбора элементов страницы. Например, чтобы выбрать только элементы `span` с классом `author`, вы могли бы написать следующее:

```
document.querySelectorAll('span.author');
```

Как вы увидите в следующем разделе, библиотека jQuery использует селекторы CSS для выбора элементов HTML на странице, причем этот подход поддерживается большинством браузеров.

Выбор элементов страницы: подход jQuery

Библиотека jQuery предлагает очень мощную технику выбора и работы с коллекциями элементов — селекторы CSS. Если вы привыкли к использованию каскадных таблиц стилей при оформлении веб-страниц, то готовы к использованию библиотеки jQuery. *Селектор CSS* — это просто инструкция, которая сообщает браузеру, к какому элементу применяется данный стиль. Например, `h1` — это селектор, определяющий стиль для каждого элемента `h1`; `.copyright`, в свою очередь, — это се-

лектор класса, оформляющий любой элемент, имеющий атрибут класса `copyright`:

```
<p class="copyright">Copyright, 2015</p>
```

С помощью библиотеки jQuery вы выбираете один и более элементов, используя специальную команду *объект jQuery*. Основной синтаксис таков:

```
$('селектор')
```

Вы можете использовать практически все селекторы CSS 2.1 и многие селекторы CSS 3, создавая объект jQuery (даже если сам браузер не понимает отдельных селекторов, например, как программа Internet Explorer не понимает некоторых селекторов CSS 3). Допустим, вы хотите выбрать элемент со специфическим идентификатором `banner` в jQuery. Напишите следующее:

```
$('#banner')
```

`#banner` — это селектор CSS, используемый для оформления элемента с идентификатором `banner`; символ `#` показывает, что вы определяете идентификатор. Конечно, если вы выбираете один или более элементов, то хотите что-нибудь с ними сделать, — и библиотека jQuery предлагает много инструментов для работы с элементами. Допустим, вы захотели изменить HTML-код внутри элемента. Напишите следующее:

```
$('#banner').html('<h1>JavaScript был тут!</h1>');
```

Подробнее работу с элементами страницы с помощью jQuery вы будете изучать, начиная с раздела «Добавление содержимого на страницу» и до конца книги. Однако сначала вы должны подробнее узнать о том, как использовать библиотеку jQuery для выбора элементов страницы.

Основные селекторы

Основные *селекторы* CSS, такие как селекторы идентификаторов, классов и элементов, образуют ядро CSS и являются прекрасным способом выбора широкого спектра элементов с использованием библиотеки jQuery.

Поскольку чтение о селекторах не лучший способ их понять, к этой книге прилагается интерактивная веб-страница для тестирования се-

лекторов. В каталоге *эксперименты* вы найдете файл *селекторы.html*. Откройте его в браузере. Испробуйте различные селекторы библиотеки jQuery, введя их в окно селекторов и щелкнув по кнопке **ОК** (рис. 4.5).

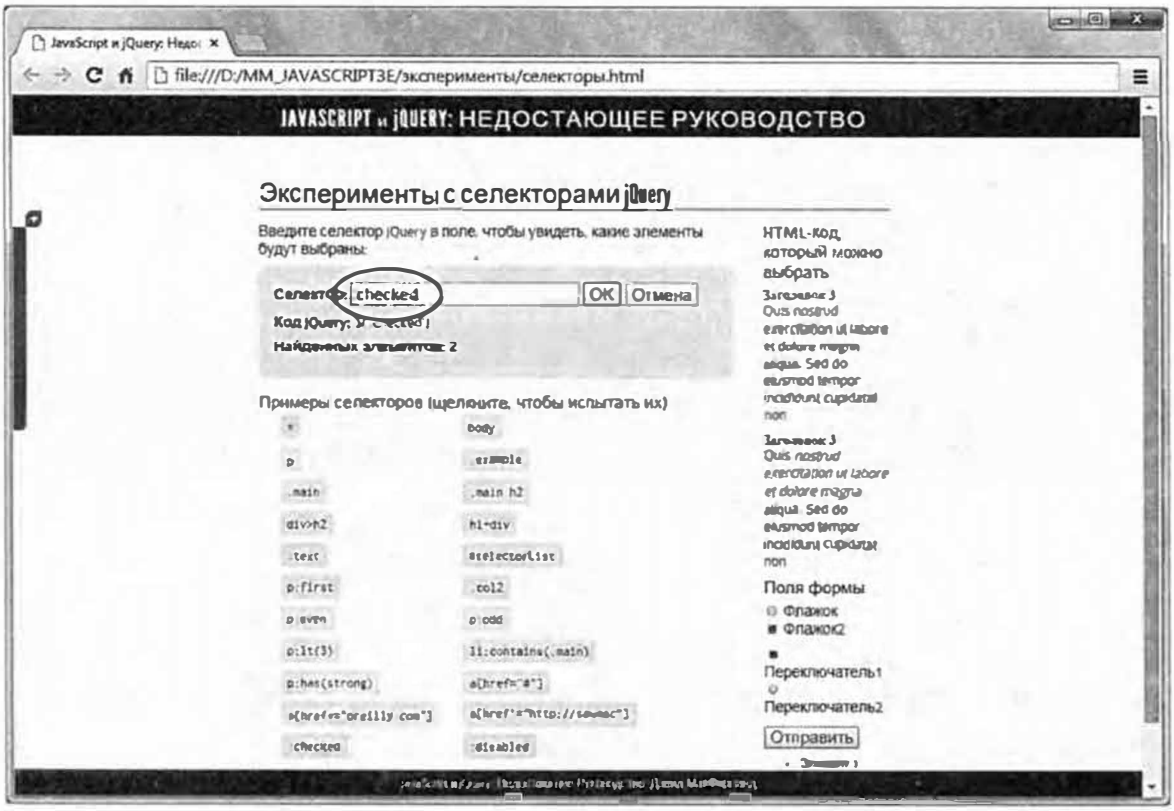

Рис. 4.5. Файл *селекторы.html*, находящийся в папке *эксперименты*, позволяет вам испытать селекторы jQuery. Просто введите селектор в поле **Селектор** формы (выделено на рисунке), а затем нажмите кнопку **ОК**. Страница превратит ваш селектор в объект jQuery, а каждый элемент страницы, соответствующий введенному селектору, окрасится красным цветом. Под полем **Селектор** отображается код, используемый для выбора элементов, а также общее число выделенных элементов. В приведенном примере `:checked` — это селектор, а выделенный переключатель и установленный флажок на странице (вы можете увидеть два таких элемента внизу справа) выделены красным

ПРИМЕЧАНИЕ

Файлы примеров находятся в папке, загруженной с сайта издательства.

Селекторы идентификаторов

Вы можете выбрать любой элемент страницы, к которому применен идентификатор, используя библиотеку jQuery и CSS-селектор идентификатора. Например, на веб-странице присутствует следующий HTML-код:

```
<p id="message">Специальное сообщение</p>
```

Для выбора элемента с использованием jQuery запишите следующий код:

```
var messagePara = $('#message');
```

В отличие от метода объектной модели документа, вы не просто используете имя идентификатора (`'message'`), вы должны применить синтаксис CSS для определения селектора идентификатора (`'#message'`).

Другими словами, вы ставите символ решетки перед именем идентификатора, как будто создавая стиль CSS для этого идентификатора.

Селекторы элементов

Для метода `getElementsByTagName()` библиотека jQuery также имеет замену. Просто передайте имя элемента библиотеке jQuery. Например, используя старый метод объектной модели документа для выбора каждого элемента `a` на странице, вы написали бы следующий код:

```
var linksList = document.getElementsByTagName('a');
```

Используя библиотеку jQuery, вы бы написали так:

```
var linksList = $('a');
```

ПРИМЕЧАНИЕ

Библиотека jQuery поддерживает более широкий набор селекторов, чем перечислено здесь. Хотя в этой книге и названы многие полезные селекторы, полный их список находится по адресу: **api.jquery.com/category/selectors/**.

Селекторы классов

Другим удобным способом выбора элементов является выбор по имени класса. Предположим, вы хотите создать навигационную панель, включающую выпадающие меню; когда посетитель наводит указатель мыши на основные навигационные кнопки, должно появляться выпадающее меню. Вам нужно, чтобы JavaScript контролировал эти меню и вам необходим способ, позволяющий запрограммировать каждую из основных навигационных кнопок на появление выпадающего меню при наведении на нее указателя мыши.

ПРИМЕЧАНИЕ

Поскольку нахождение всех элементов отдельно взятого класса — это обычная задача, в новейшие версии большинства браузеров добавлена такая возможность. Но так как не все браузеры имеют встроенную функцию нахождения элементов определенного класса (Internet Explorer8 и более ранние версии), такая библиотека, как jQuery, которая учитывает особенности всех браузеров, просто бесценна.

Одна из техник — добавление класса, например, `navButton`, к каждой из основных ссылок навигационной панели, а затем использование JavaScript для поиска ссылок, имеющих *именно это* имя класса, и применение «волшебной силы открытия меню» к этим ссылкам. Эта схема может показаться непонятной, но на данный момент важно следующее: чтобы эта навигационная панель работала, нужен способ выбора ссылок только с конкретным именем класса.

К счастью, библиотека jQuery предлагает простой метод выбора всех элементов с определенным именем класса. Просто используйте селектор класса CSS следующим образом:

```
$('.submenu')
```

Обратите внимание, что, как и в случае с селекторами класса CSS, перед именем селектора класса нужно ставить точку. Единожды выбрав элементы нужного класса, вы можете манипулировать ими, используя jQuery. Например, чтобы скрыть все элементы с именем класса `.submenu`, напишите следующее:

```
$('.submenu').hide();
```

Подробнее о jQuery-функции `hide()` вы узнаете в разделе «Эффекты jQuery» главы 6. А пока данный пример просто дает вам представление о принципе работы библиотеки jQuery.

УСКОРЯЕМ РАБОТУ

Понимание CSS

Каскадные таблицы стилей — это слишком большая тема для обсуждения в рамках книги о языке JavaScript. Чтобы получить от этой книги максимальную пользу, у вас должны быть базовые знания о вебдизайне, CSS и о том, как его использовать. CSS — это наиболее

важный инструмент веб-дизайнера, предназначенный для создания красивых сайтов, поэтому, если вы знаете о них недостаточно, пришло время разобраться в этой теме. CSS не только поможет вам пользоваться библиотекой jQuery, вы обнаружите, что можете использовать JavaScript в сочетании с CSS для того, чтобы легко добавлять на страницу интерактивные визуальные эффекты.

Если вы хотите быстро изучить CSS, то к вашим услугам есть множество ресурсов:

Ознакомьтесь с HTML Dog CSS Tutorials (**www.htmldog.com/guides/css/**). На этом сайте вы найдете руководства базового, среднего и продвинутого уровня.

Вы также можете воспользоваться «Большой книгой CSS3» Дэвида МакФарланда, полностью раскрывающей тему каскадных таблиц стилей (включая многочисленные практические руководства, как в этом пособии).

При работе с библиотекой jQuery наиболее важно понимать селекторы CSS — инструкции, сообщающие браузеру, к какому элементу применяется CSS. Чтобы научиться работать с селекторами, подойдут ресурсы, перечисленные здесь. Если вам нужно просто освежить свои знания о различных селекторах, посетите следующие ресурсы:

- **css.maxdesign.com.au/selectutorial/**
- **developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors/**

Специальные селекторы

Библиотека jQuery также предлагает более сложные селекторы CSS для точного указания на нужные элементы. Не волнуйтесь о том, что ими придется овладеть прямо сейчас: когда вы прочитаете еще несколько глав и лучше поймете, как работает библиотека jQuery и как использовать ее для манипулирования веб-страницами, то, возможно, захотите вернуться к этому разделу и перечитать его еще раз.

- **Селекторы потомков** позволяют указывать на элемент, заключенный внутри другого элемента. Например, вы создали неупорядоченный список ссылок и присвоили идентификатор `navBar` элементу `ul` данного списка: `<ul id="navBar">`. Выражение jQuery `$('a')` выбирает на странице все элементы `a`. Однако если вы хотите выбрать только ссылки неупорядоченного списка, то нужно использовать селектор вложенности следующим образом:

```
$('#navBar a')
```

Опять же синтаксис является базовым для CSS: селектор, пробел, следующий селектор. Селектор, указанный в списке последним, — это цель (в данном случае a), в то время как каждый селектор слева представляет элемент, окружающий эту цель.

- **Дочерние селекторы** указывает на элемент, также являющийся дочерним (прямым «потомком» другого элемента). Например, в HTML-коде, изображенном на рис. 4.4, элементы h1 и p являются дочерними по отношению к элементу body, но элемент strong не является таковым (поскольку он заключен в элемент p). Для создания дочернего селектора вы сначала называете родительский элемент, за которым следует угловая скобка >, а затем — дочерний. Например, для выбора элементов p, которые являются дочерними по отношению к элементу body, напишите следующее:

```
$('body > p')
```

- **Соседние родственные селекторы** позволяют выбрать элемент, появляющийся сразу после другого элемента. Например, у вас есть невидимая панель, которая появляется только после щелчка кнопкой мыши по вкладке. В вашем HTML-коде вкладка может быть представлена элементом заголовка (например, h2), а скрытая панель — элементом div, который следует за заголовком. Чтобы сделать элемент div (то есть панель) видимым, вы должны его выбрать. Вы можете легко сделать это с помощью библиотеки jQuery и дополнительного соседнего родственного селектора:

```
$('h2 + div')
```

Чтобы создать соседний родственный селектор, просто добавьте символ + между двумя селекторами (они могут быть любого типа: идентификаторами, классами или элементами). Селектор справа будет выбран, но только если он идет сразу после селектора, указанного слева.

- **Селекторы атрибута** позволяют выбирать элементы в зависимости от того, есть ли у селектора определенный атрибут, и даже удостовериться в том, что атрибут соответствует конкретному значению. С помощью селектора атрибута вы можете находить элементы img, имеющие набор атрибутов alt, или даже определенное текстовое значение alt. Также вы можете найти все ссылки, ведущие за пределы вашего сайта, и добавить к ним код, чтобы они открывались в новом окне.

Вы добавляете селектор атрибута после имени элемента, чей атрибут вы проверяете. Например, чтобы найти элементы img, имеющие набор атрибутов alt, вы должны записать следующее:

```
$('img[alt]')
```

Рассмотрим различные селекторы атрибута:

- `[attribute]` выбирает элементы, имеющие указанные в HTML-коде атрибуты. Например, `$(a[href])` находит все элементы `a`, имеющие набор атрибутов `href`. Выбирая по атрибуту `href`, вы исключаете именованные элементы привязки (`<a name="placeOnPage"></a>`), используемые просто как ссылки в рамках страницы;
- `[attribute="value"]` выбирает элементы, имеющие определенный атрибут с конкретным значением. Например, чтобы найти все окна с текстом в форме, вы можете использовать следующий код:

```
$('input[type="text"]')
```

Поскольку большинство элементов форм имеет один и тот же элемент (`input`), единственный способ определить тип элемента формы — это проверить тип его атрибута (выбор элементов форм — это настолько обычная процедура, что библиотека jQuery включает специальные селекторы именно для этой цели, см. раздел «Структура форм» главы 8);

- `[attribute^="value"]` находит элементы с атрибутом, который *начинается* с определенного значения. Например, если вы хотите найти ссылки, ведущие за пределы вашего сайта, используйте такой код:

```
$('a[href^="http://"]')
```

Обратите внимание, что из всего значения атрибута должно совпадать только самое начало. Так, `href^=http://` находит ссылки, ведущие на сайты http://www.yahoo.com, http://www.google.com и т. д. Этот селектор можно использовать для идентификации ссылок `mailto`:

```
$('a[href^="mailto:"]')
```

- `[attribute$="value"]` выбирает элементы, чьи атрибуты заканчиваются определенным значением, что великолепно подходит для поиска файловых расширений. Например, с помощью данного селектора вы можете найти ссылки, указывающие на файлы PDF (возможно, чтобы использовать язык JavaScript для добавления специального символа PDF или динамического генерирования ссылки на сайт Adobe.com, чтобы ваш посетитель смог загрузить программу Acrobat Reader). Код для выбора ссылок, указывающих на PDF-файлы, выглядит так:

```
$('a[href$=".pdf"]')
```

- `[attribute*="value"]` находит элементы с атрибутом, в любом месте которого содержится конкретное значение. Например, вы можете найти любые ссылки, указывающие на отдельно взятый домен. Данный код помогает найти ссылку, ведущую на сайт **eksmo.ru**:

  ```
  $('a[href*="eksmo.ru"]')
  ```

  Данный селектор гибок в использовании и позволяет находить ссылки, указывающие не только на *http://www.eksmo.ru*, но и на *http://eksmo.ru* и *www.eksmo.ru/library.html*.

ПРИМЕЧАНИЕ

Библиотека jQuery имеет набор селекторов, полезных при работе с формами. Эти селекторы позволяют легко выбирать элементы форм, например, текстовые поля, поля паролей и переключатели. Об этих селекторах вы узнаете в разделе «Выбор элементов формы» главы 8.

Фильтры jQuery

Библиотека jQuery позволяет фильтровать выборки, основываясь на определенных характеристиках. Например, с помощью фильтра `:even` несложно выбрать все четные элементы коллекции. Вы также можете находить элементы, содержащие определенные элементы, текст, скрытые элементы и даже элементы, которые *не* соответствуют данному селектору. Чтобы использовать фильтр, добавьте двоеточие после основного селектора, а за ним введите имя фильтра. Например, чтобы найти все четные строки в таблице, можно написать селектор jQuery следующим образом:

```
$('tr:even')
```

Данный код выбирает все четные элементы `tr`. Чтобы сузить выборку, вы можете найти все четные строки в таблице с именем класса `striped`:

```
$('.striped tr:even')
```

`:even` и остальные фильтры работают следующим образом:

- Фильтры `:even` и `:odd` выбирают каждый *второй* элемент в группе. Эти фильтры работают несколько странно; просто помните, что выборка jQuery — это список всех элементов, соответствующих указан-

ному селектору. Учитывая это, они напоминают массивы (см. раздел «Массивы» главы 2). Каждый элемент выборки jQuery имеет индекс, а как вы помните, индексация элементов массива всегда начинается с 0 (см. раздел «Доступ к элементам массива» главы 2). Итак, поскольку фильтры *:even* фильтруют по четным значениям индекса (например, 0, 2 или 4), этот фильтр возвращает в данном случае первый, третий и пятый (и т. д.) элементы выборки, другими словами, он в действительности выбирает каждый нечетный элемент! Фильтр `:odd` работает так же, за исключением того, что он выбирает каждый нечетный индексный номер (1, 3, 5 и т. д.).

- Фильтры `:first` и `:last` выбирают первый и последний элемент группы. Например, если вы хотите выбрать первый абзац на странице, введите следующее:

```
$('p:first');
```

А для того, чтобы выбрать последний абзац на странице, наберите следующее:

```
$('p:last');
```

- Фильтр `:not()` находит элементы, не соответствующие данному типу селектора. Например, вы хотите выбрать все элементы `a`, кроме тех, которые относятся к классу `navButton`:

```
$('a:not(.navButton)');
```

Вы сообщаете фильтру `:not()` имя селектора, который следует игнорировать. В данном случае `.navButton` — это селектор класса, то есть данный код переводится как «без класса `.navButton`». Разрешается использовать фильтр `:not()` с любыми фильтрами и с большинством селекторов библиотеки jQuery. Например, чтобы найти все ссылки, начинающиеся не с `http://`, вы можете написать следующий код:

```
$('a:not([href^="http://"])')
```

- Фильтр `:has()` находит элементы, содержащие другой селектор. Например, вы хотите найти все элементы `li`, но только если внутри них есть элемент `a`:

```
$('li:has(a)')
```

Данный фильтр несколько отличается от селекторов потомков, так как выбираются элементы не `a`; а `li`, причем только те из них, которые содержат ссылку.

- Фильтр `:contains()` находит элементы, содержащие конкретный текст. Например, чтобы найти все ссылки, гласящие «Щелкни по мне!», вы можете создать следующий объект jQuery:

```
$('a:contains(Щелкни по мне!)')
```

- Фильтр `:hidden` находит скрытые элементы, к которым относятся элементы, обладающие свойством CSS `display`, установленным на значение `none` (это означает, что вы не видите их на странице); элементы, которые вы скрываете, используя jQuery-функцию `hide()` (описана в разделе «Основы отображения и сокрытия» главы 6); элементы, чьи высота и ширина равны нулю, а также скрытые элементы форм. (Данный селектор не применим к элементам, у которых CSS-свойству `visibility` присвоено значение `invisible`.) Допустим, вы скрыли несколько элементов `div`. Вы можете найти их и сделать видимыми, используя jQuery следующим образом:

```
$('div:hidden').show();
```

Данная строка кода не воздействует на элементы `div`, которые в настоящий момент видимы на странице. (О функции библиотеки jQuery `show()` вы узнаете подробнее в разделе «Основы отображения и сокрытия» главы 6.)

- Фильтр `:visible` противоположен фильтру `:hidden`. Он находит на странице видимые элементы.

Понимание выборок jQuery

Выбирая один или более элементов с использованием объекта jQuery, например, `$('#navBar a')`, вы не получаете в итоге традиционного списка узлов объектной модели документа, как при использовании команд `getElementById()` или `getElementsByTagName()`. Вместо этого вы получаете особую выборку элементов, применимую только в jQuery. Эти элементы «не понимают» традиционных методов объектной модели; поэтому если бы вы прочитали о методах объектной модели документа в другой книге, то обнаружили бы, что ни один из них не работает с объектом jQuery. Это может показаться серьезным недостатком, но почти у всех обычных методов и свойств объектной модели есть эквиваленты jQuery, поэтому вы можете делать практически все, что позволяет традиционная объектная модель документа, и даже быстрее и с меньшим количеством строк кода.

Однако между принципом работы DOM и выборок jQuery есть два концептуальных различия. Библиотека jQuery была создана, чтобы облегчить и ускорить процесс программирования на языке JavaScript. Одна из целей библиотеки — предоставить как можно больше функциональности в наименьшем объеме кода. Чтобы достичь этого, jQuery использует два необычных принципа: *автоматические циклы* и *связывание функций*.

Автоматические циклы

Как правило, когда вы используете объектную модель документа и выбираете несколько элементов страницы, вы должны затем создать цикл (см. раздел «Работа с повторяющимися задачами с использованием циклов» главы 3) и просмотреть каждый из выбранных узлов, а затем что-либо с ним сделать. Например, если вы хотите выбрать все изображения на странице, а затем скрыть их (это может быть полезно при создании слайд-шоу на основе JavaScript), то должны сначала выбрать изображения, а затем создать цикл для работы с каждым из них.

Поскольку проход по списку элементов с помощью цикла является обычной задачей, библиотека jQuery имеет эту встроенную возможность. Другими словами, применяя jQuery-функцию к выборке элементов, вам не обязательно создавать цикл самим, поскольку функция делает это автоматически.

Например, чтобы выбрать все изображения в элементе `div` с идентификатором `slideshow`, а затем скрыть их, нужно записать следующий jQuery-код:

```
$('#slideshow img').hide();
```

Создаваемый с помощью фрагмента `$('#slideshow img')` список элементов может включать 50 изображений. Функция `hide()` автоматически проходит через весь список, скрывая каждую из картинок. Данная возможность настолько удобна (представьте себе количество `for`-циклов, которые пришлось бы писать), что совершенно непонятно, почему она не является частью языка JavaScript.

Связывание функций

Иногда вам нужно будет произвести над выбранными элементами несколько операций. Допустим, вы хотите установить ширину и высоту

элемента `div` (с идентификатором `popUp`), используя язык JavaScript. Обычно вам пришлось бы писать как минимум две строки кода. Однако библиотека jQuery позволяет обойтись всего одной строкой:

```
$('#popUp').width(300).height(300);
```

Библиотека jQuery использует уникальный принцип, называемый *связыванием*, что позволяет вам добавлять функции одну за другой. Каждая функция соединена со следующей функцией точкой и применяется к той же коллекции элементов, что и предыдущая. Так, вышеприведенный код изменяет ширину элемента с идентификатором `popUp`, *а также* изменяет высоту этого элемента.

Связывание jQuery-функций позволяет вам выполнять большое количество действий. Например, вы не только хотите установить ширину и высоту элемента `div`, но и планируете добавить в него текст и заставить проявляться (если сейчас он невидим на странице). Вы можете сделать это таким образом:

```
$('#popUp').width(300).height(300).text('Привет!').↵
fadeIn(1000);
```

Этот код применяет четыре jQuery-функции: `width()`, `height()`, `text()` и `fadeIn()` к элементу с идентификатором `popUp`.

СОВЕТ

Длинная строка связанных jQuery-функций может быть сложна для восприятия, поэтому некоторые программисты разбивают ее на несколько строк:

```
$('#popUp').width(300)
.height(300)
.text('Сообщение')
.fadeIn(1000);
```

Если вы поставите точку с запятой в последней строке цепочки, интерпретатор JavaScript воспримет все строки как одну инструкцию.

Возможность связывания функций довольно необычна и является особой функцией библиотеки jQuery. Другими словами, вы не можете добавлять в цепочку функции, не являющиеся функциями jQuery (созданные вами или встроенные в язык JavaScript) без создания дополнительного кода.

Добавление содержимого на страницу

Библиотека jQuery предлагает много функций для манипулирования содержимым страницы: от простой замены HTML-кода до точного расположения нового HTML относительно выбранного элемента и полного удаления элементов и содержимого со страницы.

ПРИМЕЧАНИЕ

Файл *содержимое_функции.html*, расположенный в каталоге *эксперименты*, позволит вам опробовать каждую из этих jQuery-функций. Просто откройте файл в браузере, введите какой-нибудь текст в текстовое поле и щелкните по любой из функций, чтобы посмотреть, как она работает.

Рассмотрим некоторые из этих функций. Допустим, у вас есть страница со следующим HTML-кодом:

```
<div id="container">
      <div id="errors">
      <h2>Ошибки:</h2>
      </div>
</div>
```

Далее перечислены пять наиболее полезных jQuery-функций, предназначенных для манипулирования содержимым страницы.

- Функция `.html()` может считывать существующий HTML-код внутри элемента, а также замещать текущее содержимое другим HTML-кодом. Вы используете функцию `html()` вместе с выборкой jQuery.

  Для возвращения HTML-кода из выборки просто добавьте `.html()` после выборки jQuery. Например, вы можете запустить следующую команду, используя фрагмент HTML-кода в начале этого раздела:

  ```
  alert($('#errors').html());
  ```

Данный код создает окно оповещения с текстом "`<h2>Ошибки:</h2>`". Используя функцию `html()` таким образом, вы можете сделать копию HTML-кода, содержащегося внутри определенного элемента и вставить его в другой элемент на странице.

Если в качестве аргумента вы сообщите функции `.html()` строку, то замените текущее содержимое выборки:

```
$('#errors').html('<p>В данной форме четыре ошибки</p>');
```

Данная строка кода заменяет весь HTML-код внутри элемента с идентификатором `errors`. В итоге получится:

```
<div id="container">
   <div id="errors">
         <p>В данной форме четыре ошибки</p>
   </div>
</div>
```

Обратите внимание, что заменяется уже существующий элемент `h2`. Вы можете избежать замены этого HTML-кода, используя функции, перечисленные ниже.

ПРИМЕЧАНИЕ

Если вы используете функцию `html()` или `text()` для извлечения HTML-кода или текста из выборки jQuery, содержащей множество элементов, то HTML-код или текст будет извлечен только из первого элемента выборки. Например, если вы примените код `var divContents = $('div'). html();` к странице, на которой присутствуют 10 элементов `div`, то в переменной `divContents` будет храниться HTML-код только из первого элемента `div` на странице.

Тем не менее, при использовании функций `html()` или `text()` для вставки HTML-кода или текста в выборку jQuery, все выбранные элементы окажутся затронуты этой вставкой. Например, код `$('div'). html('<p>Привет</p>');` заменит HTML-код во всех элементах `div` на странице одним абзацем и словом «Привет».

- Работа функции `.text()` похожа на работу функции `.html()`, но не принимает HTML-элементы. Она полезна, если вам необходимо заменить текст, заключенный в элемент. Например, в коде в начале данного раздела вы видите элемент `h2` с заключенным в него текстом «Ошибки:». Допустим, после запуска программы для проверки формы на наличие ошибок вы захотели заменить текст «Ошибки:» на текст «Ни одной ошибки не найдено», в этом случае вы можете использовать следующий код:

```
$('#errors h2').text('Ни одной ошибки не найдено');
```

Элемент h2 остается как есть, изменяется только текст, заключенный в него. Библиотека jQuery кодирует любые HTML-элементы, которые вы передаете функции text(), поэтому элемент p превращается в сочетание символов <p>. Это может пригодиться, если вам необходимо отобразить элементы и угловые скобки *на* странице. Скажем, вы можете отобразить пример HTML-кода, который могут увидеть ваши посетители.

- Функция .append() добавляет HTML-код в качестве последнего дочернего элемента выбранного элемента. Допустим, вы выбрали элемент div, но вместо замены его содержимого вы просто хотите добавить фрагмент HTML-кода перед закрывающим тегом . Функция .append() — это отличный способ добавить элемент в конец маркированного списка (ul) или нумерованного списка (ol). Предположим, вы запускаете следующий код на странице с HTML, приведенным в начале этого раздела:

```
$('#errors').append('<p>В данной форме четыре ↵
ошибки</p>');
```

После выполнения этой функции вы получаете следующий HTML-код:

```
<div id="container">
   <div id="errors">
         <h2>Ошибки:</h2>
         <p>В данной форме четыре ошибки</p>
   </div>
</div>
```

Обратите внимание, что первоначальный HTML-код внутри элемента div остается прежним и после него добавляется новый фрагмент HTML.

- Функция .prepend() схожа с функцией append(), но добавляет HTML-код прямо после открывающего тега выборки. Например, вы применяете следующий код к HTML, приведенному выше:

```
$('#errors').prepend('<p>В данной форме четыре ↵
ошибки</p>');
```

После выполнения функции prepend() вы получаете следующий HTML:

```
<div id="container">
   <div id="errors">
         <p>В данной форме четыре ошибки</p>
         <h2>Ошибки:</h2>
   </div>
</div>
```

Теперь вновь добавленное содержимое появляется сразу после открывающего тега `<div>`.

- Если вы хотите добавить HTML-код вне выборки — или до открывающего тега выбранного элемента, или прямо после закрывающего тега элемента — используйте функции `before()` или `after()`. Например, принято проверять текстовое поле формы, чтобы убедиться, что посетитель не оставил его пустым. Предположим, что HTML-код этого поля до подтверждения выглядит так:

```
<input type="text" name="userName" id="userName">
```

Теперь допустим, что, когда посетитель отправляет форму, это поле пусто. Вы можете написать программу, проверяющую поле и добавляющую за ним сообщение об ошибке. Чтобы добавить сообщение после этого поля (не волнуйтесь о том, как проверить, правильно ли заполнены элементы формы, об этом вы прочитаете в разделе «Проверка формы» главы 8), вы можете использовать функцию `.after()` таким образом:

```
$('userName').after('<span class="error">Имя ↵
пользователя обязательно!</span>');
```

Данная строка кода выводит на веб-страницу сообщение об ошибке, HTML-код будет выглядеть так:

```
<input type="text" name="userName" id="userName">
<span class="error">Имя пользователя обязательно!</span>
```

Функция `.before()` просто помещает новое содержимое перед выбранным элементом. Поэтому строка:

```
$('#userName').before('<span class="error">Имя ↵
пользователя обязательно</span>');
```

Сгенерирует следующий HTML-код:

```
<span class="error">Имя пользователя обязательно</span>
<input type="text" name="userName" id="userName">
```

ПРИМЕЧАНИЕ

Функции, перечисленные в данном разделе (`html()`, `text()` и т. д.), — это наиболее распространенные способы добавления и изменения содержимого на странице, однако есть и другие. Вы можете найти другие функции на сайте **api.jquery.com/category/manipulation/**.

Замена и удаление выборок

Иногда вам может понадобиться полностью заменить выбранный элемент или переместить его. Например, вы создали всплывающее диалоговое окно, используя язык JavaScript (не при помощи старомодного метода `alert()`, а более профессионально оформленное диалоговое окно, являющееся абсолютным позиционированием элемента `div`, всплывающего в верхней части страницы). Когда посетитель щелкает по кнопке **Закрыть** (Close), вы, конечно же, хотите убрать диалоговое окно. Чтобы сделать это, воспользуйтесь jQuery-функцией `remove()`. Допустим, всплывающее диалоговое окно имеет идентификатор `popup`; чтобы его удалить, можно использовать следующий код:

```
$('#popup').remove();
```

Функция `.remove()` не ограничивается только одним элементом. Предположим, вы хотите удалить все элементы `span` с примененным к ним классом `error`. Вы можете сделать это следующим образом:

```
$('span.error').remove();
```

Вы также можете полностью заменить выборку новым содержимым. Предположим, у вас есть страница с фотографиями товаров, продаваемых вашей компанией. Когда посетитель щелкает кнопкой мыши по изображению товара, данный товар добавляется в покупательскую корзину. Вы можете заменить элемент `img` определенным текстом (например, «Добавлено в корзину»), который будет появляться после щелчка по картинке. Вы научитесь заставлять отдельно взятые элементы реагировать на события (например, на щелчок по изображению) в следующей главе, а сейчас предположим, что существует элемент `img` с идентификатором `product101`, который вы хотите заменить текстом. Вот как это делается с помощью библиотеки jQuery:

```
$('#product101').replaceWith('<p>Добавлено в ↵
корзину</p>');
```

Данный код удаляет элемент `img` со страницы и заменяет его элементом `p`.

ПРИМЕЧАНИЕ

Библиотека jQuery также включает функцию `clone()`, позволяющую копировать выбранный элемент. Вы увидите ее в действии в руководстве в конце данной главы.

Установка и чтение атрибутов элемента

Библиотека jQuery хороша не только в добавлении, удалении и изменении элементов. И это не все действия, которые вы захотите произвести, работая с выборкой элементов. Часто вам понадобится изменить значение атрибута элемента: добавить класс элемента или изменить CSS-свойство. Вы также можете получить значение атрибута, например, узнать, куда ведет отдельно взятая ссылка.

Классы

Каскадные таблицы стилей — это очень мощная технология, позволяющая осуществлять любое изощренное форматирование вашего HTML-кода. Одно правило CSS может добавить на страницу цветной фон, а другое — полностью скрыть элемент. Вы можете создать по-настоящему продвинутые визуальные эффекты, просто используя язык JavaScript для удаления, добавления или изменения класса, относящегося к элементу. Поскольку браузеры обрабатывают и осуществляют инструкции CSS очень быстро и эффективно, просто добавляя к элементу класс, вы можете полностью изменить представление элемента и даже заставить его исчезнуть со страницы.

ПОЛЕЗНЫЙ ИНСТРУМЕНТАРИЙ

Просмотр визуализированного HTML-кода

Одна из проблем использования языка JavaScript при работе с объектной моделью документа DOM, например, в ходе добавления, изменения, удаления и упорядочивания кода HTML, состоит в том, что сложно понять, как будет выглядеть HTML-код страни-

цы по окончании работы кода JavaScript. Например, команда **View Source** (Просмотреть исходный код), доступная в любом браузере, показывает веб-страницу такой, какой она была загружена с веб-сервера. Другими словами, вы видите HTML-код до того, как он был изменен JavaScript, что мешает вам выяснить, производит ли написанный вами код JavaScript нужный вам HTML-код. Например, если бы вы могли увидеть, как выглядит HTML-код после того, как JavaScript выведет 10 сообщений об ошибках на страницу формы или после того, как программа JavaScript создаст сложное всплывающее диалоговое окно с текстом и элементами формы, то было бы гораздо проще понять, получаете ли вы в итоге желаемый результат.

К счастью, большинство основных браузеров включает средства разработчика, которые позволяют вам просматривать *визуализированный HTML-код* — HTML-код, отображаемый браузером после того, как над ним поработал JavaScript. Обычно эти средства находятся в нижней части окна браузера под веб-страницей. Различные вкладки позволяют вам получить доступ к JavaScript-коду, HTML и CSS, а также к другим полезным ресурсам. Точное название вкладки и способ открытия панели инструментов различны в разных браузерах:

- В браузере Chrome выберите команду меню **Вид** ⇒ **Инструменты** ⇒ **Инструменты разработчика** (View ⇒ Developer ⇒ Developer Tools) и щелкните по вкладке **Элементы** (Elements) на панели в нижней части окна браузера.
- В браузере Firefox выберите команду меню **Инструменты** ⇒ **Веб-разработка** ⇒ **Инспектор** (Tools ⇒ Developer ⇒ Inspector). Это приведет к открытию панели в нижней части окна браузера, в которой будет отображаться HTML-код, на который влияет JavaScript.
- В браузере Internet Explorer нажмите клавишу **F12**, чтобы открыть панель **Средства разработчика** (Developer Tool), а затем щелкните по вкладке **HTML**. На вкладке **HTML** отображается HTML-код страницы (тот же, что и при использовании команды **Просмотр HTML-кода** (View Source)). Однако если вы щелкните по кнопке **Обновить** (Refresh) (клавиша **F5**), то во вкладке **HTML** отобразится визуализированный HTML-код, включающий изменения, созданные с помощью JavaScript.
- В браузере Safari удостоверьтесь, что меню **Разработка** (Developer) отображено (выберите команду меню **Safari** ⇒ **Настройки** (Safari ⇒ Preferences), щелкните по кнопке **Дополнения** (Advanced) и убедитесь, что флажок **Показывать меню «Разработка» в строке меню** (Show Develop menu in menu bar) установ-

лен). Затем откройте нужную страницу и выберите команду меню **Разработка ⇒ Показать веб-инспектор** (Developer ⇒ Show Web Inspector). Щелкните по вкладке **Элементы** (Elements) на панели, появившейся в нижней части окна браузера.

- В браузере Opera выберите команду меню **Инструменты ⇒ Дополнительно ⇒ Opera Dragonfly** (Tools ⇒ Advanced ⇒ Opera Dragonfly). (Dragonfly — это название встроенного в браузер инструмента разработчиков.) На панели, появившейся в нижней части окна браузера, щелкните по вкладке **Документы** (Documents).

Библиотека jQuery предлагает некоторые функции для манипулирования атрибутом класса элемента.

- Функция `addClass()` добавляет элементу конкретный класс. Вы вводите функцию `addClass()` после выборки jQuery и передаете функции строку, представляющую собой имя класса, который вы хотите добавить. Например, чтобы добавить класс `externalLink` всем ссылкам, ведущим за пределы вашего сайта, можете использовать следующий код:

```
$('a[href^=http://]').addClass('externalLink');
```

Этот фрагмент кода возьмет HTML:

```
<a href="http://www.eksmo.ru/">
```

и изменит его на следующий:

```
<a href="http://www.eksmo.ru/"
class="externalLink">
```

Чтобы получить пользу от данной функции, создайте стиль класса CSS заранее и добавьте его к таблице стиля страницы. Затем, когда JavaScript присоединит к элементу имя класса, браузер сможет применить к данному элементу свойства стиля заранее определенного правила CSS.

ПРИМЕЧАНИЕ

Применяя функции `addClass()` и `removeClass()`, вы только передаете им имя класса, опуская точку, которая обычно используется при создании селектора класса. Например, запись `addClass('externalLink')` правильна, а `addClass('.externalLink')` — нет.

Данная jQuery-функция также решает проблемы, возникающие, если элементу уже присвоен класс. Функция `addClass()` не исключает присвоенных классов, она просто добавляет новый.

ПРИМЕЧАНИЕ

Добавление нескольких имен классов одному элементу допустимо и часто очень удобно. Посетите ресурс **www.cvwdesign.com/txp/article/177/use-more-than-one-css-class**, чтобы найти более подробную информацию по этой технике.

- Функция `removeClass()` противоположна функции `addClass()`. Она удаляет указанный класс из выбранных элементов. Например, если вы хотите удалить класс `highlight` из элемента `div` с идентификатором `alertBox`, сделайте следующее:

```
$('#alertBox').removeClass('highlight');
```

- Наконец, вы сможете включать или отключать отдельные классы, то есть добавлять класс, если его еще нет, или удалять имеющийся класс. Переключение — распространенный способ показа элемента в действии или в бездействии. Например, щелкая по элементу интерфейса «переключатель», вы его включаете (on); когда вы щелкаете по нему снова, точка в его центре исчезает (off).

Допустим, на вашей веб-странице есть переключатель, который, будучи включенным, изменяет класс элемента `body`. Вы можете полностью изменить стиль веб-страницы путем создания второго набора стилей с использованием селекторов потомков. Когда вы щелкаете по переключателю снова, это говорит о намерении удалить класс из элемента `body`, чтобы страница была отображена в прежнем виде. Предположим, переключатель, изменяющий стиль страницы, имеет идентификатор `changeStyle`. Вы хотите, чтобы он переключал класс с именем `altStyle`, когда посетитель щелкает по нему. Напишите следующий код:

```
$('#changeStyle').click(function() {
    $('body').toggleClass('altStyle');
});
```

Не беспокойтесь о первой и третьей строках приведенного выше кода. Здесь мы имеем дело с событиями, которые позволяют сценариям реагировать на действия. Щелчок на переключателе — событие на стра-

нице. Вы узнаете о событиях в следующей главе. Текст, выделенной полужирным шрифтом, демонстрирует функцию `toggleClass()`, которая либо добавляет, либо удаляет класс `altStyle` при каждом щелчке.

Чтение и изменение свойств CSS

JQuery-функция `css()` позволяет изменять свойства CSS определенного элемента, то есть вместо того, чтобы применять к элементу стиль класса, вы можете непосредственно добавлять границу, цвет фона, устанавливать ширину или задавать расположение. Вы можете использовать функцию `css()` тремя способами: находить текущее значение свойства CSS определенного элемента, устанавливать свойство CSS определенного элемента или настраивать несколько свойств CSS одновременно.

Для определения текущего значения свойства CSS передайте имя свойства функции `css()`. Предположим, вы хотите узнать цвет фона элемента `div` с идентификатором `main`:

```
var bgColor = $('#main').css('background-color');
```

После выполнения данного кода переменная `bgColor` будет содержать строку со значением цвета фона элемента.

ПРИМЕЧАНИЕ

Библиотека jQuery не всегда возвращает значения CSS так, как вы ожидаете. В случае с цветами (вроде цвета фона или свойств цвета CSS) jQuery всегда возвращает либо значение `rgb`, например, `rgb(255, 0, 10)`, либо, если в цвете есть прозрачность, значение `rgba`, например, `rgba(255,10,10,.5)`. Библиотека jQuery возвращает значения RGB вне зависимости от того, был ли цвет в таблице стиля определен с использованием шестнадцатеричной системы (`#F4477A`), RGB с использованием процентов (`rgb(100%,10%,0%)`) или цветовой модели HSL (`hsl(72,100%,50%)`). Кроме того, библиотека jQuery переводит все значения в пикселы, поэтому, даже если вы используете CSS для установки размера шрифта элемента `body` на 150%, jQuery возвратит значение в пикселах при проверке свойства `font-size`.

Функция `css()` также позволяет присваивать элементу свойство CSS. Для этого вы должны передать функции два аргумента: имя свой-

ства CSS и значение. Например, чтобы увеличить размер шрифта для элемента body до 200%, вы можете сделать следующее:

```
$('body').css('font-size', '200%');
```

Второй присваиваемый аргумент может быть строкой, например, '200%', или числовым значением, которое jQuery переведет в пикселы. Например, чтобы изменить отступ в элементах класса .pullquote на 100 пикселов, можно написать следующий код:

```
$('.pullquote').css('padding',100);
```

В данном примере библиотека jQuery устанавливает свойство padding на 100 пикселов.

ПРИМЕЧАНИЕ

Устанавливая свойство CSS с помощью функции css(), вы можете использовать метод сокращений CSS. Рассмотрим, как можно добавить черную рамку толщиной в один пиксел вокруг всех абзацев класса highlight:

```
$('p.highlight').css('border', '1px solid black');
```

Часто полезно изменять свойства CSS, основываясь на текущем значении. Например, вы хотите добавить на веб-страницу кнопку «Увеличить текст». Когда посетитель нажимает ее, размер шрифта увеличивается в два раза. Для этого нужно прочитать значение, а затем установить новое значение. В данном случае вы сначала определяете текущий размер шрифта, а потом увеличиваете это значение в два раза. Это чуть более сложно, чем может показаться на первый взгляд. Далее следуют код и его объяснение:

```
var baseFont = $('body').css('font-size');
baseFont = parseInt(baseFont);
$('body').css('font-size',baseFont * 2);
```

В первой строке возвращается значение размера шрифта элемента body. Причем значение возвращается в пикселах и является строкой, например: '16px'. Если вы хотите удвоить размер (умножить на 2), вы должны преобразовать эту строку в число, удалив часть px. Во второй строчке кода используется метод JavaScript parseInt(), который обсуждается в разделе «Преобразование строки в число» главы 16.

По сути, эта функция отсекает все, что следует за числом, то есть после выполнения второй строчки кода переменная `baseFont` содержит число, например, 16. Наконец, третья строка заново устанавливает свойство `font-size`, умножая значение переменной `baseFont` на 2.

ПРИМЕЧАНИЕ

Данный код влияет на размер шрифта страницы, только если другие элементы на странице (абзацы, заголовки и т. д.) имеют размер шрифта, установленный с использованием относительных значений, например, em или процентов. Если другие элементы имеют абсолютные значения, например, пикселы, то изменение величины шрифта элемента `body` их не затронет.

Одновременное изменение нескольких свойств CSS

Если вы хотите изменить более одного свойства CSS определенного элемента, вам необязательно много раз использовать функцию `.css()`. Например, если вам нужно динамически подсвечивать элемент `div` (возможно, в качестве реакции на действия посетителя), вы можете изменить цвет фона в элементе `div` *и* добавить вокруг него рамку:

```
$('#highlightedDiv').css('background- ↵
color','#FF0000');
$('#highlightedDiv').css('border','2px solid ↵
#FE0037');
```

Другой способ — передача функции `.css()` так называемой объектной константы. *Литерал объекта (*или *объектная константа)* — это список пар свойство/значение. После каждого имени свойства вы добавляете двоеточие (`:`), за которым следует значение; пары свойство/значение разделяются запятыми; все это заключено в фигурные скобки — `{}`. Таким образом, литерал объекта для двух значений свойства CSS, приведенных выше, выглядит так:

```
{ 'background-color' : '#FF0000', 'border' : ↵
'2px solid #FE0037' }
```

Поскольку литерал объекта может быть неудобным для чтения, если он написан в одну строку, многие программисты разбивают его на несколько строк. Следующий код функционально не отличается от предыдущего однострочного кода:

```
{
'background-color' : '#FF0000',
'border' : '2px solid #FE0037'
}
```

Базовая структура объектной константы изображена на рис. 4.6.

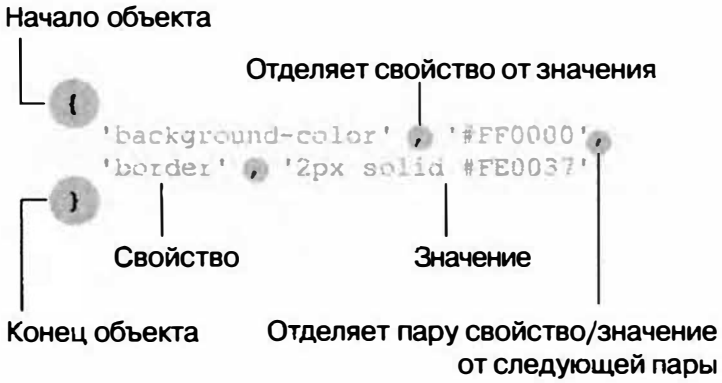

Рис. 4.6. Литерал объекта JavaScript позволяет создать список свойств и значений. Язык JavaScript считает объектную константу единым блоком информации (как массив, который является списком значений). Вы станете часто использовать подобную объектную константу, настраивая плагины jQuery

Чтобы использовать объектную константу с функцией `css()`, просто передайте ей ее:

```
$('#highlightedDiv').css({
      'background-color' : '#FF0000',
      'border' : '2px solid #FE0037'
});
```

Внимательно изучите данный пример, так как он несколько отличается от тех, что были приведены ранее, а также потому, что вы будете встречать много похожих примеров кода в следующих главах. Первая вещь, на которую следует обратить внимание, — это то, что код является единой инструкцией JavaScript (то есть просто одной строкой кода).

Такое заключение можно сделать по точке с запятой, появляющейся только в самом конце. Данная инструкция разбивается на четыре строки для упрощения чтения кода.

Далее обратите внимание на то, что литерал объекта является аргументом (как один фрагмент данных), передаваемым функции `css()`. В части кода `css({` открывающая круглая скобка является частью

функции, тогда как фигурная скобка { означает начало объекта. Рассмотрим три символа в последней строке: } — это конец литерала объекта и конец аргумента, переданного функции;) — означает окончание функции, это последняя скобка функции `css();` и ; — конец инструкции JavaScript.

Если от всего этого у вас болит голова, можете изменять свойства CSS по одной строке за раз:

```
$('#highlightedDiv').css('background- color', ↵
'#FF0000');
$('#highlightedDiv').css('border'. '2px solid ↵
#FE0037');
```

Лучше использовать встроенную в библиотеку jQuery возможность связывания (см. раздел «Связывание функций»).

Связывание применяет несколько jQuery-функций к одной коллекции элементов, добавляя функции одну за другой:

```
$('#highlightedDiv').css('background-color', ↵
'#FF0000') .css('border','2px solid #FE0037');
```

Этот код расшифровывается так: найди элемент с идентификатором `highlightedDiv` и измени цвет его фона, а затем измени цвет рамки. По сравнению с выборкой связывание характеризуется лучшей производительностью. В приведенном выше коде выбор элементов — `$('#highlightedDiv')` — производится дважды, так браузеру приходится выполнять весь jQuery-код, чтобы выбрать элемент. Поэтому данный код далек от оптимального.

```
$('#highlightedDiv').css('background- color', ↵
'#FF0000');
$('#highlightedDiv').css('border','2px solid ↵
#FE0037');
```

Данный код заставляет браузер выбирать элемент, изменять его свойство CSS, снова выбирать этот элемент (тратя на это ресурс процессора) и снова применять CSS.

При использовании метода связывания, браузер выбирает элемент только один раз, а затем дважды выполняет функцию CSS, что положительно сказывается на скорости и эффективности.

Чтение, установка и удаление атрибутов HTML

Поскольку изменение классов и свойств CSS с использованием языка JavaScript — это обычные задачи, в библиотеке jQuery для этого есть встроенные функции. Однако функции `addClass()` и `css()` — на самом деле являются просто кратчайшим способом изменения атрибутов HTML `class` и `style`.

Библиотека jQuery содержит функции общего назначения для работы с атрибутами HTML — функции `attr()` и `removeAttr()`.

Функция `attr()` позволяет читать указанный атрибут HTML-элемента. Например, чтобы определить графический файл, на который указывает элемент `img`, вы передаете функции строку `'src'` (свойство `src` элемента `img`):

```
var imageFile = $('#banner img').attr('src');
```

Функция `attr()` возвращает значение атрибута, установленное в HTML. Этот код возвращает свойство `src` для первого элемента `img` внутри другого элемента с идентификатором `banner`, так что переменная `imageFile` будет содержать путь к изображению: например, `'images/banner.png'` или `'http://www.thesite.com/images/banner.png'`.

ПРИМЕ

При передаче имени атрибута функции `attr()` можете не волноваться о регистре имени атрибута — `href`, `HREF` или даже `HrEf` будут работать.

Передавая второй аргумент функции `attr()`, вы можете установить атрибут элемента. Например, чтобы перейти к иному изображению, измените свойство `src` элемента `img` следующим образом:

```
$('#banner img').attr('src','images/newImage.png');
```

Если вы хотите полностью удалить атрибут элемента, используйте функцию `removeAttr()`. Например, данный код удаляет свойство `bgColor` из элемента `body`:

```
$('body').removeAttr('bgColor');
```

Работа с каждым элементом выборки

Как говорилось в разделе «Понимание выборок jQuery», одним из уникальных свойств библиотеки jQuery является то, что большинство функций автоматически прорабатывают в цикле каждый элемент выборки. Например, чтобы на странице исчезли все элементы `img`, нужна всего лишь одна строка кода JavaScript:

```
$('img').fadeOut();
```

Функция `.fadeOut()` заставляет элемент медленно исчезать. Прикрепив данную функцию к выборке jQuery, содержащей несколько элементов, функция прорабатывает выборку в цикле и заставляет медленно исчезать каждый ее элемент. Во многих случаях вам понадобится перебрать серию элементов и с каждым из них выполнить определенное действие. Именно для этой цели библиотека jQuery предлагает функцию `.each()`.

Предположим, вы хотите составить список всех внешних ссылок, имеющихся на странице, поместить его в рамке с библиографией и назвать, например, «Сайты, упомянутые в этой статье» (можете этого не делать, просто представьте). В любом случае такую рамку вы можете создать так:

1. **Определите все ссылки, ведущие за пределы вашего сайта.**
2. **Получите атрибут `HREF` каждой ссылки (URL-адрес).**
3. **Добавьте эту ссылку в другой список ссылок в рамке с библиографией.**

Библиотека jQuery не располагает функцией, непосредственно осуществляющей описанные шаги, но вы можете воспользоваться функцией `each()`, чтобы сделать это самостоятельно. Это просто jQuery-функция, поставьте ее в конце выборки элементов следующим образом:

```
$('selector').each();
```

Анонимные функции

Чтобы использовать функцию `each()`, вы передаете ей особый аргумент — *анонимную функцию*. Она содержит действия, которые вы хотите выполнить над каждым элементом. Такая функция называется *анонимной*, потому что, в отличие от уже изученных функций, которые

вы создавали в разделе «Функции: многократное использование кода» главы 3, вы не присваиваете ей имени. Рассмотрим базовую структуру анонимной функции:

```
function() {
      //здесь помещается код
}
```

Поскольку эта функция не имеет имени, вы не сможете ее вызвать. Например, при работе с обычной функцией вы используете ее имя и две круглые скобки: `calculateSalesTax();`. В данном случае вы используете анонимную функцию в качестве аргумента, который вы передаете другой функции (это странно, но так и есть!). Посмотрите, как вы можете сделать анонимную функцию частью функции `each()`:

```
$('selector').each(function() {
      //здесь помещается код
});
```

На рис. 4.7 изображены различные части данной конструкции. Последняя строка особенно сложна для понимания, поскольку в ней содержатся три различных символа, которые закрывают три части общей структуры. Символ `}` означает конец функции (это также конец аргумента, переданного функции `each()`); скобка `)` — это последняя часть функции `each()`; а `;` означает конец инструкции JavaScript. Другими словами, интерпретатор JavaScript считает весь этот код единой инструкцией.

Рис. 4.7. JQuery-функция `each()` позволяет перебрать все элементы выборки страницы и произвести серию действий над каждым из них. Ключом к использованию функции является понимание анонимных функций

Теперь, когда мы разобрались с внешней структурой, пора поместить что-нибудь внутрь анонимной функции: все команды, которые вы хотите применить к каждому элементу выборки. Функция `each()` действу-

ет как цикл, то есть инструкции, содержащиеся в анонимной функции, будут поочередно применяться к каждому элементу. Допустим, у вас на странице находятся 50 изображений и вы добавляете следующий код JavaScript к одному из сценариев страницы:

```
$('img').each(function() {
      alert('Найдено изображение!');
});
```

Появятся 50 диалоговых окон с оповещением «Найдено изображение». (Это очень раздражает, поэтому не пытайтесь это повторить.)

ПРИМЕЧАНИЕ

Как вы знаете, при добавлении библиотеки jQuery на страницу вам следует использовать функцию `document.ready()`, чтобы убедиться, что HTML-код будет загружен до того, как браузер приступит к выполнению кода JavaScript. Эта функция также принимает анонимную функцию в качестве аргумента:

```
$(document).ready(function() {
// код помещается здесь
// анонимная функция
});
```

Ключевые слова this и $(this)

Используя функцию `each()`, вы, конечно же, хотите получить доступ или установить атрибуты каждого элемента, например, найти URL-адрес для каждой внешней ссылки. Чтобы получить доступ к элементу, прорабатываемому в настоящий момент в цикле, используйте специальное ключевое слово `this`. Оно относится к элементу, вызывающему анонимную функцию. Так, при первом проходе цикла слово `this` относится к первому элементу выборки jQuery, при втором проходе цикла — ко второму элементу и т. д.

Слово `this` относится к элементу традиционной объектной модели документа, поэтому вы сможете получить доступ к традиционным свойствам объектной модели. Но, как вы прочитали в этой главе, особая выборка jQuery позволяет вам использовать все чудесные jQuery-функции. Так, чтобы превратить слово `this` в его эквивалент в jQuery, напишите: `$(this)`.

Вы можете подумать, что вся эта затея со словом `this` — это чья-то злая шутка, которая выдумана для того, чтобы у вас пухла голова. Это не шутка, но действительно несколько путаная вещь. Чтобы лучше разобраться в том, как использовать `$(this)`, взгляните еще раз на задачу, описанную в начале этого раздела (создание списка внешних ссылок в библиографической рамке внизу страницы).

Предположим, что в HTML-коде страницы уже есть элемент `div`, готовый для вставки в него внешних ссылок:

```
<div id="Библиография">
<h3>Веб-страницы, упомянутые в данной статье</h3>
<ul id="bibList">
</ul>
</div>
```

Первый шаг — получение списка всех ссылок, ведущих за пределы вашего сайта. Вы можете сделать это с помощью селектора атрибутов (см. раздел «Специальные селекторы»):

```
$('a[href^="http://"]')
```

Теперь, чтобы выполнить цикл для каждой ссылки, добавляем функцию `each()`:

```
$('a[href^="http://"]').each()
```

Затем добавляем анонимную функцию:

```
$('a[href^="http://"]').each(function() {
});
```

Первый шаг в анонимной функции — возвращение URL-адреса для ссылки. Поскольку каждая ссылка имеет собственный URL-адрес, вам нужен доступ к каждому элементу на каждом этапе цикла. Ключевое слово `$(this)` позволяет сделать именно это:

```
$('a[href^=http://]').each(function() {
      var extLink = $(this).attr('href');
});
```

Код, выделенный полужирным шрифтом, выполняет несколько действий: создает новую переменную (`extLink`), в которой сохраняет-

ся значение атрибута `href` выбранного элемента. При каждом проходе цикла часть `$(this)` относится к иной ссылке на странице, то есть в каждой итерации цикла значение переменной `extLink` меняется.

После этого нужно просто присоединить новый элемент списка к элементу `ul`:

```
$('a[href^=http://]').each(function() {
      var extLink = $(this).attr('href');
      $('#bibList').append('<li>' + extLink + '</li>');
});
```

Вы будете использовать ключевое слово `$(this)` почти при каждом применении функции `each()`, и со временем фрагмент кода `$(this)` станет вашим лучшим другом. Чтобы помочь вам разобраться с этим понятием, предлагаем выполнить руководство.

ПРИМЕЧАНИЕ

Сценарий, используемый в этом примере, хорошо подходит, чтобы проиллюстрировать, как используется ключевое слово `$(this)`, но он не является самым лучшим способом записи на страницу всех внешних ссылок. Во-первых, если на странице нет ссылок, элемент `div`, жестко закодированный в HTML-код страницы, по-прежнему станет появляться, но будет пустым. Кроме того, если у посетителя страницы отключен JavaScript, он увидит не ссылки, а только пустую рамку. Более рационально использовать JavaScript также для создания заключающего элемента `div`. Пример этого вы найдете в файле *04_01.html* среди файлов примеров для данной главы.

Автоматические «броские цитаты»

В последнем руководстве этой главы вы создадите сценарий, упрощающий добавление на страницу «броских цитат» (подобных тем, что изображены на рис. 4.8). *«Броской цитатой»* называют интересную фразу из основного текста страницы, заключенную в рамку. Газеты, журналы и сайты используют такие цитаты для привлечения внимания читателей и выделения важных или интересных сведений. Добавление «броских цитат» вручную требует копирования текста со страницы и помещения его в элемент `div`, `span` или другой контейнер. Все это отнимает много времени, добавляет к окончательному варианту страницы

лишний HTML-код и дублированный текст. К счастью, язык JavaScript позволяет быстро поместить на страницу любое количество «броских цитат» с помощью небольшого HTML-кода.

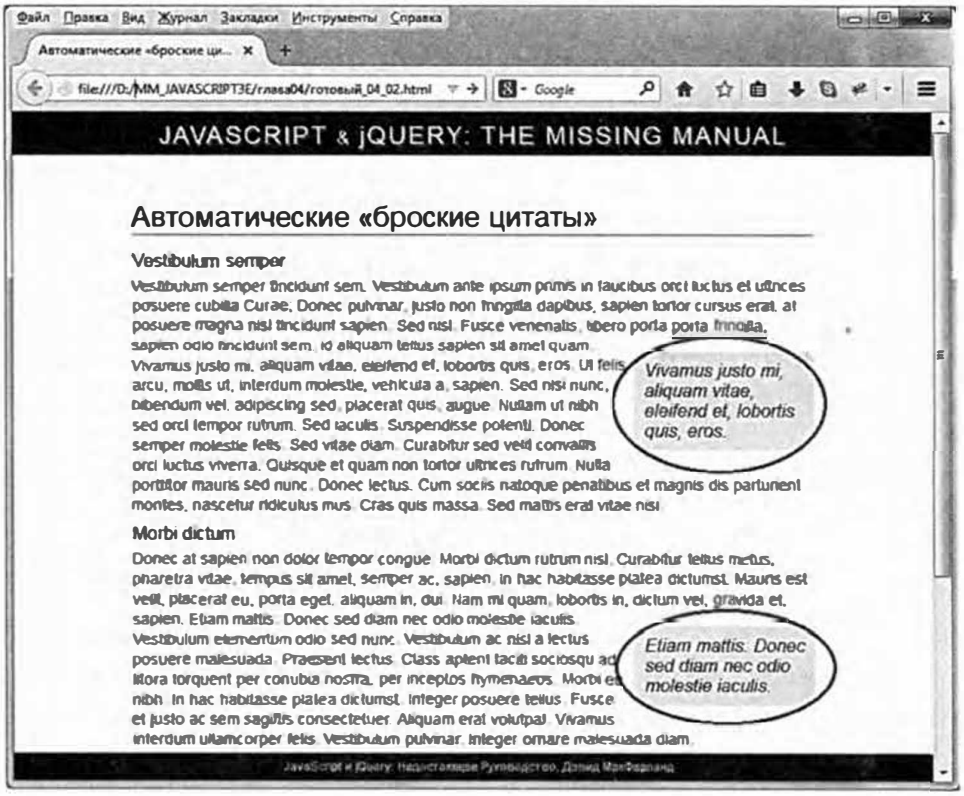

Рис. 4.8. Добавление «броских цитат» в HTML-код страницы вручную — это сложно. Вы можете использовать язык JavaScript для автоматизации этого процесса. В данном случае две «броские цитаты», сгенерированные с помощью кода JavaScript, находятся в правой части страницы (выделены на рисунке)

Обзор

Сценарий, который вы сейчас создадите, будет выполнять следующие действия.

1. **Находить все элементы `span`, содержащие особый класс `pq` (pull quote — «броская цитата»).**

   Все, что нужно сделать с HTML-кодом вашей страницы, это заключить в элементы `span` тот текст, который вы хотите сделать «броскими цитатами». Предположим, на странице есть абзац текста и вы хотите выделить из него несколько слов. Просто поместите этот текст между открывающим и закрывающим тегами элемента `span`:

```
<span class-"pq">...вот как я обнаружил ↵
Лох-Несское чудовище.</span>
```

2. **Дублировать все элементы `span`.**

   Каждая рамка с «броской цитатой» по существу является элементом `span`, в котором содержится текст цитаты. Используйте JavaScript, чтобы продублировать существующий элемент `span`.

3. **Удалять класс `pq` из дублированного элемента `span` и добавлять в него новый класс `pullquote`.**

   Магия форматирования «броской цитаты» (рамка, большой кегль, цвет границы и фона) не является заслугой языка JavaScript. Таблица стилей страницы содержит селектор класса CSS `.pullquote`, который и делает все это. Итак, просто воспользовавшись JavaScript для изменения имени класса дублированного элемента, вы полностью измените вид новых элементов `span`.

4. **Добавлять на страницу дублированный элемент `span`.**

   Наконец, вы добавляете на страницу дублированный элемент `span` (в шаге 2 элемент просто сохраняется в памяти браузера, но не добавляется на страницу; это позволяет и далее работать с дублированным элементом, прежде чем показать его пользователю, просматривающему страницу).

Верстка кода

Теперь вы представляете, чего собираетесь добиться с помощью сценария. Пора открыть текстовый редактор и воплотить задумку в жизнь.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *04_02.html* из каталога *глава04*.**

   Вы начнете с добавления ссылки на файл jQuery.

2. **Щелкните мышью в пустой строке над закрывающим тегом `</head>` и введите следующий код:**

```
<script src="../_js/jquery.min.js"></script>
```

Этот код загружает файл jQuery. Заметьте, что имя папки, содержащей файл jQuery — *_js* (не забудьте символ нижнего подчеркивания в самом начале). Далее вы добавите набор тегов элемента `script` для своей программы.

ПРИМЕЧАНИЕ

Вы увидите, что при создании ссылки на файл jQuery не указывается номер версии, хотя, как следует из приведенного выше текста в рамке, вам нужно использовать номер версии следующим образом:

```
<script src="../_js/jquery.1.11.0.min.js"></script>
```

Номер версии опущен с целью облегчения процесса обновления файлов примеров новыми версиями библиотеки jQuery. Например, на момент написания данной книги новейшей версией является jQuery 1.11.0 (разница между версиями 1 и 2 описана во врезке в разделе «Ссылки на файл jQuery, расположенный на сервере CDN» в начале данной главы). Однако к тому моменту, как эта книга попадет к вам в руки, может выйти версия 1.11.1 или 1.12.0. Файлы примеров будут включать последнюю версию библиотеки jQuery, чтобы вы могли воспользоваться всеми нововведениями.

3. **Нажмите клавишу Enter для создания новой строки под кодом jQuery и добавьте выделенный полужирным шрифтом код:**

```
<script src="../_js/jquery.min.js"></script>
<script>
</script>
```

ПРИМЕЧАНИЕ

Номера строк слева указаны просто для удобства. Не вводите их в качестве части сценария на веб-страницу.

Теперь добавьте функцию `document.ready()`.

4. **Щелкните мышью в пустой строке между тегами элемента `script` и добавьте выделенный полужирным шрифтом код:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4
```

```
5 }); // конец ready
6 </script>
```

Использование комментариев JavaScript (`// конец ready`) становится все более удобным по мере того, как ваши программы удлиняются и усложняются. В объемной программе у вас будет множество сочетаний символов `});`, каждая из которых означает окончание анонимной функции и вызова функции. Добавление комментария, указывающего на то, чему соответствует то или иное сочетание символов `});` упрощает восприятие и понимание кода.

Шаги с 1 по **4** — это подготовка к написанию любой программы с использованием библиотеки jQuery, поэтому важно, чтобы вы понимали логику описанных выше действий. Далее вы создадите сердце своей программы, выбрав элементы `span`, содержащие текст, который должен появиться в качестве «броских цитат».

5. **Добавьте код, выделенный полужирным шрифтом в строке 4:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function( ) {
4 $('span.pq')
5 }); // конец ready
6 </script>
```

Часть `$('span.pq')` — это селектор jQuery, который находит каждый элемент `span` с присвоенным ему классом `pq`. Добавим код, необходимый для работы с этими элементами `span` в цикле.

6. **Добавьте код, выделенный полужирным шрифтом в строках 4 и 6:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function( ) {
4  $('span.pq').each(function() {
5
6  }); // конец each
7 }); // конец ready
8 </script>
```

Как уже говорилось, jQuery-функция `.each()` позволяет прорабатывать в цикле выборку элементов. Функция принимает аргумент, который является анонимной функцией.

Далее вы начнете создавать функцию, которая будет применяться к каждому элементу `span`, найденному на этой странице. Начните с создания копии элемента `span`.

7. Добавьте код, выделенный полужирным шрифтом в строке 5:

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function( ) {
4  $('span.pq').each(function() {
5        var quote=$(this).clone();
6        }); // конец each
7 }); // конец ready
8 </script>
```

Данная функция начинает работу с создания новой переменной `quote`, которая содержит «клон» (или просто копию) существующего элемента `span` (см. раздел «Ключевые слова `this` и `$(this)`», если забыли, что означает часть `$(this)`). JQuery-функция `.clone()` дублирует элемент, включая весь HTML-код внутри него. В данном случае делается копия элемента `span`, включая текст внутри этого элемента, который появится в рамке «броской цитаты».

Клонирование позволяет скопировать элемент полностью, включая все присвоенные ему атрибуты. В данном примере первоначальный элемент `span` имеет класс `pq`. Удалим этот класс из копии.

8. Добавьте две строки кода, выделенные полужирным шрифтом в строках 6 и 7:

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function( ) {
4  $('span.pq').each(function() {
5        var quote=$(this).clone();
6        quote.removeClass('pq');
```

```
7          quote.addClass('pullquote');
8   }); // конец each
9 }); // конец ready
10 </script>
```

Как говорилось в разделе «Классы», функция `removeClass()` удаляет имеющийся у элемента класс, а функция `addClass()` добавляет элементу новый класс. В данном случае мы заменяем класс в копии, поэтому вы сможете использовать класс CSS с именем `.pullquote` для форматирования элемента `span` как рамки для «броской цитаты».

Наконец, добавим элемент `span` на страницу.

9. Добавьте код, выделенный полужирным шрифтом (строка 8):

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function( ) {
4   $('span.pq').each(function() {
5         var quote=$(this).clone();
6         quote.removeClass('pq');
7         quote.addClass('pullquote');
8         $(this).before(quote);
9   }); // конец each
10 }); // конец ready
11 </script>
```

Эта строка завершает функцию — до нее вы работаете с копией элемента `span`, находящейся в памяти браузера. Никто из просматривающих страницу не увидит этой копии, пока она не будет добавлена в объектную модель документа.

В данном случае вы вставляете копию элемента `span` перед уже имеющимся в вашем HTML-коде одноименным элементом. В сущности, на странице получится примерно следующий HTML-код:

```
<span class="pullquote">...вот как я обнаружил ↵
Лох-Несское чудовище.</span> <span class-"pq"> ↵
...вот как я обнаружил Лох-Несское чудовище.</span>
```

Хотя кажется, что текст просто продублирован, форматирование CSS заставляет рамку с «броской цитатой» переместиться к правому краю страницы.

ПРИМЕЧАНИЕ

Для достижения визуального эффекта рамки с «броской цитатой» на странице присутствует стиль CSS, использующий свойство CSS `float`. Рамка с цитатой появляется у правого края абзаца, в котором содержится текст данной цитаты, а текст абзаца обтекает рамку. Если эта техника вам не знакома, то вы можете прочитать о свойстве CSS `float` на сайте: **css.maxdesign.com.au/floatutorial/**. Если вы хотите исследовать стиль `.pullquote`, просто обратитесь к разделу заголовка в файле примера. Этот стиль и его свойства перечислены там.

Теперь JavaScript-код завершен. Однако вы не увидите никаких рамок с «броскими цитатами», пока «не отшлифуете» HTML-код страницы.

10. **Найдите первый элемент `p` в HTML-коде страницы, затем выберите в тексте фразу и заключите ее в теги `<span class="pq">` и `</span>`. Например:**

```
<span class="pq">Nullam ut nibh sed orci tempor↵
rutrum.</span>
```

Вы можете повторить этот процесс, чтобы добавить «броские цитаты» в другие абзацы.

11. **Сохраните файл и просмотрите его в браузере.**

Конечный результат должен выглядеть примерно так, как показано на рис. 4.8. Если вы не видите рамки с «броской цитатой», убедитесь, что правильно добавили тег `</span>` на шаге 10. Также просмотрите советы в главе 1, касающиеся исправления ошибок в программе. Полная версия этого руководства находится в файле *готовый_04_02.html*.

Глава 5

ДЕЙСТВИЕ/РЕАКЦИЯ: ИНТЕРАКТИВНЫЕ СТРАНИЦЫ С ПОМОЩЬЮ СОБЫТИЙ

При обсуждении языка JavaScript вы часто можете слышать слово «интерактивный»: «JavaScript позволяет создавать интерактивные веб-страницы». На самом деле имеется в виду то, что с помощью языка JavaScript вы можете заставить свои веб-страницы реагировать на действия пользователя: наведение указателя мыши на кнопку навигации выводит список ссылок; нажатие на переключатель активизирует какие-либо варианты в форме, щелчок по эскизу картинки затеняет страницу и показывает на экране крупную версию данного изображения.

Все действия посетителя, на которые может реагировать веб-страница, называются *событиями*. Язык JavaScript является *событийно управляемым* языком: без событий ваши веб-страницы были бы неспособны отвечать на действия пользователей или делать что-нибудь на самом деле интересное. Это как настольные приложения. Вы включаете компьютер, но по-настоящему он начинает работать только тогда, когда вы щелкаете по значкам файлов, выбираете пункты меню и передвигаете указатель мыши по экрану.

Что такое события?

Браузеры запрограммированы на распознавание простейших действий, таких как загрузка страницы, передвижение указателя мыши, нажатие клавиши на клавиатуре или изменение размера окна браузера. Каждое из этих действий на странице является *событием*. Чтобы сделать веб-страницу интерактивной, вы пишете программы, реагирующие на события. Таким образом, вы можете запрограммировать появление или исчезновение элемента `div` при щелчке кнопкой мыши; появление изображения при наведении указателя на ссылку; проверку содержания текстового поля, при нажатии на кнопку отправки формы.

Событие представляет собой точный момент, в который что-либо происходит. Например, вы нажимаете кнопку мыши, а в тот момент,

когда вы ее отпускаете, браузер сигнализирует, что произошел щелчок (событие `click`). Момент, в который произошло событие, программисты называют *запуском события*.

При щелчке кнопкой мыши браузер на самом деле запускает несколько событий. Как только вы нажимаете кнопку, происходит событие `mousedown`, а когда вы ее отпускаете — `mouseup` и затем `click` (рис. 5.1).

ПРИМЕЧАНИЕ

Понимание того, как и когда запускаются эти события, может прийти не сразу. Чтобы вы поработали с различными типами событий, в папку с файлами примеров включена демонстрационная страница. Откройте в браузере файл *события.html*, находящийся в каталоге *эксперименты*. Затем переместите указатель мыши, щелкните и введите какой-нибудь текст, чтобы увидеть некоторые из событий, происходящих на веб-странице (см. рис. 5.1).

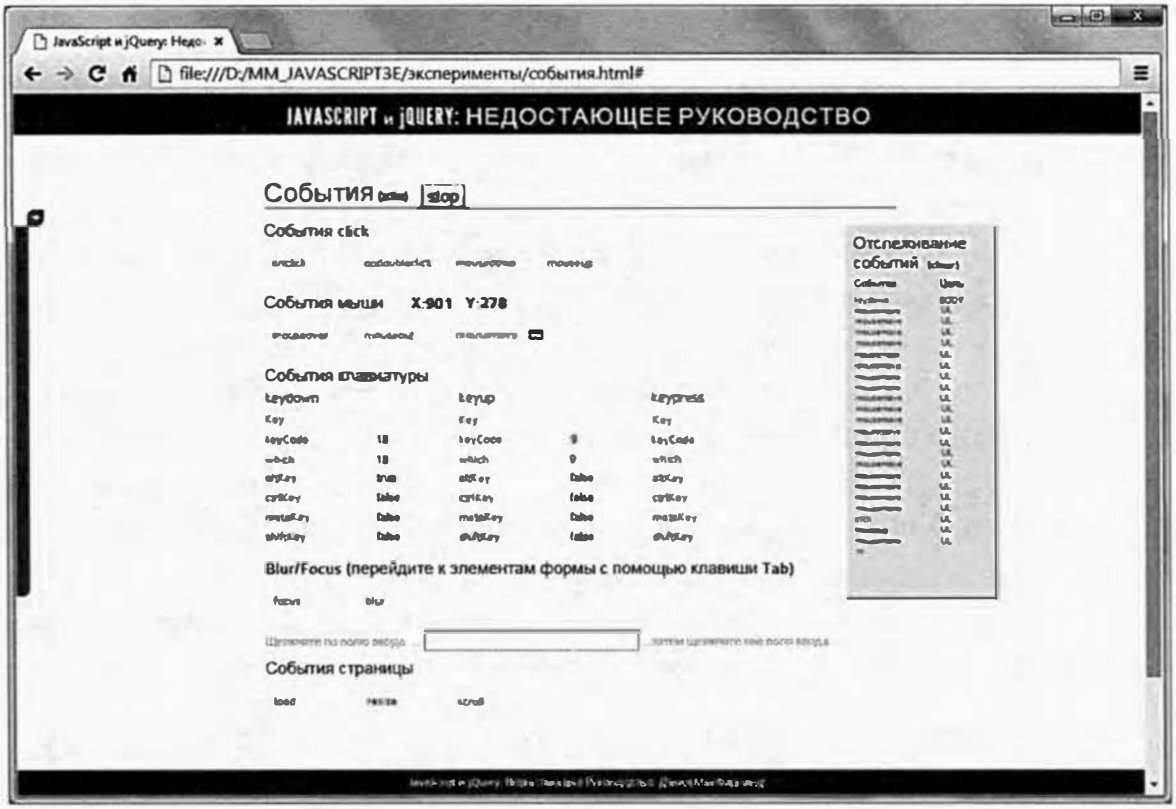

Рис. 5.1. Хотя вы можете этого не осознавать, браузеры постоянно запускают события, когда вы щелкаете кнопкой мыши, печатаете или перемещаете курсор. Например, двойной щелчок кнопкой мыши запускает события `mousedown`, `mouseup` и `onclick`, а также событие `doubleclick`. Файл *события.html* (в папке *эксперименты*) демонстрирует многие из таких событий в действии

События мыши

С тех пор как Стив Джобс выпустил компьютер Macintosh в 1984 г., мышь стала важнейшим устройством для всех персональных компьютеров. Ее используют, чтобы открывать приложения, перетаскивать файлы в папки, выбирать элементы в меню и даже рисовать. Браузеры предлагают много способов отслеживания того, как посетитель использует мышь при работе с веб-страницей.

- **`click`**. Это событие запускается, когда вы нажимаете и отпускаете кнопку мыши. Обычно оно присваивается ссылке, например, с эскизом изображения, при щелчке по которой отображается рисунок большего размера. Однако вы не ограничены только ссылками. Вы можете заставить любой элемент реагировать на событие — даже на простой щелчок в любом месте страницы.

ПРИМЕЧАНИЕ

Событие `click` действует и в том случае, если ссылка нажимается посредством клавиатуры. Перейдя к ссылке с помощью клавиши **Tab**, нажмите клавишу **Enter** — и событие запустится.

- **`dblclick`**. Данное событие запускается, когда вы дважды нажимаете и отпускаете кнопку мыши. Это действие вы выполняете, чтобы открыть файл или каталог на рабочем столе или в окне файлового менеджера операционной системы. Щелчок по веб-странице — это не самое обычное действие, поэтому, используя это событие, вы должны ясно дать посетителю понять, в каком месте страницы можно совершить двойной щелчок и что после этого произойдет. Следует также отметить, что двойной щелчок кнопкой мыши равносилен двум одиночным щелчкам, поэтому не присваивайте события одиночного и двойного щелчка одному и тому же элементу. В противном случае функция одинарного щелчка будет выполнена дважды до того, как запустится функция двойного щелчка.
- **`mousedown`**. Это первая часть щелчка — момент, когда вы нажимаете кнопку мыши, но не отпускаете ее. Удобно при перетаскивании элементов по странице. Вы можете позволить посетителям перетаскивать элементы по странице так, как они это делают с ярлыками на рабочем столе операционной системы. Пользователь должен щелкнуть кнопкой мыши по какому-либо элементу и, не отпуская ее, переместить элемент на новое место, а потом отпустить кнопку

мыши, в результате произойдет перемещение (вы узнаете о том, как обеспечить это с помощью плагина jQuery UI в главе 12).

- **mouseup**. Событие `mouseup` — это вторая часть щелчка — момент, когда вы отпускаете кнопку мыши. Это событие удобно использовать тогда, когда вы «бросаете» на новое место элемент, который раньше находился в другом месте.
- **mouseover**. Запускается, когда вы проводите указателем мыши по элементу страницы. Используя событие `mouseover`, вы можете присвоить навигационной кнопке обработчик событий, заставляя появляться подменю, когда посетитель проводит указателем мыши по кнопке. (Если вы привыкли к использованию псевдокласса CSS `:hover`, то знаете, как работает это событие.)
- **mouseout**. Запускается, когда вы смещаете курсор с элемента. Вы можете использовать это событие, чтобы сигнализировать о том, что посетитель убрал указатель мыши со страницы, либо чтобы спрятать всплывающее меню, когда указатель покидает его.
- **mousemove**. Это событие запускается, когда вы передвигаете курсор, то есть оно происходит постоянно. Вы используете событие `mousemove` для отслеживания текущего положения указателя мыши на экране. Дополнительно вы можете присвоить его отдельно взятому элементу в коде страницы, например, `div`, и реагировать только на передвижения указателя мыши внутри этого элемента.

ПРИМЕЧАНИЕ

Поскольку событие `mousemove` запускается очень часто (множество раз при движении мыши), будьте осторожны, назначая этому событию действия. Обработка действий в ответ на каждое движение мыши может значительно замедлить работу программы и понизить общую отзывчивость веб-страницы.

События документа/окна

Окно браузера само по себе «понимает» запуск различных событий, начиная от момента загрузки страницы до момента, когда посетитель покидает ее:

- **load.** Запускается, когда браузер заканчивает загрузку всех файлов веб-страницы: самого HTML-файла, связанных изображений, Flash-фильмов, внешних файлов CSS и JavaScript. Обычно веб-

дизайнеры используют это событие для запуска любой программы JavaScript, манипулирующей веб-страницей. Однако загрузка веб-страницы и всех ее файлов может занять много времени, если с ней связано много графических или других крупных файлов. Иногда это означает, что в течение некоторого времени после отображения страницы в браузере код JavaScript не будет работать. К счастью, библиотека jQuery предлагает более удобную замену для события `load` (см. раздел «Больше концепций для событий jQuery» данной главы).

- **`resize`**. Запускается, когда вы изменяете размер окна браузера, нажимая кнопку разворачивания окна, либо изменяете ширину окна браузера, перетаскивая его границу. Некоторые дизайнеры используют это событие для изменения шаблона страницы, когда посетитель изменяет размер окна своего браузера. Например, можно проверить его ширину — если окно очень широко, то следует изменить дизайн и добавить больше столбцов, чтобы контент поместился на странице.
- **`scroll`**. Запускается, когда вы используете полосу прокрутки или клавиатуру (клавиши **PageUp**, **PageDown**, **Home**, **End** и т. д.) либо прокручиваете веб-страницу с помощью колеса мыши. Если на странице нет полос прокрутки, событие `scroll` не запускается. Некоторые программисты используют его, чтобы понять, в каком месте на экране появятся элементы (после прокрутки страницы).

ПРИМЕЧАНИЕ

Как и в случае с событием `mousemove`, событие `scroll` запускается снова и снова при прокрутке страницы. Поэтому будьте осторожны, назначая этому событию сложные действия.

- **`unload`**. Запускается, когда вы щелкаете по ссылке для перехода на другую страницу, закрываете вкладку в окне браузера или само окно. Это событие можно сравнить с «последним вздохом» программы JavaScript, и оно дает вам возможность совершить последнее действие, прежде чем посетитель покинет страницу. Нечистоплотные программисты используют это событие, чтобы предельно усложнить процесс покидания страницы. Каждый раз, когда посетитель пытается покинуть страницу, появляется новое окно, и страница загружается снова. Но это событие можно использовать и с добрыми намерениями, например, программа может предупреждать посети-

теля о форме, которую он начал заполнять, но не отправил, или программа может отослать данные формы на веб-сервер, чтобы сохранить информацию, прежде чем посетитель покинет страницу.

События форм

В прежние времена, еще до появления языка JavaScript, люди работали с сайтами в основном посредством форм, созданных с помощью HTML. Ввод данных в поле формы был единственным способом отправки информации на сайт. Поскольку формы все еще являются важной частью Всемирной паутины, вы узнаете о многих событиях форм, которые сможете использовать.

- **`submit`**. Запускается, когда посетитель отправляет данные формы с помощью щелчка по кнопке **Submit** (Отправить) или нажатия клавиши **Enter**, когда курсор находится в текстовом поле. Наиболее часто это событие используется при проверке форм, чтобы убедиться, что все требуемые элементы формы правильно заполнены, до того как данные будут отосланы на веб-сервер. Вы узнаете о проверке форм в разделе «Проверка формы» главы 8.

- **`reset`**. Пусть и нечасто, но кнопка **Reset** (Отменить) может пригодиться вам, чтобы отменить любые изменения, сделанные в форме. Это событие возвращает страницу в состояние, в котором она находилась на тот момент, когда она была загружена. Вы можете запустить сценарий, когда посетитель пытается отменить форму, используя событие `reset`. Например, если пользователь внес в форму какие-либо изменения, вы можете вызвать диалоговое окно с вопросом: «Вы уверены, что хотите отменить сделанные изменения?». Это окно даст пользователям возможность нажать кнопку «НЕТ» и предотвратить обнуление (очистку) формы.

- **`change`**. Многие элементы формы запускают это событие при изменении их статуса, например, когда кто-нибудь нажимает переключатель или выбирает пункт из выпадающего меню. Вы можете использовать событие `change`, чтобы сразу же проверить выбор, сделанный в меню, или посмотреть, какой переключатель был выбран.

- **`focus`**. Запускается, когда вы переходите к элементу формы, щелкая по нему мышью или нажимая клавишу **Tab**. Иначе говоря, теперь внимание браузера сфокусировано на этом элементе страницы. Так, выбирая переключатель или устанавливая флажок, вы вводите эти элементы в фокус. Вы можете реагировать на событие `focus` с по-

мощью языка JavaScript. Например, добавить полезную инструкцию в текстовое поле: «Введите ваше имя». Когда посетитель устанавливает текстовый курсор в поле (вводя его в фокус), вы можете стереть эти инструкции, и у пользователя будет пустое поле, которое можно заполнить.

- **`blur`**. Противоположно событию `focus`. Запускается, когда элемент формы выводится из фокуса (при нажатии клавиши **Tab** или при щелчке кнопкой мыши за пределами элемента). Событие `blur` также полезно при проверке форм. Например, когда кто-либо вводит свой электронный адрес в текстовое поле и клавишей **Tab** переходит к следующему элементу, вы можете сразу же проверить, что ввел посетитель, чтобы убедиться, что это — правильный почтовый адрес.

ПРИМЕЧАНИЕ

События `focus` и `blur` также применимы к ссылкам на странице. Когда вы с помощью клавиши **Tab** переходите к ссылке, запускается событие `focus`, а когда вы «сходите» со ссылки с помощью клавиши **Tab** (или щелчка кнопкой мыши), запускается событие `blur`.

События клавиатуры

Браузеры также отслеживают, как посетитель использует свою клавиатуру, поэтому вы можете присваивать клавишам команды или позволять человеку управлять сценарием, нажимая различные клавиши. Например, нажатие клавиши **Пробел** может запустить или остановить анимацию JavaScript.

К сожалению, браузеры по-разному реагируют на события клавиатуры, иногда сложно даже установить, какая буква была введена! (Технику определения того, какая буква была введена с клавиатуры, вы найдете в совете в разделе «Объект события» данной главы.)

- **`keypress`**. Это событие запускается в тот момент, когда вы нажимаете клавишу. Причем, чтобы оно сработало, вам необязательно эту клавишу отпускать. Событие `keypress` запускается снова и снова до тех пор, пока клавиша удерживается нажатой, поэтому данное событие позволяет проверить, удерживает ли посетитель клавишу нажатой. Например, если вы создали гоночную онлайн-игру, вы можете назначить клавишу для педали газа. Игроку требуется только нажать и удерживать эту клавишу, чтобы заставить машину двигаться.

- **keydown**. Подобно событию `keypress`, оно запускается при нажатии клавиши. На самом деле оно запускается непосредственно *перед* запуском события `keypress`. В браузере Opera событие `keydown` запускается только один раз; в других программах работает так же, как и событие `keypress`, — запускается снова и снова, пока клавиша нажата.
- **keyup**. Запускается, когда вы отпускаете клавишу.

Использование событий: способ jQuery

Традиционно программирование с использованием событий было сложным. Долгое время браузер Internet Explorer работал с событиями иначе, чем другие браузеры, требуя двух фрагментов кода (один для браузера Internet Explorer и другого для всех остальных браузеров), чтобы код работал. К счастью, браузер Internet Explorer9 теперь использует тот же метод обработки событий, как и другие браузеры, так что процесс программирования стал намного проще. Однако многие люди по-прежнему используют браузер Internet Explorer8 и более ранние версии, поэтому необходимо хорошее решение, делающее программирование с использованием событий простым и совместимым со всеми браузерами. К счастью, у вас есть библиотека jQuery.

Как вы узнали из предыдущей главы, библиотеки JavaScript, такие как jQuery, решают множество проблем, связанных с программированием на языке JavaScript, включая досадную несовместимость браузеров. Библиотеки также зачастую упрощают решение основных задач, связанных с JavaScript. jQuery облегчает процесс назначения событий и *помощников событий* (функций, работающих с событиями).

Как вы знаете, процесс jQuery-программирования включает (а) выбор элемента страницы и (б) произведение некоторых действий над этим элементом. В самом деле, так как события являются неотъемлемой частью программирования на языке JavaScript, следует воспринимать jQuery-программирование как процесс, состоящий из трех шагов.

1. Выберите один или более элементов.

В предыдущей главе было объяснено, как библиотека jQuery позволяет использовать селекторы CSS для выбора частей страницы, с которыми вы хотите работать. Присваивая события, вам следует выбирать элементы, с которыми посетитель будет взаимодействовать. Например,

по каким элементам посетитель станет щелкать кнопкой мыши — по ссылке, ячейке таблицы или изображению? Если вы хотите присвоить событие наведения указателя мыши (`mouseover`), то на какой элемент страницы он должен навести курсор, чтобы действие произошло?

2. Назначьте событие.

В библиотеке jQuery существуют функции, эквивалентные большинству событий объектной модели документа (DOM). Итак, чтобы присвоить событие элементу, вы просто добавляете точку, имя события и пару круглых скобок. Так, например, если вы хотите добавить событие наведения указателя мыши к каждой ссылке на странице, то вы можете это сделать следующим образом:

```
$('a').mouseover();
```

Чтобы добавить событие щелчка элементу с идентификатором `menu`, напишите следующий код:

```
$('#menu').click();
```

Вы можете использовать любое из событий, перечисленных в начале данной главы (а также события, которые обсуждаются в разделе «События jQuery»).

Добавление события — это еще не все. Чтобы в момент запуска события что-то происходило, вы должны предоставить событию функцию.

3. Передайте функцию событию.

Вы должны определить, что должно происходить при запуске события. Чтобы это сделать, просто присвойте событию функцию. В ней должны содержаться команды, которые станут выполняться при запуске события, например, отображение ранее скрытого элемента `div` или подсвечивание элемента, на котором находится указатель мыши.

Вы можете передать имя функции следующим образом:

```
$('#start').click(startSlideShow);
```

Когда вы присваиваете событию функцию, то опускаете круглые скобки (), обычно добавляемые после имени функции для ее вызова. Другими словами, следующий код работать не будет:

```
$('#start').click(startSlideShow())
```

Однако наиболее распространенный способ добавления функции событию — это передать событию *анонимную функцию* — функцию без имени (о ней вы можете прочитать в разделе «Анонимные функции» главы 4). Общая структура анонимной функции выглядит следующим образом:

```
function() {

// здесь находится ваш код

}
```

Схема использования анонимной функции с событием изображена на рис. 5.2.

ПРИМЕЧАНИЕ

Чтобы подробнее узнать о том, как работать с библиотекой jQuery и событиями, посетите сайт **api.jquery.com/category/events/**.

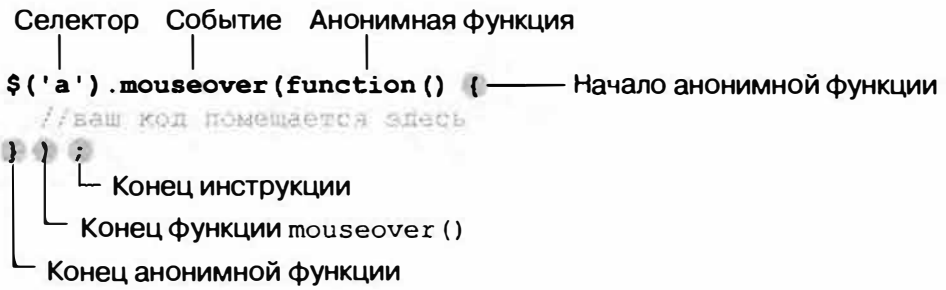

Рис. 5.2. В библиотеке jQuery событие работает как функция, то есть ему можно передать аргумент. Можно рассматривать анонимную функцию в качестве аргумента — фрагмента данных, переданного функции. Воспринимая ее таким образом, вы легче поймете, за что отвечает каждый знак пунктуации. Например, в последней строке символ } означает конец анонимной функции (и конец аргумента, переданного функции `mouseover()`); символ) — это конец функции `mouseover()`, а точка с запятой — это конец всей инструкции, которая начиналась с селектора `$('a')`

Рассмотрим простой пример. Допустим, у вас есть веб-страница со ссылкой, которой присвоен идентификатор `menu`. Когда посетитель наводит указатель мыши на эту ссылку, вы хотите, чтобы появился скрытый список дополнительных ссылок, имеющий идентификатор `submenu`. Итак, вам нужно: присвоить меню событие `mouseover()`, а затем вызвать функцию, показывающую подменю. Процесс можно подразделить на четыре шага.

1. **Выберите меню:**

```
$('#menu')
```

2. **Прикрепите событие:**

```
$('#menu').mouseover();
```

3. **Добавьте анонимную функцию:**

```
$('#menu').mouseover(function() {
}) // конец события mouseover
```

Вам часто станут встречаться сочетания закрывающей фигурной скобки, закрывающей круглой скобки и точки с запятой — });, которые отмечают конец анонимной функции внутри вызова функции. Поскольку вы будете встречать их повсюду, следует добавлять комментарий JavaScript — в данном примере: `// конец события mouseover`, чтобы уточнить, что означают эти сочетания знаков препинания.

4. **Добавьте необходимые действия (в данном случае — отображение подменю):**

```
$('#menu').mouseover(function() {
$('#submenu').show();
});// конец события mouseover
```

Многие считают «безумное множество» знаков препинания, связанное с анонимными функциями, очень сложным для понимания (эти последние }); особенно хороши). Пунктуация действительно запутана, но лучший способ освоиться в странном мире языка JavaScript — это набить руку в упорной практике, и следующее руководство поможет закрепить только что представленные идеи.

ПРИМЕЧАНИЕ

Функция `show()` обсуждается в следующей главе в разделе «Основы отображения и сокрытия».

Использование событий на практике

Данное руководство предоставляет вам краткое введение в процесс использования событий. Вы заставите страницу реагировать на несколько различных типов событий, так что вы получите представление о том, как работают события jQuery и как их использовать.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 за информацией о том, как скопировать файлы примеров.

1. **В текстовом редакторе откройте файл *05_01.html* в каталоге *глава05*.**

   Вы начнете с добавления ссылки на файл jQuery.

2. **Установите текстовый курсор в начало пустой строки перед закрывающим тегом `</head>` и введите следующий код:**

```
<script src="../_js/jquery.min.js"></script>
```

   Эта строка загружает файл jQuery. Обратите внимание на то, что имя папки, содержащей файл jQuery — `_js` (не забудьте символ нижнего подчеркивания в начале). Далее вы добавите набор тегов элемента `script` для вашего программного кода.

3. **Нажмите клавишу Enter, чтобы создать новую пустую строку под кодом загрузки файла jQuery и добавьте код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery.min.js"></script>

<script>

</script>
```

   Теперь добавьте функцию `document.ready()`.

4. **Щелкните в пустой строке между тегами элемента `script` и добавьте код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery.min.js"></script>

<script>

$(document).ready(function() {

}); // конец функции ready

</script>
```

   Не забудьте указать комментарий после символов `});`. Несмотря на то, что написание комментариев занимает время, они окажутся весь-

ма полезны при выделении частей программы. На данный момент вы выполнили шаги, необходимые при использовании библиотеки jQuery на странице.

Далее необходимо добавить событие. Ваша первая задача будет простой: вывод окна оповещения при двойном щелчке кнопкой мыши в любом месте страницы. Для начала вы должны выбрать элемент (в данном случае — страницу), к которому вы собираетесь добавить событие.

5. **Щелкните в пустой строке внутри функции `.ready()` и добавьте код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html')
}); // конец функции ready
</script>
```

Код `$('html')` выбирает элемент HTML, то есть все окно браузера. Далее вы добавите событие.

6. **После селектора jQuery введите `.dblclick();// конец события double click`, чтобы ваш код выглядел следующим образом:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(); // конец события double click
}); // конец функции ready
</script>
```

Часть `.dblclick()` — это jQuery-функция, которая подготавливает браузер к выполнению какого-либо действия при двойном щелчке кнопкой мыши. Отсутствует только часть «какое-либо действие», которая требует передачи анонимной функции в качестве аргумента функции `dblclick()` (если вам нужно освежить свои знания о принципе работы функций и что означает «передача аргумента», обратитесь к разделу «Функции: многократное использование кода» главы 3).

7. **Добавьте анонимную функцию — код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(function() {
}); // конец события double click
}); // конец функции ready
</script>
```

Не волнуйтесь, остальные руководства в этой книге не будут расписаны так детально, однако очень важно понять, что делает каждая часть кода. Часть `function() {}` — это просто внешняя оболочка, она ничего не делает до тех пор, пока вы не добавите программный код между фигурными скобками: это следующий шаг.

8. **Наконец, добавьте инструкцию, вызывающую окно оповещения:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(function() {
alert('ой');
}); // конец события double click
}); // конец функции ready
</script>
```

Если вы просмотрите страницу в веб-браузере и дважды щелкните в любом месте страницы, появится окно оповещения JavaScript со словом «ой». Если оно не появляется, перепроверьте ваш код, чтобы удостовериться, что вы ничего не пропустили.

ПРИМЕЧАНИЕ

После стольких усилий появление слова «ой» на экране может оставить вас с ощущением неоправданных ожиданий. Но помните, что часть `alert()` данного сценария не существенна — остальной код,

который вы ввели, демонстрирует фундаментальные основы того, как использовать события с библиотекой jQuery. По мере того, как вы приобретаете больше знаний о программировании и библиотеке jQuery, вы сможете легко заменить окно оповещения серией действий, которая (при двойном щелчке по странице) перемещает элемент по экрану, отображает интерактивное слайд-шоу или начинает игру.

Теперь, когда вы знакомы с основами, вы попробуете несколько других событий.

1. **Добавьте фрагмент, выделенный полужирным шрифтом, чтобы ваш код выглядел следующим образом:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(function() {
alert('ой');
}); // конец события double click
$('a').mouseover(function() {
}); // конец события mouseover
}); // конец функции ready
</script>
```

   Данный код выбирает все ссылки на странице (часть `$('a')`), затем добавляет анонимную функцию событию `mouseover`. Другими словами, когда кто-то наводит указатель мыши на ссылку, что-нибудь происходит.

2. **Добавьте две инструкции JavaScript к анонимной функции, которую вы ввели в предыдущем шаге:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(function() {
alert('ой');
```

```
}); // конец события double click
$('a').mouseover(function() {
var message = "<p>Вы навели указатель мыши на ↵
ссылку</p>";
$('.main').append(message);
}); // конец события mouseover
}); // конец функции ready
</script>
```

Первая строка, `var message = "<p>Вы навели указатель мыши на ссылку</p>";`, создает новую переменную `message` и сохраняет в ней строку. Строка — это HTML-элемент абзаца с текстом. Следующая строка выбирает на странице элемент с именем класса `main` (это часть `$('.main')`) и затем прибавляет к окончанию этого элемента содержимое переменной `message`. Страница содержит элемент `div` с классом `main`, так что этот код просто добавляет строку «Вы навели указатель мыши на ссылку» к окончанию этого элемента `div` каждый раз, когда посетитель наводит курсор на ссылку. (См. раздел «Добавление содержимого на страницу» главы 4, чтобы освежить свои знания о jQuery-функции `append()`.)

3. **Сохраните страницу, просмотрите ее в браузере и наведите указатель мыши на любую ссылку.**

   Каждый раз при наведении указателя мыши на ссылку, на страницу добавляется абзац (рис. 5.3). Теперь вы добавите последний фрагмент кода: при щелчке по кнопке формы на странице, браузер изменит текст на этой кнопке.

4. **Наконец, добавьте фрагмент, выделенный полужирным шрифтом, чтобы ваш код выглядел следующим образом:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
$('html').dblclick(function() {
alert('ой');
}); // конец события double click
$('a').mouseover(function() {
```

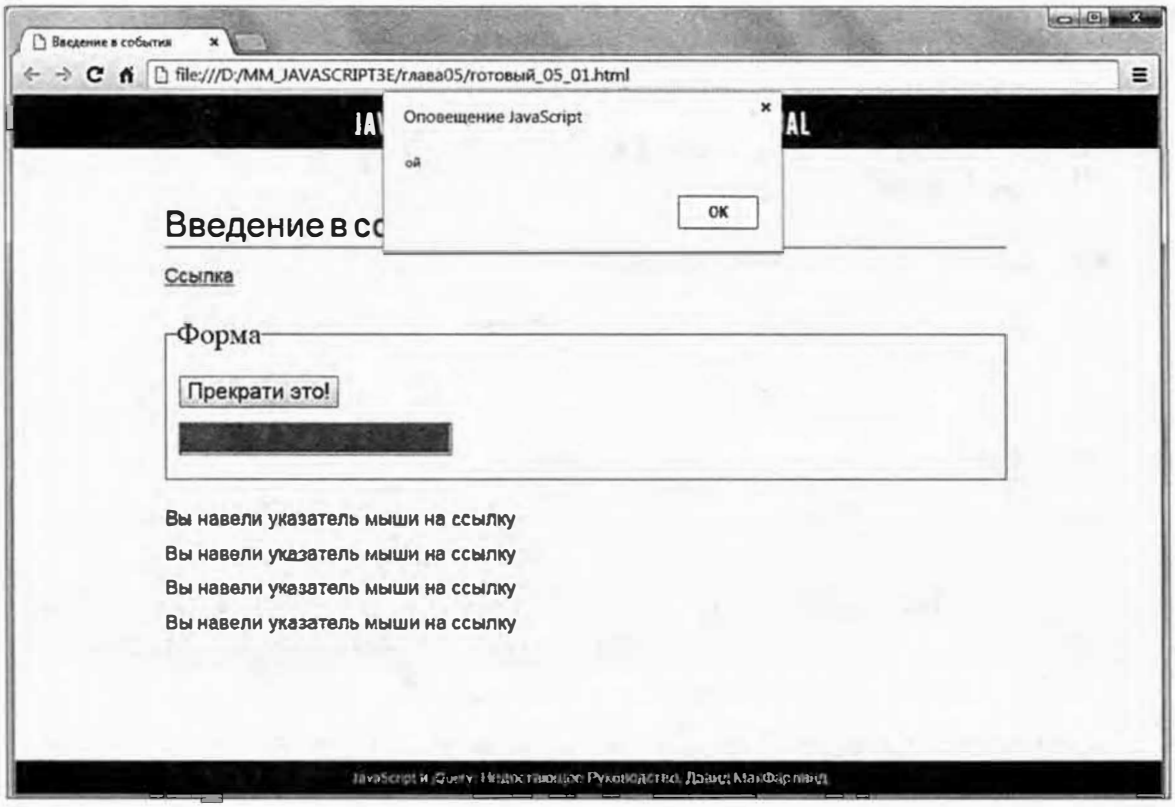

Рис. 5.3. С помощью библиотеки jQuery вы легко можете заставить свои веб-страницы реагировать на действия пользователя, например, открывать окно оповещения при двойном щелчке по странице, добавлять текст на страницу в ответ на наведение указателя мыши на ссылку или щелчок по кнопке формы

```
var message = "<p> Вы навели указатель мыши ↵
на ссылку </p>";
$('.main').append(message);
}); // конец события mouseover
$('#button').click(function() {
$(this).val("Прекрати это!");
}); // конец события click
}); // конец функции ready
</script>
```

Вы должны понимать основы: код `$('#button')` выбирает элемент с идентификатором `button` (в данном случае кнопка формы) и добавляет ему событие `click`, поэтому, когда кто-то щелкает по кнопке, что-нибудь происходит. В данном примере на кнопке появляется фраза «Прекрати это!». Вот, как код внутри анонимной функции это делает:

В разделе «Ключевые слова `this` и `$(this)`» главы 4 вы узнали, как использовать фрагмент кода `$(this)` внутри цикла jQuery.

В случае с событиями все точно так же: `$(this)` относится к элементу, реагирующему на событие, то есть к элементу, который вы выбираете и к которому прикрепляете событие. В данном случае — это кнопка формы. (Вы узнаете о функции `val()` библиотеки jQuery в разделе «Структура форм» главы 8, а если вкратце, то вы используете ее для чтения или изменения значения элемента формы. В данном примере передача строки «Прекрати это!» функции `val()` устанавливает значение кнопки на «Прекрати это!».)

5. **Сохраните страницу, просмотрите ее в браузере и щелкните по кнопке формы.** Текст на кнопке должен мгновенно измениться (рис. 5.3). В качестве дополнительного упражнения добавьте код для изменения цвета фона текстового поля на красный при щелчке по нему. Вот подсказка: вам нужно (а) выбрать текстовое поле; (б) использовать событие `focus()` (см. раздел «Структура форм» главы 8); (в) использовать ключевое слово `$(this)` (как в шаге 12) для обращения к текстовому полю внутри анонимной функции; (г) использовать функцию `.css()` (см. раздел «Чтение и изменение свойств CSS» главы 4) для изменения цвета фона текстового поля. Вы можете найти законченную версию страницы в файле *готовый_05_01.html* в папке *глава05*.

Больше концепций для событий jQuery

Поскольку события — это важный элемент обеспечения интерактивности веб-страницы, jQuery включает некоторые специфичные для данной библиотеки функции, которые упрощают программирование и делают ваши страницы более интерактивными.

Ожидание загрузки HTML-кода

Когда страница загружается, браузер пытается немедленно запустить все обнаруженные сценарии. Таким образом, сценарии, находящиеся в разделе заголовка страницы, могут быть выполнены до окончания загрузки страницы — подобный пример вы видели в викторине о лунах, где страница оставалась пустой до тех пор, пока сценарий не закончил задавать вопросы. К сожалению, из-за этого феномена часто возникают проблемы. Поскольку многие программы на языке JavaScript подразумевают манипулирование содержимым веб-страницы — ото-

бражение всплывающего сообщения при щелчке по ссылке, сокрытие определенных элементов веб-страницы, окрашивание строк таблицы и т. д. — в итоге будут возникать ошибки, если ваша программа попытается управлять элементами, которые еще не загрузились и не отображаются в браузере.

Самый распространенный способ решения данной проблемы — это использование события загрузки, чтобы дождаться, пока страница полностью загрузится и отобразится, прежде чем выполнять код JavaScript. К сожалению, задержка с выполнением кода JavaScript до тех пор, пока не загрузится страница, может привести к странным результатам. Событие загрузки запускается только после того, как загрузятся все файлы веб-страницы, то есть все изображения, ролики, внешние таблицы стилей и т. д. В результате, если на странице много графики, посетитель может любоваться пустой страницей несколько секунд, пока не загрузятся все рисунки и не начнется выполнение кода JavaScript. Если код JavaScript вносит на страницу много изменений, например, оформляет строки таблицы, скрывает видимые в данный момент меню или даже контролирует макет страницы, посетитель вдруг увидит, как страница меняется прямо у него на глазах.

К счастью, на помощь приходит библиотека jQuery. Вместо того чтобы полагаться на событие загрузки для запуска программы JavaScript, jQuery предлагает специальную функцию `ready()`, которая ожидает окончания загрузки HTML-кода, а потом запускает сценарии страницы. Таким образом, JavaScript может непосредственно манипулировать веб-страницей без длительного ожидания загрузки изображений или роликов (эта на самом деле сложная и полезная возможность — еще одна причина пользоваться библиотекой JavaScript).

Вы уже работали с функцией `ready()` в некоторых руководствах данной книги. Основная ее структура такова:

```
$(document).ready(function() {
//здесь находится ваш код
});
```

Как правило, весь программный код находится внутри этой функции, которая настолько важна, что вам следует задействовать ее на всех страницах, где вы используете библиотеку jQuery. Вы должны включить ее только однажды, это обычно первая и последняя строки вашего кода. Необходимо поместить эту функцию между тегами элемента `script`

(в конце концов, это код JavaScript) и строго после элемента `script`, добавляющего на страницу саму библиотеку jQuery.

Итак, в контексте полной веб-страницы функция будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Заголовок страницы</title>
<script src="js/jquery.js"></script>
<script>
$(document).ready(function( ) {
// весь ваш код JavaScript размещается здесь.
}); // конец функции ready()
</script>
</head>
<body>
Контент веб-страницы...
</body>
</html>
```

СОВЕТ

Поскольку функция `ready()` используется практически каждый раз при добавлении библиотеки jQuery на страницу, ее можно записывать кратко. Удалите часть `$(document).ready` и введите:

```
$(function() {
  //некие действия
});
```

Альтернатива для функции `$(document).ready()`

Помещение функции `$(document).ready()` в раздел заголовка HTML-документа (`<head>`) позволяет отложить выполнение кода JavaScript до загрузки HTML-кода. Однако для этого существует и другой способ: можно поместить код JavaScript после кода HTML. Напри-

мер, многие веб-разработчики просто добавляют свой код JavaScript непосредственно перед закрывающим тегом </body>:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Заголовок страницы</title>
</head>
<body>
Контент веб-страницы...
<script src="js/jquery.js"></script>
<script>
// весь ваш код JavaScript размещается здесь.
</script>
</body>
</html>
```

В данном случае в использовании функции `$(document).ready()` нет необходимости, поскольку к тому времени, как сценарий загрузится, документ будет готов. Этот подход имеет ряд важных преимуществ. Во-первых, вам не нужно вводить дополнительный код для включения функции `.ready()`. Во-вторых, загрузка и запуск кода JavaScript замедлит работу веб-браузера до тех пор, пока сценарий не загрузится и не прекратит выполняться. Если вы используете много внешних файлов JavaScript, на загрузку которых уходит некоторое время, то ваша веб-страница отобразится не сразу. Посетителям вашего сайта может показаться, что загрузка страницы происходит очень медленно.

В блогах, посвященных веб-дизайну, вы можете прочитать, что расположение сценариев в нижней части страницы является правильным способом добавления кода JavaScript. Тем не менее у этого подхода есть и минусы. В некоторых случаях код JavaScript, добавленный на страницу, оказывает сильное воздействие на ее внешний вид. Например, вы можете использовать код JavaScript, чтобы полностью перерисовать сложную HTML-таблицу, чтобы ее легче было просматривать. Вы также можете придать тексту страницы замечательный вид (**letteringjs.com**).

В этих случаях, если прежде чем загрузить jQuery и выполнить код JavaScript вы будете ждать, пока загрузится и отобразится HTML-код, посетители вашего сайта могут видеть, как страница меняется прямо у них на глазах (трансформируясь под воздействием кода JavaScript). Это изменение может привести в замешательство. Кроме того, если вы создаете веб-приложение, которое не работает без кода JavaScript, то нет никакого смысла показывать вашим посетителям HTML-код до окончания загрузки кода JavaScript. В конце концов, до окончания загрузки кода JavaScript кнопки, виджеты и инструменты интерфейса вашего веб-приложения, созданные на его основе, являются просто бесполезными фрагментами HTML-кода.

Таким образом, ответ на вопрос о том, где следует размещать код JavaScript, не является однозначным. В некоторых случаях ваш сайт покажется более отзывчивым, если вы поместите код JavaScript после кода HTML, а иногда, — если поместите его перед ним. К счастью, благодаря кэшированию, производящемуся браузером, как только одна страница вашего сайта загрузит необходимые файлы JavaScript, остальные страницы сразу же получат доступ к файлам в кэше браузера, и не будут тратить время на их повторную загрузку. Другими словами, не переживайте: если вы считаете, что ваша веб-страница отображается не достаточно быстро, то вы можете попробовать переместить сценарии в конец страницы. Если это поможет, воспользуйтесь этим подходом. Однако во многих случаях не имеет значения, используете вы функцию `.ready()` в верхней части страницы или нет.

ПРИМЕЧАНИЕ

При создании веб-страницы на компьютере и ее тестировании непосредственно в веб-браузере вы не столкнетесь с описанными в этом разделе проблемами. Только тогда, когда вы размещаете свой сайт на веб-сервере, и вам требуется загрузить сценарий и файлы страниц через медленное интернет-соединение, у вас могут возникнуть проблемы, связанные со скоростью загрузки и отображения веб-страницы.

События наведения и смещения указателя мыши

События наведения и смещения указателя мыши часто используются вместе. Например, когда вы наводите курсор на кнопку, появляется меню, а когда смещаете его — меню исчезает. Поскольку использование

этих событий в паре обычно, библиотека jQuery предлагает краткий способ обращения к обоим. Функция `hover()` работает, как и любое другое событие, кроме того, что вместо принятия одной функции в качестве аргумента, оно принимает две функции. Первая функция выполняется при наведении указателя мыши на элемент, а вторая — при смещении с него. Основная структура выглядит так:

```
$('#селектор').hover(функция1, функция2);
```

Вы часто будете использовать функцию `hover()` с двумя анонимными функциями. Данный код может выглядеть несколько странно; следующий пример его разъяснит. Допустим, кто-то наводит указатель мыши на ссылку с идентификатором `menu` и вы хотите, чтобы появился (на данный момент невидимый) элемент `div` с идентификатором `submenu`. Когда курсор сдвигают со ссылки, подменю снова оказывается скрытым. Для осуществления этого можно использовать функцию `hover()`:

```
$('#menu').hover(function() {
$('#submenu').show();
}, function() {
$('#submenu').hide();
}); //окончание события hover
```

Чтобы упростить чтение инструкции, содержащей много анонимных функций, поместите каждую функцию на отдельную строку. Итак, чуть более простая для чтения версия кода, приведенного выше, будет выглядеть так:

```
$('#menu').hover(
function() {
$('#submenu').show();
},//окончание события mouseover
function() {
$('#submenu').hide();
}//окончание события mouseout
); //окончание события hover
```

На рис. 5.4 показано, как работает этот код при наведении и смещении указателя мыши.

Рис. 5.4. JQuery-функция `hover()` позволяет назначать две функции сразу. Первая функция запускается, когда указатель мыши наводится на элемент, а вторая — когда он покидает элемент

Если метод анонимной функции кажется вам слишком запутанным, вы по-прежнему можете использовать старые добрые именованные функции (см. раздел «Функции: многократное использование кода» главы 3).

Создайте именованную функцию для выполнения при запуске события наведения указателя на элемент; затем создайте функцию для выполнения при запуске события смещения указателя мыши с элемента, и, наконец, передайте имена двух этих функций функции `hover()`. Итак, вы можете переписать приведенный выше код следующим образом:

```
function showSubmenu() {
$('#submenu').show();
}
function hideSubmenu() {
$('#submenu').hide();
}
$('#menu').hover(showSubmenu, hideSubmenu);
```

Если данная техника кажется вам проще, пользуйтесь ею. Между этими способами нет особой разницы. Просто некоторым программистам нравится, что при использовании анонимной функции можно сохранять весь код в одной инструкции вместо разделения его на несколько отдельных инструкций.

ПРИМЕЧАНИЕ

Версии библиотеки jQuery до 1.9 включали очень полезную функцию `toggle()`, которая работала точно так же, как событие `hover()`, за исключением того, что вместо реакции на события наведения и смещения курсора она реагировала на щелчки кнопкой мыши. Первый щелчок запускал первую функцию, второй — вторую функцию. Другими словами, вы могли переключать состояния с помощью щелчков. Например, вы могли отображать элемент страницы первым щелчком и закрывать его вторым. Поскольку функция `toggle()` больше не является частью jQuery, вы узнаете, как обеспечивать подобную функциональность в руководстве «Создание страницы ЧаВо на практике» далее в этой главе.

Объект события

В момент запуска события браузер записывает информацию о нем и сохраняет его в так называемом *объекте события*. Объект события содержит данные, собранные в момент, когда произошло событие, такие, как координаты указателя мыши по горизонтали и вертикали, элемент, который инициировал событие, была ли нажата при запуске события клавиша **Shift**.

В библиотеке jQuery объект события доступен для функции, созданной для обработки данного события. Объект передается функции как аргумент, и, чтобы получить к нему доступ, вы всего лишь добавляете к функции имя параметра. Например, вы хотите найти координаты указателя мыши в момент, когда происходит щелчок в каком-либо месте страницы.

```
$(document).click(function(evt) {
var xPos = evt.pageX;
var yPos = evt.pageY;
alert('X:' + xPos + ' Y:' + yPos);
}); // окончание события click
```

Здесь важна переменная `evt`. Когда вызывается функция (при щелчке кнопкой мыши в окне браузера), объект события сохраняется в переменной `evt`. В пределах тела функции вы можете получить доступ к различным свойствам объекта события, используя точечный синтаксис, например, `evt.pageX` возвращает расположение указателя мыши по горизонтали (или количество пикселов от левого края окна).

ПРИМЕЧАНИЕ

В данном примере `evt` — это имя переменной, присвоенное программистом. Это не специальное ключевое слово языка JavaScript, а просто переменная для сохранения объекта события. Вы можете использовать любое имя, например, `event` или просто `e`.

Объект события имеет различные свойства и (к несчастью) их список в каждом браузере может быть разным. В табл. 5.1 перечислены некоторые общие свойства.

Табл. 5.1. Каждое событие создает объект события с различными свойствами, к которым вы можете получить доступ с помощью функции, работающей с событием

Свойство события	Описание
`pageX`	Расстояние (в пикселах) от указателя мыши до левого края окна браузера
`pageY`	Расстояние (в пикселах) от указателя мыши до верхнего края окна браузера
`screenX`	Расстояние (в пикселах) от указателя мыши до левого края монитора
`screenY`	Расстояние (в пикселах) от стрелки мыши до верхнего края монитора
`shiftKey`	Является *истинным*, если клавиша **Shift** была нажата, когда происходило событие
`which`	Используется с событием нажатия клавиши для определения числового кода нажатой клавиши (см. совет ниже)
`target`	Объект, бывший целью события; например, для события `click()` — это элемент, по которому щелкнули кнопкой мыши
`data`	Объект jQuery, использованный с функцией `bind()` для передачи данных функции, управляющей событием (см. раздел «Профессиональное управление событиями» далее в этой главе)

СОВЕТ

Если вы попытаетесь получить доступ к свойству `which` объекта события, соответствующего событию `keypress()`, вы получите числовой код нажатой клавиши. Если вы хотите узнать, какая конкретно клавиша была нажата (a, K, 9 и т. д.), вы должны преобразовать значение свойства `which` с помощью метода JavaScript, который превращает числовой код клавиши в конкретную букву или символ клавиатуры:

```
String.fromCharCode(evt.which)
```

Отмена обычного поведения событий

Некоторые элементы HTML-кода имеют заранее запрограммированные реакции на события. Например, при щелчке по ссылке обычно загружается новая веб-страница; щелчок по кнопке подтверждения формы отсылает данные на веб-сервер для обработки. Иногда вы хотите, чтобы браузер повел себя не так, как обычно. Допустим, при подтверждении формы (событие `submit()`) вы можете пожелать остановить ее отправку, пока посетитель не внесет в форму недостающих важных данных.

Функция `preventDefault()` позволяет отменить нормальное поведение браузера. Она является частью объекта события (см. предыдущий раздел), поэтому доступ к ней осуществляется в рамках функции, управляющей событием. Допустим, на странице есть ссылка с идентификатором `menu`. Она указывает на другую страницу меню (чтобы посетитель, у которого отключен JavaScript, мог попасть на эту страницу меню). Однако вы добавили умный код JavaScript, и если посетитель щелкает по ссылке, меню появляется на той же странице. Но, как правило, браузер все же проследует по ссылке, поэтому вам необходимо отменить его обычное поведение следующим образом:

```
$('#menu').click(function(evt){
// "Умный" JavaScript-код вставляется сюда
evt.preventDefault(); // не переходи по ссылке
});
```

Другой вариант — это просто возвращение значения «ложь» как результат выполнения функции события. Например, следующий код функционально аналогичен коду, приведенному выше:

```
$('#menu').click(function(evt){
// "Умный" JavaScript-код вставляется сюда
return false; // не переходи по ссылке
});
```

Удаление событий

Иногда вам может понадобиться удалить событие, которое вы ранее присвоили элементу. jQuery-функция `off()` позволяет это сделать.

Чтобы использовать ее, необходимо создать объект jQuery с элементом, с которого вы хотите удалить событие. Затем добавьте функцию `off()`, передав ей имя события в виде строки. Например, если вы хотите отменить реакцию всех элементов с классом `tabButton` на событие щелчка кнопкой мыши, то вы можете использовать следующий код:

```
$('.tabButton').off('click');
```

Ниже приведен пример краткого сценария, иллюстрирующего, как работает функция `off()`.

```
1     $('a').mouseover(function() {
2           alert('Вы навели на меня указатель мыши! ');
3     });
4     $('#disable').click(function() {
5           $('a').off('mouseover');
6     });
```

Строки 1—3 добавляют событию наведения указателя мыши функцию для всех ссылок на странице (элементы `a`). Если навести курсор на ссылку, появится окно с сообщением «Вы навели на меня указатель мыши!» Однако поскольку постоянное появление окна с сообщением будет раздражать, строки 4—6 позволяют посетителю отключить оповещение. Когда пользователь щелкает по элементу с идентификатором `disable` (например, по кнопке формы), событие наведения указателя мыши удаляется со всех ссылок и сообщение больше не появляется.

Если вы хотите удалить все события с элемента, не передавайте функции `off()` никаких аргументов. Например, чтобы удалить все события (`mouseover`, `click`, `dblclick` и т. д.) с кнопки отправки, вы можете написать следующий код:

```
$('input[type="submit"]').off();
```

Однако это довольно тяжеловесный способ, и в большинстве случаев вы не захотите удалять с элемента все обработчики событий.

ПРИМЕЧАНИЕ

Для получения более подробной информации о jQuery-функции `off()` посетите сайт: **api.jquery.com/off/**.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Остановка события

Модель событий позволяет событию распространяться не только на элемент, которому это событие было присвоено изначально. Например, представьте, что вы присвоили помощника событиям щелчка по какой-либо ссылке. Когда вы щелкаете по ней, запускается событие щелчка и выполняется функция. Однако на этом событие не заканчивается. Каждый из элементов-предков (тегов, заключающих элемент, по которому совершается щелчок) также может отреагировать на этот же щелчок. Так, если вы присвоили помощника события щелчка элементу `div`, внутри которого находится рассматриваемая ссылка, функция события этого элемента `div` также будет выполняться.

Данная концепция, известная как *всплытие события* (event bubbling), означает, что на одно и то же событие могут реагировать несколько элементов. Вот еще один пример: допустим, вы добавляете событие щелчка к изображению. При щелчке по этой картинке ее заменяет новая. Изображение находится внутри элемента `div`, которому вы также присвоили событие щелчка. В данном случае окно оповещения появляется при щелчке по элементу `div`. Теперь, когда вы щелкаете по изображению, будут выполняться обе функции. Иными словами, несмотря на то, что вы щелкаете по изображению, элемент `div` также получает событие щелчка.

Возможно, такая ситуация встретится вам нечасто, но если это случится, результат может привести вас в замешательство. Предположим, что в предыдущем примере вы не желаете, чтобы при щелчке по изображению с элементом `div` происходило что-либо. В данном случае вы должны предотвратить присвоение события элементу `div`, не затрагивая события функции, управляющей событием щелчка по изображению. Иными словами, при щелчке по картинке, функция, присвоенная этому событию, должна загрузить новое изображение, а затем — остановить событие щелчка.

Библиотека jQuery предлагает функцию `stopPropagation()`, препятствующую присвоению события любому элементу-предку. Эта функция является методом объекта события (см. раздел «Объект события»), так что вы можете получить к ней доступ в рамках функции, обрабатывающей событие:

```
$('#theLink').click(function(evt) {
// некие действия
evt.stopPropagation(); //останавливает событие
});
```

Профессиональное управление событиями

Вы можете прожить долгую и счастливую жизнь программиста, используя методы событий библиотеки jQuery и концепции, обсуждавшиеся выше. Но если вы действительно хотите взять максимум от использования техник управления событиями jQuery, то вам необходимо изучить функцию `on()`.

ПРИМЕЧАНИЕ

Если у вас все еще болит голова от предыдущего раздела, можете сразу перейти к руководству в конце данной главы, пока у вас не наберется чуть больше опыта в обращении с событиями.

Метод `on()` — это более гибкий способ работы с событиями по сравнению со специфичными jQuery-функциями, такими как `click()` или `mouseover()`. Он позволяет не только указывать событие и функцию для реакции на него, но и передавать дополнительные данные для использования функцией, обрабатывающей событие.

Это дает возможность различным элементам и событиям (например, щелчок по ссылке или наведение указателя мыши на изображение) передавать различную информацию одной и той же функции, обрабатывающей события, иными словами, одна и та же функция может действовать по-разному в зависимости от запускаемого события.

ПРИМЕЧАНИЕ

По мере развития библиотеки jQuery имена функций, используемых для добавления и удаления событий с элементов, сильно изменились. При чтении более ранних изданий данной книги или блогов вы можете наткнуться на такие функции, как `bind()`,`live()` и `delegate()`. Все они были заменены одной функцией `on()`, предназначенной для добавления событий к элементам. Кроме того, функция `off()` заменила функцию `unbind()`, которая использовалась для удаления событий с элементов.

Формат функции `on()` следующий:

```
$('#селектор').on('событиемыши', селектор, данные,
имяфункции);
```

Первый аргумент — это строка, содержащая имя события (например, щелчок, наведение курсора и любые другие события, перечисленные в разделе «События мыши»). Второй аргумент не является обязательным, так что нет необходимости предоставлять значение для него при использовании функции `on()`. Если вы все-таки решите передать аргумент, то он должен представлять собой допустимый селектор, например, `tr`, `.callout` или `#alarm`.

ПРИМЕЧАНИЕ

Вы можете использовать второй аргумент для того, чтобы применить событие к другому элементу в рамках выделенного элемента. Эта техника называется делегированием событий, и вы узнаете о ней далее.

Третий аргумент — это данные, которые вы желаете передать функции (в форме литерала объекта либо переменной, содержащей литерал объекта). Литерал объекта (или объектная константа) (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4) — это список имен свойств и их значений:

```
{
firstName : 'Иван',
lastName : 'Иванов'
}
```

Вы можете сохранить литерал объекта в переменной следующим образом:

```
var linkVar = {message:'Привет от ссылки'};
```

Четвертый аргумент, передаваемый функции `on()`, — другая функция, та самая, которая выполняет какое-либо действие при запуске события. Это может быть или анонимная функция, или функция с именем, иначе говоря, данная часть не отличается от способа использования обычных событий jQuery, описанного в разделе «Использование событий: способ jQuery».

ПРИМЕЧАНИЕ

Передача данных при использовании функции `on()` не является обязательной. Если вы хотите использовать функцию `on()`, просто

чтобы присоединить событие и функцию, опустите аргумент с данными:

```
$('селектор').on('событиемыши', имяфункции);
```

Функционально данный код идентичен следующему:

```
$('селектор').click(имяфункции);
```

Предположим, вы хотели бы создать всплывающее окно оповещение в ответ на событие, но сообщение в окне должно быть разным в зависимости от того, какой элемент инициировал событие. Один из способов сделать это — создать переменные, хранящие различные объектные константы, а затем передать переменные функции `on()`:

```
var linkVar = {message:'Привет от ссылки'};
var pVar = {message:'Привет от абзаца'};
function showMessage(evt) {
alert(evt.data.message);
}
$('a').on('mouseover',linkVar,showMessage);
$('p').on('click',pVar,showMessage);
```

На рис. 5.5 показано, как работает этот код. Он создает две переменные: `linkVar` в первой и `pVar` во второй строке. Каждая переменная содержит объектную константу с одним и тем же именем свойства (`message`), но с различным текстом. Функция `showMessage()` принимает объект события (см. раздел «Объект события») и сохраняет его в переменной `evt`. Эта функция запускает команду `alert()`, отображая свойство `message` (которое само является свойством свойства `data` объекта события). Помните, что `message` — это имя свойства, определенного в объектной константе.

Другие способы использования функции `on()`

JQuery-функция `on()` обеспечивает гибкость в процессе программирования. В дополнение к техникам, описанным в предыдущем разделе, она позволяет вам привязывать два или более событий к одной и той же функции. Например, вы пишете программу, которая заставляет крупное изображение появляться на экране при щелчке по эскизу (распространенный lightbox-эффект, который можно наблюдать на тысячах сайтах).

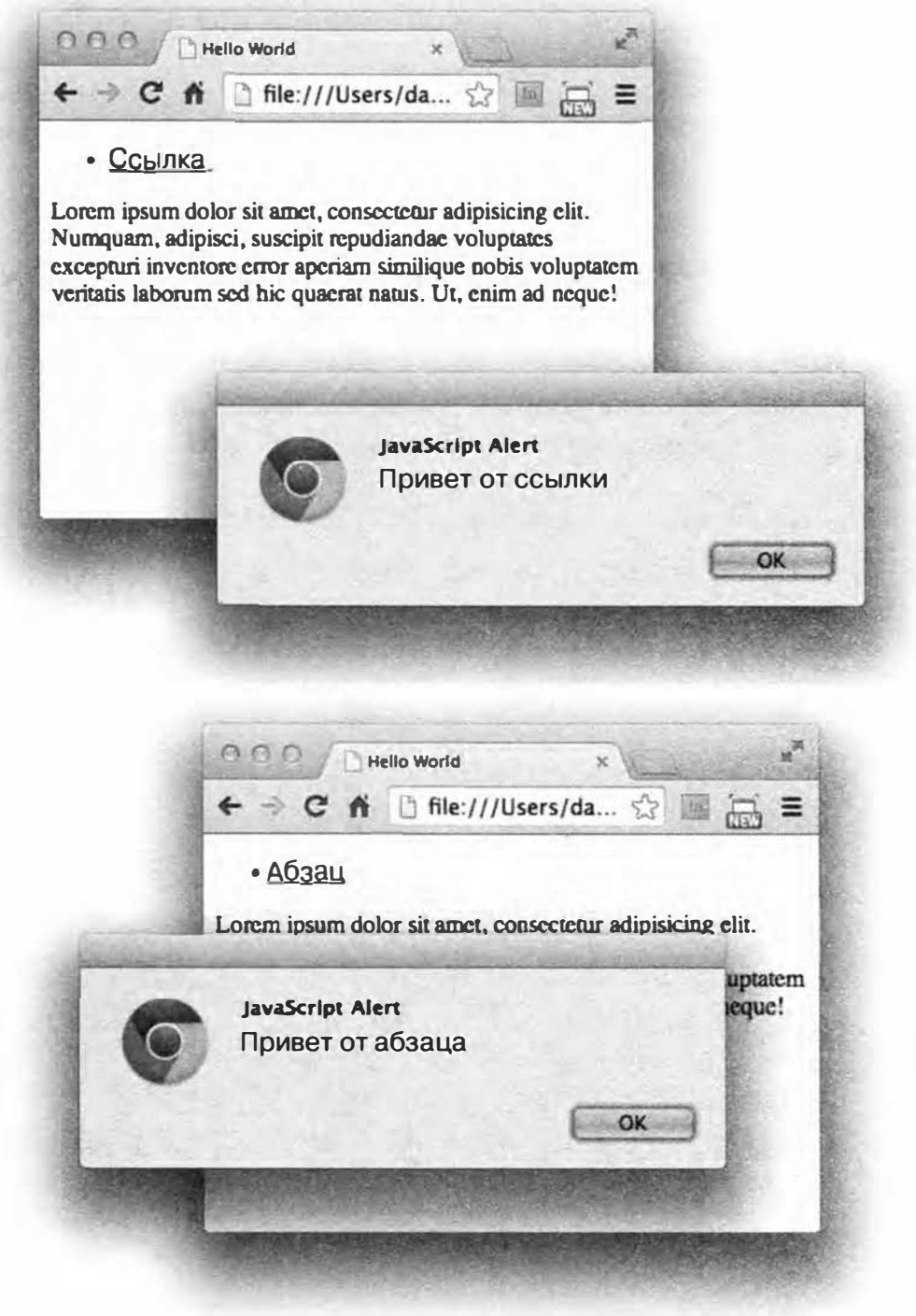

Рис. 5.5. jQuery-функция `on()` позволяет передавать данные функции, отвечающей на событие. Таким образом, вы можете использовать одно и то же имя функции для нескольких различных элементов (даже с различными типами событий), позволяя функции использовать данные, специфичные для каждого помощника событий

Вам нужно, чтобы крупное изображение исчезало, когда посетитель либо щелкает в любом месте страницы, либо нажимает какую-либо клавишу на клавиатуре (если вы предусмотрите обе возможности, ваша

программа сможет реагировать на действия пользователей, предпочитающих использовать клавиатуру вместо мыши и наоборот). Вот пример кода, который позволяет это сделать:

```
$(document).on('click keypress', function() {
$('#lightbox').hide( );
}); // завершение функции on
```

Важная часть кода здесь — это `'click keypress'`. Предоставляя несколько имен событий, разделенных пробелом, вы приказываете библиотеке jQuery выполнить анонимную функцию при запуске любого события из списка. В данном случае — при запуске либо события `click`, либо события `keypress`. К тому же, если вы хотите присоединить несколько событий, каждое из которых запускает различные действия, вам не обязательно использовать функцию `on()` несколько раз. Другими словами, если вы хотите, чтобы произошло одно событие при щелчке по элементу и другое — при наведении на этот элемент указателя мыши, вы можете написать следующий код:

```
$('#theElement').on('click', function() {
// какое-либо интересное действие
}); // окончание функции on
$('#theElement').on('mouseover', function() {
// еще какое-нибудь интересное действие
}); // окончание функции on
```

Вы можете добиться того же результата, передав функции `on()` объектную константу (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), состоящую из имени события, за которой следуют двоеточие и анонимная функция. Вот переписанный код, вызывающий функцию `on()` только раз и передающий ей объектную константу (выделена полужирным шрифтом):

```
$('#theElement').on({
'click' : function() {
// какое-либо интересное действие
}, // окончание функции click
'mouseover' : function() {
// какое-либо интересное действие
}; // окончание функции mouseover
}); // окончание функции on
```

Делегирование событий с помощью функции on()

Как отмечалось ранее, метод `on()` может принимать второй аргумент, который является еще одним селектором:

```
$('#селектор').on('событиемыши', селектор, данные, имяфункции);
```

Второй аргумент может быть любым действительным селектором jQuery, например, селектором идентификатора, класса, элемента и т. д. Передача селектора функции `on()` может значительно повлиять на ее работу. Без передачи селектора событие применяется к первоначальному селектору — `$('#selector')` — в приведенном выше примере. Допустим, вы добавили на страницу такой код:

```
$('li').on('click', function() {
$(this).css('text-decoration': 'line-through');
}); // окончание on
```

Этот код делает текст перечеркнутым в каждом элементе `li`, по которому щелкает посетитель. Помните, что в данном случае фрагмент `$(this)` относится к элементу, обрабатывающему событие, то есть к элементу `li`, по которому производится щелчок. Другими словами, событие `click` «связано» с элементом `li`. В большинстве случаев это именно то, что вам нужно сделать — определить функцию, которая выполняется, когда посетитель взаимодействует с конкретным элементом. Вы можете сделать так, чтобы функция выполнялась, когда посетитель щелкает по ссылке, наводит указатель мыши на пункт меню, отправляет форму и т. д.

ПРИМЕЧАНИЕ

Если вы все еще не уверены в том, что означает фрагмент `$(this)`, обратитесь к разделу «Ключевые слова `this` и `$(this)`» главы 4, а также к разделу «Использование событий на практике» ранее в этой главе, чтобы получить более полную информацию о том, как ключевое слово `$(this)` работает с обработчиками событий.

Тем не менее у этого метода присоединения обработчиков событий к элементам существует одна проблема: он работает только в том случае, если элемент уже присутствует на странице. Если вы динамически добавляете HTML-код после добавления такого обработчика событий, как

`click()`, `mouseover()` или `on()`, то к этим новым элементам не будут присоединены никакие обработчики событий. Далее приведен пример для объяснения данной концепции.

Представьте, что вы создали приложение, которое позволяет посетителю создавать и управлять списком дел. При первой загрузке приложения список пуст (см. изображение 1 на рис. 5.6). Посетитель может заполнить текстовое поле и щелкнуть по кнопке «Добавить задание», чтобы добавить пункты в список (см. изображение 2 на рис. 5.6). После выполнения задания посетитель может щелкнуть по пункту, чтобы отметить задание как выполненное (см. изображение 3 на рис. 5.6).

Для этого необходимо добавить событие `click` к каждому элементу `li`, чтобы при щелчке по нему стиль форматирования текста каким-либо образом менялся. На рис. 5.6 показано, что текст, соответствующий выполненному заданию, потускнел и стал перечеркнутым. Проблема в том, что при загрузке страницы на ней нет элементов `li`, так что добавление обработчика события `click` к каждому такому элементу не приведет к каким-либо изменениям. Другими словами, код, приведенный на предыдущей странице, не будет работать.

ПРИМЕЧАНИЕ

Подробнее о делегировании событий jQuery вы можете узнать по адресу: **api.jquery.com/on/**.

Вместо этого вам следует *делегировать* события, то есть, применить событие к родительскому элементу, находящемуся на более высоком уровне в цепочке (к элементу, который уже присутствует на странице), а затем отслеживать эти события на конкретных дочерних элементах. Поскольку событие применяется к уже существующему элементу, добавление новых дочерних элементов не повлияет на этот процесс. Другими словами, вы делегируете отслеживание событий уже существующему родительскому элементу. Более подробное описание вы можете найти в тексте в рамке далее в главе. А пока, чтобы код в данном примере заработал, вам нужно использовать функцию `on()` следующим образом:

```
$('ul').on('click', 'li', function() {
$(this).css('text-decoration': 'line-through');
}); // окончание функции on
```

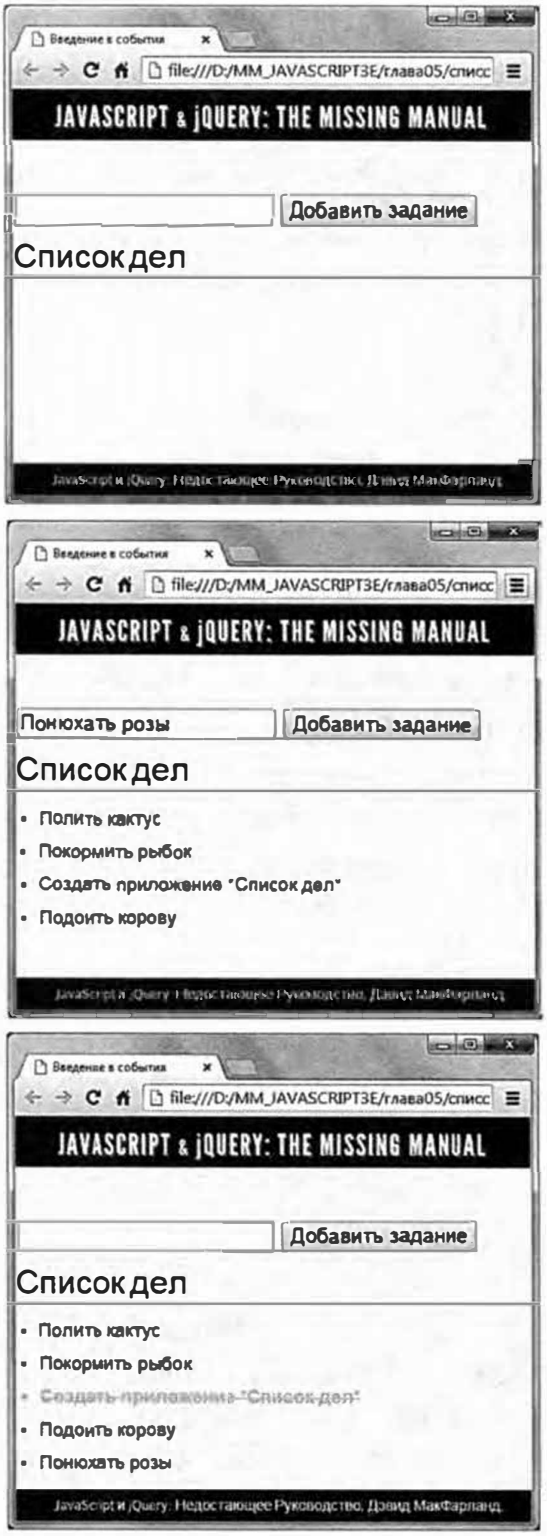

Рис. 5.6. Иногда вы будете писать код JavaScript, который динамически добавляет на страницу новый HTML-код. В данном примере посетители могут добавлять новые элементы (в данном случае задания) в неупорядоченный список. При первой загрузке в списке нет элементов, — только пустая пара тегов `<ul>` и `</ul>` (верхний рисунок). Когда посетитель вводит новое задание и щелкает по кнопке «Добавить задание», на страницу добавляются новые элементы `li` (средний рисунок). Чтобы отметить задание как выполненное, просто щелкните по соответствующему пункту списка (нижний рисунок). Вы найдете рабочий пример в файле *список_дел.html* в папке *глава05*

При создании страницы вы добавили пустой элемент `ul` в качестве контейнера для добавления каждого нового задания в элемент `li`. В результате при загрузке страницы один пустой элемент `ul` уже находится на месте. Затем выполнение приведенного выше кода добавляет в этот элемент функцию `on()`. Посетитель еще не добавил в список никаких пунктов, так что там пока отсутствуют элементы `li`. Тем не менее при добавлении селектора `'li'` в качестве второго аргумента функции `on()` вы заявляете о том, что хотите отследить события `click` не в `ul`, а в любом элементе `li`, присутствующем в этом неупорядоченном списке. Время добавления элемента `li` не имеет значения, поскольку отслеживанием событий занимается элемент `ul`.

Как делегирование событий влияет на объект `$(this)`

Как отмечалось ранее, объект `$(this)` относится к элементу, который в данный момент обрабатывается циклом (см. раздел «Ключевые слова `this` и `$(this)`» главы 4) или обработчиком событий (см. раздел «Использование событий на практике» ранее в этой главе). Обычно в обработчике событий объект `$(this)` относится к первоначальному селектору. Например:

```
$('ul').on('click', function() {
$(this).css('text-decoration': 'line-through');
}
```

В приведенном выше коде объект `$(this)` относится к элементу `ul`, по которому щелкает посетитель. Тем не менее при использовании делегирования событий первоначальный селектор больше не является элементом, с которым взаимодействует посетитель — он становится контейнером для элемента, по которому щелкает посетитель (наводит указатель мыши, применяет клавишу **Tab** и т. д.). Посмотрите на код для делегирования события еще раз:

```
$('ul').on('click', 'li', function() {
$(this).css('text-decoration': 'line-through');
}); // окончание функции on
```

В данном случае посетитель щелкает по элементу `li`, и именно он должен отреагировать на событие. Другими словами, элемент `ul` является просто контейнером, а функция обязана выполняться при щелчке по элементу `li`. Таким образом, в данном случае объект `$(this)` будет

относиться к элементу `li`, а приведенная выше функция станет перечеркивать текст в каждом элементе `li` при щелчке по нему.

Во многих случаях вы сможете обойтись совсем без делегирования. Тем не менее, если вам когда-нибудь понадобится добавить события в HTML-код, который отсутствует на странице при ее загрузке, то вам потребуется использовать данный подход. Например, в случае с технологией Ajax (глава 13) вы можете использовать делегирование событий для применения событий к HTML-контенту, который динамически добавляется на страницу веб-сервером.

ПРИМЕЧАНИЕ

В некоторых случаях вы можете применять делегирование событий просто для повышения эффективности кода JavaScript. Если вы добавляете большое количество элементов к одному и тому же обработчику событий, например, сотни ячеек таблицы, то часто лучше бывает делегировать событие элементу `table` следующим образом:

```
$('table').on('click', 'td', function () {
// код помещается здесь
});
```

Применяя событие к таблице, вы устраняете необходимость применять обработчики событий непосредственно к сотням или даже тысячам отдельных элементов, поскольку эта задача может поглотить всю память и вычислительную мощность браузера.

Создание страницы ЧаВо на практике

Раздел «Часто задаваемые вопросы» встречается на многих сайтах. Этот раздел помогает улучшать качество обслуживания клиентов с помощью непосредственного ответа на вопросы 24 часа в сутки 7 дней в неделю. К сожалению, большинство разделов ЧаВо представляют собой либо одну очень длинную страницу с вопросами и ответами, либо страницу с вопросами, которые ссылаются на отдельные страницы с ответами. Оба решения замедляют поиск ответов: в первом случае посетителю приходится прокручивать страницу в поисках вопроса и ответа на него, а во втором случае —ждать, пока загрузится новая.

В данном примере вы решите эту проблему, создав управляемую JavaScript страницу ЧаВо. Все вопросы будут видны при загрузке стра-

ницы, поэтому найти нужный вопрос не составит труда. Однако ответы окажутся скрыты — после щелчка по вопросу ответ появится на странице (рис. 5.7).

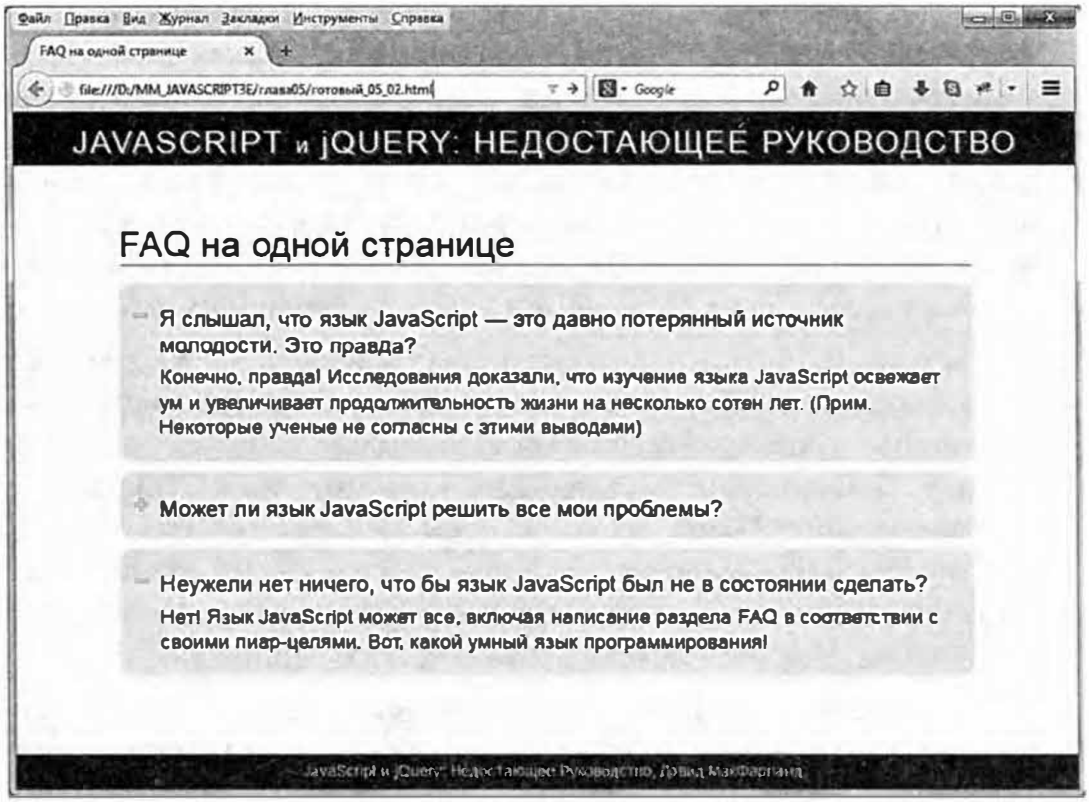

Рис. 5.7. С помощью нескольких строк на языке JavaScript вы можете отображать и скрывать элементы страницы, щелкая по ним кнопкой мыши

Обзор задачи

Для решения поставленной задачи код JavaScript должен обеспечить следующее.

- При щелчке по вопросу должен появиться соответствующий ответ.
- При щелчке по вопросу, ответ на который виден, ответ должен исчезнуть.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Делегирование событий — это магия?

Я понял основную идею делегирования событий, но как все-таки это работает?

Как говорилось ранее, событие «всплывает» по HTML-коду страницы. Когда вы щелкаете по ссылке внутри абзаца, событие `click` сна-

чала срабатывает на этой ссылке, затем на родительском элементе (абзаце), затем на элементе `body`, а затем на объекте `html`. Другими словами, событие, сработавшее на одном HTML-элементе, передается вверх каждому из его предков.

Этот факт может очень сильно помочь в случае с делегированием событий. Как говорилось ранее, иногда вам может потребоваться применить события к несуществующим HTML-элементам, например, к элементу списка дел, который появляется только после того, как страница загрузится и посетитель добавит его. Несмотря на то, что нельзя добавить событие `click` к отсутствующему элементу `li`, вы можете добавить событие `click` к уже присутствующему на странице родительскому элементу `ul`, или даже к элементу `div`, содержащему элемент `ul`.

Как уже упоминалось, у каждого события есть объект события, который содержит множество различных фрагментов информации. В данном случае самой важной информацией является свойство `target`. Данное свойство указывает на конкретный HTML-элемент, принимающий событие. Например, при щелчке по ссылке, эта ссылка является целевым элементом события `click`. Из-за всплывания событий родительский элемент может «услышать» событие, а затем определить, какой дочерний элемент являлся целевым.

Таким образом, в случае с делегированием событий вы способны заставить родительский элемент отслеживать события, например, заставить элемент `ul` отслеживать событие `click`. Затем он может установить, какой из элементов являлся целевым. Например, если целью был элемент `li`, то вы можете запустить конкретный фрагмент кода в ответ на эту ситуацию, в данном случае для перечеркивания текста задания в списке дел.

Кроме того, вам нужно, чтобы код JavaScript скрывал все ответы при загрузке страницы. Почему же не использовать CSS для сокрытия ответов? Например, установка значения свойства CSS `display` для ответов на `none` — это один из способов скрыть ответы. Проблема с данной техникой возникает в отношении посетителей, у которых отключен JavaScript: они не увидят ответов в ходе загрузки страницы и не смогут отобразить их, щелкая по вопросам. Чтобы сделать ваши страницы функциональными и для тех, у кого отключен JavaScript, лучше использовать код JavaScript для сокрытия контента страницы.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в главе 1 для справки о копировании файлов примеров.

Верстка кода

1. **В текстовом редакторе откройте файл *05_02.html* в каталоге *глава05*.**

   Код документа уже содержит ссылку на файл jQuery и функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery»). Сначала мы скроем все вопросы при загрузке страницы.

2. **Щелкните мышью в пустой строке после функции `$(document).ready()` и введите `$('.answer').hide();`.**

   Текст каждого ответа содержится в элементе `div`, которому присвоен класс *answer*. Данная строка кода выбирает каждый такой элемент `div` и скрывает его (функция `hide()` обсуждается в разделе «Основы отображения и сокрытия» главы 6).

   Сохраните страницу и откройте ее в браузере. Все ответы должны быть скрыты.

ПРИМЕЧАНИЕ

Элементы скрыты средствами JavaScript, а не CSS, поскольку у некоторых посетителей JavaScript может быть отключен. В этом случае они не увидят эффекта, который вы создаете в данном руководстве, однако у них, по крайней мере, будет возможность увидеть все ответы.

Следующим шагом является определение элементов, которым необходимо добавить обработчики событий. Поскольку ответ появляется, когда посетитель щелкает по вопросу, вы должны выбрать каждый вопрос в разделе FAQ. На данной странице каждый вопрос — это элемент `h2` в основном теле страницы.

3. **Нажмите клавишу Enter, чтобы создать новую строку, и добавьте в сценарий код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
$('.answer').hide();
$('.main h2')
```

```
}); // окончание функции ready()
</script>
```

Это основной дочерний селектор, используемый для нацеливания на каждый из элементов h2 внутри элемента с классом main (элементы h2 где-либо в другом месте страницы не будут затронуты). Теперь пришло время добавить обработчик события. Хороший кандидат — это событие щелчка, однако вашим запросам более точно отвечает вариант, при котором щелчок по вопросу либо отображает, либо скрывает ответ, то есть когда используется управляющая инструкция. Вам нужно проверить, не скрыт ли элемент div, следующий после элемента h2. Если скрыт, то его нужно отобразить, если нет, — то скрыть.

4. **Добавьте следующий выделенный полужирным шрифтом код, чтобы присоединить обработчик событий к элементам h2.**

```
$(document).ready(function() {
$('.answer').hide();
$('.main h2').click(function() {
}); // окончание click
}); // окончание ready()
```

Этот код добавляет событие click к элементам h2. Комментарий // окончание click не является обязательным, однако, как говорилось ранее, комментарии помогают разобраться в том, что делает конкретный набор символов });. После добавления этого фрагмента любой код, помещенный в рамки анонимной функции, станет выполняться всякий раз, когда посетитель будет щелкать по одному из элементов h2.

5. **В пустой строке внутри функции введите:**

```
var $answer = $(this).next('.answer');
```

Здесь вы создаете переменную $answer, которая будет содержать объект jQuery. Как говорилось ранее, объект $(this) относится к элементу, в данный момент реагирующему на событие, в данном случае, к конкретному элементу h2. Библиотека jQuery предусматривает несколько функций, облегчающих перемещение по структуре страницы. Функция .next() находит элемент, который следует непосредственно за текущим элементом. Другими словами, она находит элемент, следующий за элементом

h2. Вы можете уточнить этот поиск, передав дополнительный селектор функции .next(). Код .next('.answer') находит первый элемент после элемента h2, который также имеет класс answer.

Иначе говоря, вы сохраняете ссылку на элемент div, следующий непосредственно за элементом h2. Этот элемент div содержит ответ на вопрос. Вы сохраняете его в переменной, поскольку в функции вам потребуется несколько раз получить доступ к этому элементу: чтобы проверить, не скрыт ли ответ, отобразить, если он скрыт, и скрыть, если отображен. Каждый раз, когда вы получаете доступ к jQuery, используя код $(), вы даете браузеру команду запустить программный код внутри jQuery. Таким образом, код code $('this').next('.answer') заставляет библиотеку jQuery выполнить определенные действия. Вместо того чтобы снова и снова повторять одни и те же шаги, вы можете просто сохранить результат их выполнения в переменной, а затем использовать эту переменную для указания на элемент div, который вы хотите скрыть или отобразить.

Если вам нужно снова и снова использовать одни те же результаты работы библиотеки jQuery, хорошей идеей является сохранение этих результатов в переменной, которую вы можете многократно использовать. Это повышает эффективность программы, снимает с браузера часть лишней нагрузки и делает вашу веб-страницу более отзывчивой.

Переменная начинается с символа $ (например, $answer), это указывает на то, что вы сохраняете объект jQuery (результат выполнения функции $()).

Перед именем переменной не обязательно ставить символ $, это просто правило, распространенное среди программистов jQuery, которое они используют для того, чтобы указать на то, что они могут применять к этой переменной все замечательные функции, например, .hide().

ПРИМЕЧАНИЕ

Функция .next() — это одна из многих функций (методов) библиотеки jQuery, помогающих перемещаться по объектной модели страницы. Чтобы изучить и другие полезные функции, посетите страницу **docs.jquery.com/Traversing**. Кроме того, вы можете обратиться к разделу «Обход дерева DOM» главы 15.

6. **Добавьте в свой код пустое условие if/else следующим образом:**

```
$(document).ready(function() {
$('.answer').hide();
$('.main h2').click(function() {
var $answer = $(this).next('.answer');
if ( ) {
}else {
}
}); // окончание click
}); // окончание ready()
```

Опытные программисты обычно не вводят пустое условие `if/else` таким образом, однако в процессе обучения бывает полезно создавать код по одному фрагменту за раз. Итак, почему бы не проверить, является ли ответ скрытым в данное время.

7. **Введите `$(answer.is(':hidden')` в круглых скобках управляющей инструкции:**

```
$(document).ready(function() {
$('.answer').hide();
$('.main h2').click(function() {
var $answer = $(this).next('.answer');
if ($answer.is(':hidden')) {
} else {
}
}); // окончание click
}); // окончание ready()
```

Вы используете переменную `$answer`, созданную на шаге 5. Она содержит элемент с классом `answer`, появляющийся сразу после элемента `h2`, по которому щелкает посетитель. Помните, что это элемент `div`, содержащий ответ на вопрос в элементе `h2`.

Вы используете метод jQuery `is()`, чтобы проверить, не является ли конкретный элемент скрытым. Метод `is()` позволяет проверить,

соответствует ли текущий элемент конкретному селектору. Вы передаете функции любой селектор CSS или jQuery, и эта функция проверяет, соответствует ли объект данному селектору. Если да, функция возвращает значение `true`, если нет, — значение `false`. Это замечательно! Как вы знаете, управляющая инструкция требует наличия значения `true` или `false` (см. врезку «Возвращение логических значений» в главе 3).

Используемый в коде селектор `:hidden` является особым, присущим только jQuery селектором, который определяет скрытые элементы. В данном случае вы проверяете, не скрыт ли ответ. Если нет, вы можете отобразить его.

8. **Добавьте строку `$answer.slideDown();` в свою программу:**

```
$(document).ready(function() {
$('.answer').hide();
$('.main h2').click(function() {
var $answer = $(this).next('.answer');
if ($answer.is(':hidden')) {
$answer.slideDown();
} else {
}
}); // окончание click
}); // окончание ready()
```

`slideDown()` — это одна из jQuery-функций, предназначенных для создания анимации (вы узнаете об анимации подробнее в следующей главе). При ее использовании скрытые элементы отображаются на странице, как бы выезжая из-под отображенных. На данном этапе вы можете проверить результаты своего труда. Сохраните страницу и просмотрите ее в веб-браузере. Щелкните по одному из вопросов на странице. После этого под ним должен отобразиться ответ (если он не отобразится, дважды проверьте введенный вами код и обратитесь к советам по устранению ошибок, приведенным в конце главы 1).

Если вы посмотрите на страницу, то увидите голубой символ «+» слева от каждого вопроса. Это обычное обозначение, которое говорит «Эй, это еще не все». Чтобы обозначить то, что посетитель может

скрыть ответ, замените символ «+» символом «–». Это легко сделать, добавив класс для элемента h2.

9. **После строки кода, добавленной в предыдущем шаге, введите следующее:**

```
$(this).addClass('close');
```

Помните, что объект $(this) относится к элементу, реагирующему на событие (см. раздел «Использование событий на практике» ранее в этой главе). В данном случае это элемент h2. Таким образом, эта новая строка кода добавляет класс close, когда ответ отображен. Значок «—» определяется в рамках таблицы стиля как фоновое изображение (и вновь CSS облегчает программирование на языке JavaScript).

В следующем шаге вы завершите вторую половину эффекта переключения — обеспечите сокрытие ответа при втором щелчке по вопросу.

10. **В части `else` управляющей инструкции добавьте еще две строки кода (выделены полужирным шрифтом). Итоговый код должен выглядеть следующим образом:**

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
$('.answer').hide();
$('.main h2').click(function() {
var $answer = $(this).next('.answer');
if ($answer.is(':hidden')) {
$answer.slideDown();
$(this).addClass('close');
} else {
$answer.fadeOut();
$(this).removeClass('close');
}
}); // окончание click
}); // окончание ready()
</script>
```

Эта часть программы отвечает за сокрытие ответа. Вы могли бы использовать функцию `slideUp()`, скрывающую элемент, как бы задвигая его, однако для интереса и разнообразия в данном случае вы заставите ответ исчезнуть с помощью функции `fadeOut()`, о которой вы подробнее узнаете в следующей главе.

Наконец, вы удалите класс `close` из элемента `h2`: это приведет к повторному появлению знака «+» слева от вопроса.

Сохраните страницу и просмотрите ее. Теперь при щелчке по вопросу не просто появляется ответ, но и изменяется значок вопроса (см. рис. 5.7).

ПРИМЕЧАНИЕ

По окончании вы можете попробовать заменить функцию `slideDown()` функцией `fadeIn()`, а функцию `fadeOut()` — функцией `slideUp()`. Какой вариант вам больше нравится?

Глава 6
АНИМАЦИЯ И ЭФФЕКТЫ

В двух предыдущих главах вы изучили основы использования библиотеки jQuery: как загружать библиотеку, выбирать элементы страницы и реагировать на такие события, как щелчок кнопкой мыши или наведение курсора на ссылку. Большинство программ jQuery подразумевает три действия: выбор элемента на странице, прикрепление к нему события и реагирование на это событие путем выполнения неких действий. В этой главе вы приступите к изучению части «некие действия» с помощью встроенных в библиотеку jQuery функций эффектов и анимации. Вы также освежите свои знания о некоторых важных свойствах CSS, имеющих отношение к созданию визуальных эффектов. Кроме того, вы узнаете о том, как комбинировать анимацию CSS3 с инструментами jQuery, предназначенными для легкого создания анимационных эффектов.

Эффекты jQuery

Появление элементов на веб-странице и их исчезновение — обычная задача для языка JavaScript. Выпадающие навигационные меню, всплывающие подсказки и автоматизированные слайд-шоу основаны на способности отображать и скрывать элементы. Библиотека jQuery предлагает несколько функций, с помощью которых осуществляется отображение и сокрытие элементов.

Чтобы использовать каждый из доступных эффектов, вы должны применить их к выборке jQuery, как и любую другую функцию библиотеки. Например, чтобы скрыть все элементы класса `submenu`, вы можете написать следующее:

```
$('.submenu').hide();
```

Каждая из функций эффекта может иметь настройки скорости и функцию *обратного вызова*. Скорость — это период времени, необходимый для выполнения эффекта, а обратный вызов — это функция, которая выполняется по завершении эффекта (подробнее о функции

обратного вызова написано в разделе данной главы «Выполнение действия после завершения эффекта»).

Чтобы задать скорость эффекта, соответствующей функции передается одно из трех строковых значений (`'fast'`, `'normal'` или `'slow'`) либо число, представляющее количество миллисекунд, необходимых для завершения эффекта (1000 = 1 секунда, 500 = полсекунды и т. д.). Например, код для медленного исчезновения элемента будет выглядеть так:

```
$('element').fadeOut('slow');
```

Если же вы хотите, чтобы элемент исчезал *действительно* медленно — в течение 10 секунд:

```
$('element').fadeOut(10000);
```

При использовании эффекта исчезновения элемент не удаляется из объектной модели документа (см. раздел «Объектная модель документа (DOM)» главы 4). Его код по-прежнему находится в памяти браузера, однако, настройка `display` (такая же, как и одноименная настройка CSS) устанавливается на значение `none`. Таким образом, данный элемент больше не занимает визуальное пространство, поэтому остальное содержимое страницы может переместиться на освободившееся место. Вы можете увидеть все эффекты jQuery в действии в файле *эффекты.html* в каталоге *эксперименты* (рис. 6.1).

ПРИМЕЧАНИЕ

Ключевые слова, используемые для задавания скорости эффекта, — `'fast'`, `'normal'` или `'slow'` — соответствуют 200, 400 и 600 миллисекундам. Поэтому код:

```
$('element').fadeOut('slow');
```

равнозначен коду:

```
$('element').fadeOut(600);
```

Основы отображения и сокрытия

Библиотека jQuery предлагает три функции для отображения и сокрытия элементов.

- **`show()`** показывает ранее скрытый элемент. Она не работает, когда элемент уже виден на странице. Если вы не устанавливаете значение скорости, элемент появляется моментально. Однако если скорость

задана, например, `show(1000)`, то элемент анимируется от верхнего левого к нижнему левому углу.

- **`hide()`** скрывает видимый элемент. Она не работает, если элемент уже скрыт, и, точно так же, как и в случае с функцией `show()`, если вы не передаете значения скорости, то элемент моментально исчезает. Однако при установленном значении скорости он исчезает, как бы сужаясь.
- **`toggle()`** переключает текущее состояние элемента. Если элемент в данный момент отображен, то функция `toggle()` скрывает его. Если элемент скрыт, она заставляет его появиться. Эта функция идеально подходит, когда вам нужен один элемент управления (например, кнопка) для попеременного показа и сокрытия элемента.

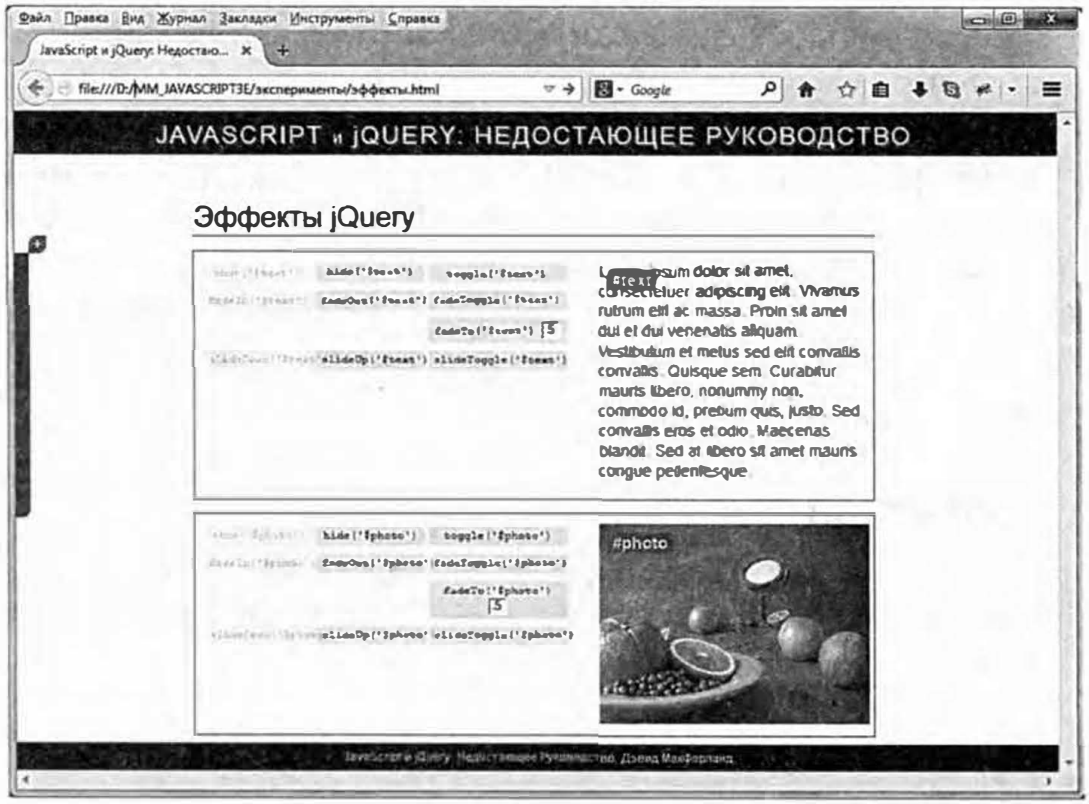

Рис. 6.1. Вы можете протестировать визуальные эффекты jQuery в файле *эффекты.html* в каталоге *эксперименты*. Щелкните по функции (например, `fadeOut('#photo')`), чтобы посмотреть, как выглядят текст и изображения, когда они исчезают, «скользят» или появляются. Некоторые элементы окрасятся в серый цвет. Это значит, что эффект неприменим к элементу. Например, нет смысла заставлять фотографию появляться, если она уже видна на странице

В руководстве в предыдущей главе вы видели функцию `hide()` в действии. Сценарий использует функцию `hide()`, чтобы скрыть все ответы на странице пока загружается ее HTML-код.

Постепенное появление и исчезновение элементов

Для более впечатляющего эффекта вы можете заставить элемент постепенно появляться и исчезать. В данном случае вы просто меняете степень его непрозрачности за определенный период времени. Для достижения этой цели библиотека jQuery предлагает три функции:

- **`fadeIn()`** показывает скрытый элемент. Сначала на странице появляется пространство для элемента (это может означать, что остальные элементы страницы сдвинутся с места), затем элемент постепенно становится видимым. Данная функция не работает, если он уже виден на странице. Если вы не зададите скорость, эффект появления будет использовать значение `'normal'` (400 миллисекунд);
- **`fadeOut()`** заставляет элемент исчезать, делая его прозрачным. Она не работает, если элемент уже скрыт. Как и в случае с функцией `fadeIn()`, если вы не присваиваете значения скорости, то элемент исчезает в течение 400 миллисекунд. В примере «Создание страницы ЧаВо на практике» в предыдущей главе эта функция обеспечивала исчезновение ответов;
- **`fadeToggle()`** сочетает эффекты функций `fadeIn()` и `fadeOut()`. Если в настоящий момент элемент скрыт, то при использовании функции `fadeToggle()` он появляется, если виден — исчезает. Вы можете использовать данную функцию для того, чтобы заставить рамку с инструкциями появляться и исчезать со страницы. Например, у вас есть кнопка со словом «Инструкции». Когда посетитель щелкает по ней, появляются инструкции; при повторном щелчке — они исчезают. Для того чтобы рамка появлялась или исчезала в течение полсекунды (500 миллисекунд), вы можете написать следующий код:

```
$('#button').click(function() {
$('#instructions').fadeToggle(500);
}); // окончание события click
```

- **`fadeTo()`** работает немного не так, как остальные функции. Она изменяет степень непрозрачности изображения до определенного значения. Например, вы можете сделать картинку полупрозрачной. В отличие от остальных эффектов, задавание скорости является обязательным. Дополнительно присваивается числовая величина в диапазоне от 0 до 1, характеризующая степень непрозрачности элемента. Например, чтобы сделать все абзацы на 75% непрозрачными, напишите так:

```
$('p').fadeTo('normal',.75);
```

Данная функция изменяет степень непрозрачности элемента вне зависимости от того, виден он или нет. Например, вы задаете значение непрозрачности, равное 50%, для элемента, который в настоящий момент скрыт. Если затем вы отобразите этот элемент, он окажется полупрозрачным. Если вы скрываете полупрозрачный элемент и заставляете его появиться, то при появлении степень его непрозрачности возвращается.

Если вы установите значение непрозрачности на 0, то элемент не будет виден, однако продолжит занимать место на странице. Иными словами, в отличие от других эффектов исчезновения, установка значения непрозрачности на 0 оставит пустое место там, где находится элемент.

Кроме того, если вы зададите для элемента значение непрозрачности, а затем скроете его, то он исчезнет со страницы. Если затем вы отобразите этот элемент, он «вспомнит» свой параметр непрозрачности, поэтому браузер заставит элемент снова появиться на странице, но со степенью непрозрачности 50%.

Скользящие элементы

Чтобы добавить больше действия, можно также заставить элемент ускользать из виду и появляться в поле зрения. Следующие функции похожи на только что описанные тем, что они могут заставлять элементы страницы постепенно появляться и исчезать, а также могут принимать значение скорости.

- **`slideDown()`** заставляет скрытый элемент появиться в поле зрения. Сначала появляется верхняя часть, и, по мере появления остальной части, то, что находится под элементом, сдвигается вниз. Функция не влияет на элемент, уже видимый на странице. Если вы не присваиваете значения скорости, то элемент появляется на странице, используя настройку `'normal'` (400 миллисекунд). В руководстве «Создание страницы ЧаВо на практике» в предыдущей главе эта функция обеспечивала появление ответов.
- **`slideUp()`** удаляет элемент из виду, скрывая его нижнюю часть и двигая все, что находится ниже, вверх по мере того, как элемент исчезает. Эффекта не будет, если элемент уже скрыт. Как и в случае с функцией `slideDown()`, если вы не задаете значение скорости, элемент ускользает в течение 400 миллисекунд.
- **`slideToggle()`** применяет функцию `slideDown()` к элементу, который в данный момент скрыт, а функцию `slideUp()` — к види-

мому. Она позволяет создать единый элемент управления (например, кнопку) как для отображения, так и для сокрытия элемента.

УСКОРЯЕМ РАБОТУ

Абсолютное позиционирование с помощью CSS

Обычно, когда вы скрываете элемент на веб-странице, другие элементы сдвигаются, чтобы занять его место. Например, если вы скроете изображение, оно исчезнет, а содержимое, которое располагается под ним, сдвинется вверх. Подобным образом появление элемента заставляет остальное содержимое подвинуться и освободить место для вновь отображенного элемента. Если вы не хотите, чтобы ваш контент «прыгал» из стороны в сторону, то можете использовать CSS и *абсолютное позиционирование* для размещения элемента так, чтобы он не мешал остальному содержимому. Другими словами, используя свойство CSS `position`, вы способны разместить элемент `div`, изображение или абзац поверх страницы так, как будто они находятся на отдельном слое.

Чтобы элемент появлялся вверху страницы, установите значение свойства `position` на `absolute`. Далее вы можете уточнить местоположение этого элемента на странице, используя свойства `left`, `right`, `top` и/или `bottom`. Рассмотрим элемент `div`, содержащий форму авторизации. Она обычно невидима, но если посетитель щелкнет по ссылке, форма авторизации надвинется поверх остального содержимого страницы. Вы можете позиционировать элемент `div` следующим образом:

```
#login {
position: absolute;
left: 536px;
top: 0;
width: 400px;
}
```

Данный код размещает элемент `div` в верхней части окна браузера на расстоянии 536 пикселов от левого края. Допускается также позиционировать элемент относительно правого края окна браузера, используя свойство `right`, или относительно нижнего края, используя свойство `bottom`.

Конечно, вам может понадобиться поместить элемент относительно чего-либо кроме окна браузера. Например, всплывающие подсказки обычно помещаются относительно другого элемента: рядом со

словом на странице может быть знак «?», при щелчке по которому появляется небольшая рамка с определением данного слова. В данном случае подсказку следует позиционировать не относительно краев окна браузера, а рядом со словом. Чтобы достичь этого, необходимо задать значение `relative` свойства CSS `position` для элемента, окружающего абсолютно позиционированный элемент. Например, посмотрите на следующий HTML-код:

```
<span class="word">Слонопотам
<span class="definition">Вымышленное животное, ↵
упоминающееся в сборниках рассказов о приключениях ↵
Винни-Пуха</span>
</span>
```

Чтобы элемент `span` с определением появлялся под словом, вы должны сначала разместить элемент `span` со словом, а затем абсолютно позиционировать определение:

```
.word { position: relative; }
.definition {
position: absolute;
bottom: -30px;
left: 0;
width: 200px;
}
```

За более подробной информацией об абсолютном позиционировании обратитесь на сайт: **www.elated.com/articles/css-positioning/** или прочтите «Большую книгу CSS3» Дэвида МакФарланда.

Всплывающее окно авторизации на практике

В данном руководстве вы попрактикуетесь в использовании эффектов библиотеки jQuery, создав распространенный элемент пользовательского интерфейса — панель, которая выдвигается в ответ на щелчок кнопкой мыши (рис. 6.2).

Задача довольно проста:

1. **Выбрать абзац с сообщением Login.**

   Помните, что основная часть программ jQuery начинается с выбора элемента на странице. В данном случае абзац Login будет реагировать на щелчки кнопкой мыши.

2. **Прикрепить к выбранному абзацу событие `click`.**

   Без событий язык JavaScript не был бы интерактивным: посетителю необходимо взаимодействовать с выборкой (абзац Login), чтобы что-нибудь произошло.

3. **Создать переключатель для последовательного отображения и** сокрытия **формы.**

   Два предыдущих шага являются всего лишь напоминанием (они необходимы для многих программ jQuery). В этом последнем шаге вы будете использовать эффекты, о которых узнали. Вы можете заставить форму мгновенно появиться (функция `show()`), выдвинуться (функция `slideDown()`) или постепенно проявиться (функция `fadeIn()`).

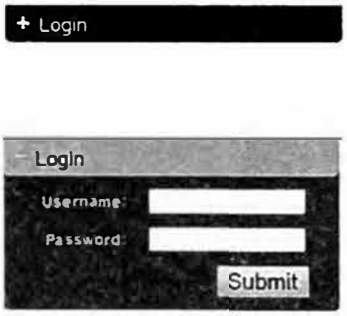

Рис. 6.2. Форма авторизации обычно скрыта (верхний рисунок), однако простой щелчок кнопкой мыши отображает форму, готовую для ввода данных

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Верстка кода

1. **В текстовом редакторе откройте файл *06_01.html* в каталоге *глава06*.**

   Этот документ уже содержит ссылку на файл jQuery и функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5). Сначала выберите абзац с текстом «Login».

2. **Щелкните в пустой строке после функции `$(document).ready()` и введите `$('#open')`.**

   Текст Login находится в элементе абзаца, окруженного ссылкой: `<a href="form.html"><p id="open">Login</p></a>`. Эта ссыл-

ка будет направлять пользователей на другую страницу, содержащую форму авторизации. Эта ссылка нужна потому, что если в браузере посетителя отключен JavaScript, то он не увидит скрытую форму. Добавляя ссылку, вы гарантируете, что даже люди с отключенным JavaScript смогут получить доступ к форме авторизации.

Только что введенный вами фрагмент `$('#open')` выбирает этот абзац. Теперь пришло время добавить обработчик событий.

ПРИМЕЧАНИЕ

Элемент `p`, упомянутый в шаге 2, находится в элементе `a`. Если вы уже в течение некоторого времени занимаетесь созданием вебсайтов, то можете посчитать такой HTML-код ошибочным. В HTML4 и в более ранних версиях так и было. При использовании типа документа HTML5 (см. раздел «HTML: скелет-основа» во введении) допустимо включать в ссылки такие блочные элементы, как `p`, `h1` и даже `div`. Это позволяет легко создавать крупные «кликабельные» элементы-ссылки.

3. **Добавьте выделенный полужирным шрифтом код, чтобы сценарий выглядел так:**

```
$(document).ready(function() {
$('#open').click(function(evt) {
}); // окончание события click
}); // окончание функции ready
```

Данный фрагмент кода добавляет обработчик события `click`, так что каждый щелчок кнопкой мыши по абзацу приведет к какому-то результату. В данном случае при первом щелчке должна появиться форма, а при повторном — она должна исчезнуть, и т. д. Другими словами, форма обязана поочередно отображаться и скрываться. Библиотека jQuery предлагает три функции, служащие для этой цели: `toggle()`,`fadeToggle()` и `slideToggle()`. Отличие заключается лишь в том, как выглядит эффект.

Вы также добавляете аргумент в анонимную функцию внутри метода `click`. Как говорилось в разделе «Объект события» главы 5, каждому событию автоматически передается объект события, который содержит различные свойства и методы. Вам нужно, чтобы этот объект дал jQuery задание предотвратить переход по ссылке на страницу с формой.

4. **Установите текстовый курсор в начало пустой строки внутри функции `click` и введите:**

```
evt.preventDefault();
```

Метод `evt.preventDefault();` не позволяет браузеру перейти по ссылке, окружающей абзац. Помните о том, что эта ссылка призвана обеспечить доступ к форме авторизации тем посетителям, у которых отключен JavaScript. Однако браузерам с включенным JavaScript вы должны дать задание оставаться на странице и выполнить дополнительный код.

5. **После добавленной в предыдущем шаге строки кода введите:**

```
$('#login form').slideToggle(300);
```

Этот код выбирает форму авторизации, а затем выдвигает ее, если в данный момент она невидима. Если она видима — задвигает ее обратно. Наконец, вы замените класс абзаца, чтобы форма меняла вид, используя стиль класса CSS.

6. **Добавьте выделенный полужирным шрифтом код, чтобы итоговый сценарий выглядел так:**

```
$(document).ready(function() {
$('#open').click(function(evt) {
evt.preventDefault();
$('#login form').slideToggle(300);
$(this).toggleClass('close');
}); // окончание события click
}); // окончание функции ready
```

Как вы помните из раздела «Ключевые слова `this` и `$(this)`» главы 4 в пределах обработчика событий вы можете использовать ключевое слово `$(this)` для обращения к элементу, реагирующему на событие. В данном случае `$(this)` относится к абзацу, по которому производится щелчок — часть `$('#open')` в строке 2. Функция `toggleClass()` просто добавляет или удаляет класс элемента. Как и другие функции-переключатели, функция `toggleClass()` добавляет конкретный класс, если он отсутствует, или удаляет класс, если он присутствует. В данном примере в таблице стиля на странице присутствует стиль класса — `.close`. Этот стиль просто добавляет другое фоновое изображение, чтобы указать посетителю на возможность закрытия формы авторизации. (Посмотрите в раздел заголовка документа и вы увидите стиль и то, что он делает.)

7. Сохраните страницу и просмотрите ее в браузере.

Щелкните по абзацу Login несколько раз, чтобы увидеть, как работает сценарий. Вы найдете готовую версию руководства в файле *готовый_06_01.html* в папке *глава 06*. Вы также можете попробовать другие эффекты переключения, заменив функцию `slideToggle()` на `toggle()` или `fadeToggle()`.

Но что если вам нужны два различных эффекта? Один для того, чтобы заставить форму появиться (например, выдвинуться), а другой, например, для того, чтобы форма медленно исчезла. Код в шаге 5 не подойдет, так как функция `click()` не позволит вам выбирать между разными действиями. Вам потребуется использовать тот же трюк, который вы использовали в разделе «Создание страницы ЧаВо на практике» главы 5: когда посетитель щелкает по ссылке Login вам нужно проверить, не скрыта ли форма. И если скрыта, — отобразить ее, а если нет — скрыть.

Чтобы заставить форму выдвигаться и медленно исчезать в ответ на последовательные щелчки кнопкой мыши, вы можете использовать следующий код:

```
$(document).ready(function() {
   $('#open').click(function(evt) {
evt.preventDefault();
if ($('#login form').is(':hidden')) {
$('#login form').fadeIn(300);
$(this).addClass('close');
} else {
$('#login form').slideUp(600);
$(this).removeClass('close');
}
}); // окончание события click
}); // окончание функции ready
```

ПРИМЕЧАНИЕ

Вы найдете готовую версию приведенного выше кода в файле *готовый_06_02.html* в папке *глава 06*.

Анимация

Вы не ограничены только эффектами, встроенными в библиотеку jQuery. Используя функцию `animate()`, вы можете анимировать любое свойство CSS, принимающее числовое значение (пикселы, em или проценты). Например, размер текста, позицию элемента на странице, степень непрозрачности объекта или толщину границы.

ПРИМЕЧАНИЕ

Сама по себе библиотека jQuery не может анимировать изменение цвета, например, текста, фона или границы. Однако это может сделать плагин jQuery UI, который предусматривает множество дополнительных анимационных эффектов. Вы узнаете об инструментах jQuery UI для создания анимации в главе 12.

Для использования этой функции вам следует передать объектную константу (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), содержащую список свойств CSS, которые вы хотите изменить, и значения, изменение на которые необходимо анимировать.

Например, вы желаете анимировать элемент, передвинув его на 650 пикселов от левого края страницы, изменив степень непрозрачности на 50% и увеличив размер шрифта до 24 пикселов. Следующий код создает объект именно с такими свойствами и значениями:

```
{
left: '650px',
opacity: .5,
fontSize: '24px'
}
```

Заметьте, что вы должны указывать значение в кавычках, только если оно включает единицу измерения, например px, em или %. В данном примере вам следует указать значение `'650px'` в кавычках, так как оно содержит часть `'px'`. Однако значение непрозрачности не нужно брать в кавычки, поскольку `.5` — это просто числовое значение, которое не содержит каких-либо символов. Заключение в кавычки свойства (`left`, `opacity` и `fontSize`) не является обязательным.

ПРИМЕЧАНИЕ

Язык JavaScript не принимает дефисы в именах свойств CSS. Например, `font-size` — это свойство CSS, но JavaScript не понимает его, поскольку дефис имеет особое значение (в языке JavaScript дефис соответствует операции «минус»). При использовании свойств CSS в JavaScript удалите дефис и сделайте заглавной первую букву слова, следующего за дефисом. Например, `font-size` станет `fontSize`, а `border-width` — `borderWidth`. Однако если вы предпочитаете имена свойств, принятые в CSS (во избежание путаницы), заключайте их в кавычки:

```
{
'font-size': '24px',
'border-left-width': '2%'
}
```

Допустим, вы хотите анимировать элемент с идентификатором `message`, используя данные настройки. Вы можете использовать функцию `animate()` так:

```
$('#message').animate(
{
left: '650px';
opacity: .5,
fontSize: '24px';
},
1500
);
```

Функция `animate()` может принимать несколько аргументов. Первый — это литерал объекта, содержащий свойства CSS, которые вы желаете анимировать. Второй — это длительность анимации (в миллисекундах). В примере, приведенном выше, анимация длится 1500 миллисекунд (1,5 секунды).

ПРИМЕЧАНИЕ

Чтобы анимировать положение элемента с использованием свойств CSS `left`, `right`, `top` или `bottom`, вы должны установить значение свойства CSS `position` на `absolute` или `relative`. Только этим

двум свойствам позиционирования можно присвоить значения (см. врезку «Абсолютное позиционирование с помощью CSS»).

Вы также можете задать свойство относительно его текущего значения, используя операции += или –= как часть настроек анимации. Например, вы хотите анимировать элемент, сдвигая его на 50 пикселов вправо при каждом щелчке по нему кнопкой мыши. Вот как это делается:

```
$('#moveIt').click(function() {
$(this).animate(
{
left:'+=50px'
},
1000); окончание animate
}); окончание click
```

Управление скоростью анимации

Функции эффектов библиотеки jQuery (`slideUp()`, `fadeIn()` и т. д.), а также функция `animation()`, принимают еще один аргумент, который контролирует скорость во время анимации: `easing`. Например, при перемещении элемента по странице движение элемента может начинаться медленно, затем ускоряться, и, наконец, замедляться по мере завершения анимации. Использование этого аргумента может сделать анимацию визуально более интересной и динамичной.

Библиотека jQuery включает только два метода `easing`: `swing` и `linear`. Метод `linear` производит равномерную анимацию, так что каждый шаг анимации подобен всем остальным (например, если вы анимируете перемещение элемента по экрану, каждый шаг будет соответствовать одному и тому же расстоянию). Метод `swing` более динамичен: анимация начинается в более быстром темпе, а затем замедляется. Метод `swing` установлен по умолчанию, поэтому если вы не укажете аргумент `easing`, библиотека jQuery будет использовать метод `swing`.

Аргумент `easing` указывается вторым по порядку для любого эффекта jQuery, поэтому чтобы заставить элемент сместиться вверх, используя линейный метод, напишите следующий код:

```
$('#element').slideUp(1000,'linear');
```

При использовании функции `animate()` метод `easing` указывается в качестве третьего аргумента после объектной константы, содержащей свойства CSS, которые вы хотите анимировать, и значения общей скорости анимации. Например, чтобы использовать линейный метод `easing` в коде, приведенном в начале данного раздела, напишите следующее:

```
$('#message').animate(
{
left: '650px',
opacity: .5,
fontSize: '24px'
},
1500,
'linear'
);
```

Вы не ограничены только этими двумя методами, предоставляемыми библиотекой jQuery. Благодаря неустанной работе других программистов, вы можете добавить множество других методов, некоторые из которых весьма драматичны и интересны. На самом деле, библиотека jQuery UI включает множество дополнительных методов `easing`. Подробнее о плагине jQuery UI вы узнаете в части III книги, однако можете уже сейчас начать использовать его, чтобы сделать анимацию более интересной.

Чтобы использовать плагин jQuery UI (который представляет собой внешний файл JavaScript), вы должны поместить ссылку на этот файл на своей странице после ссылки на библиотеку jQuery. После присоединения плагина jQuery UI вы можете использовать любой доступный метод (за полным списком методов обратитесь на сайт: **api.jqueryui.com/easings/**). Например, вы хотите, чтобы элемент `div` увеличивался в размере при щелчке по нему кнопкой мыши, и вам также нужно, чтобы анимационный эффект был интересным, поэтому вы используете метод `easeInBounce`. Если элементу `div` присвоен идентификатор `animate`, то ваш код может выглядеть таким образом:

```
1 <script src="_js/jquery.min.js"></script>
2 <script src="_js/jquery-ui.min.js"></script>
3 <script>
4 $(document).ready(function() {
```

```
5     $('#animate').click(function() {
6           $(this).animate(
7           {
8                 width: '400px',
9                 height: '400px'
10          },
11          1000,
12          'easeInBounce'); // окончание функции animate
13    }); // окончание события click
14 }); // окончание функции ready
15 </script>
```

Строки 1 и 2 загружают библиотеку jQuery и плагин jQuery UI. Строка 4 — это вездесущая функция `ready()` (см. раздел «Больше концепций для событий jQuery» главы 5), а строка 5 добавляет элементу `div` обработчик события `click`. Самое интересное происходит в строках с 6 по 12. Как вы помните из раздела «Ключевые слова `this` и `$(this)`» главы 4, в пределах события ключевое слово `$(this)` относится к элементу, реагирующему на событие — в данном случае, к элементу `div`. Другими словами, при щелчке по элементу `div` вы также анимируете этот элемент, изменяя его ширину и высоту (строки 8 и 9). Строка 11 заставляет анимацию длиться в течение 1 секунды (1000 миллисекунд), а строка 12 устанавливает метод `easeInBounce` (вы можете подставить любой другой метод, например `easeInOutSine`, `easeInCubic` и т. д.).

ПРИМЕЧАНИЕ

Посмотреть код, приведенный выше, в действии вы можете, открыв файлы *easing_пример1.html* и *easing_пример2.html* в папке *глава06*. Файл *easing_пример2.html* иллюстрирует использование события `toggle()` для применения двух различных методов `easing` к элементу `div`.

Выполнение действия после завершения эффекта

Иногда вам нужно сделать что-либо по завершении эффекта. Допустим, когда появляется конкретная фотография, вы хотите, чтобы на экране отобразилась надпись. Другими словами, надпись должна поя-

виться на странице после загрузки фотографии. Как правило, эффекты не выполняются один за другим, они происходят непосредственно после их вызова. Итак, если в вашем сценарии есть строка кода, при выполнении которой появляется фотография, и вторая строка, вызывающая появление надписи, то надпись появится во время загрузки фотографии.

Для решения этой дилеммы вы можете передать *функцию обратного вызова* любому эффекту. Она выполняется только после завершения эффекта. Функция обратного вызова передается большинству эффектов в качестве второго аргумента (а функции `fadeTo()` — в качестве третьего аргумента).

Например, на вашей странице есть изображение с идентификатором `photo`, а под ним — абзац с идентификатором `caption`. Чтобы появилась фотография и только потом — надпись, можно использовать функцию обратного вызова таким образом:

```
$('#photo').fadeIn(1000,function() {
$('#caption').fadeIn(1000);
});
```

Конечно же, если вы хотите запустить функцию во время загрузки страницы, нужно перед этим скрыть фотографию и надпись, а затем использовать эффект `fadeIn` следующим образом:

```
$('#photo, #caption').hide();
$('#photo').fadeIn(1000,function() {
$('#caption').fadeIn(1000);
});
```

Если вы используете функцию `animate()`, то функция обратного вызова появляется после других аргументов — объектной константы, содержащей свойства CSS, изменение которых анимируется, значения продолжительности анимации и настроек метода `easing`. Настройки метода `easing` необязательны, так что вы можете просто передать функцию `animate()`, список свойств, продолжительность и функцию обратного вызова. Например, вам нужно не только заставить появиться фото, но также увеличить его ширину и высоту с нуля до полного размера (эффект увеличения масштаба). Для этого вы можете использовать функцию `animate()`, а затем отобразить надпись:

```
1 $('#photo').width(0).height(0).css('opacity',0);
2 $('#caption').hide();
3 $('#photo').animate(
4     {
5           width: '200px',
6           height: '100px',
7           opacity: 1
8     },
9     1000,
10    function() {
11          $('#caption').fadeIn(1000);
12    }
13 ); // окончание функции animate
```

Строка 1 приведенного выше кода устанавливает значения ширины, высоты и степени непрозрачности фотографии на 0. (Это скрывает фотографию и готовит ее к анимации). Строка 2 скрывает надпись. Строки с 3 по 13 описывают действие функции анимации, а в строках с 10 по 12 появляется функция обратного вызова. Возможно, данный код выглядит устрашающе, но, к сожалению, использование функции обратного вызова — это единственный способ выполнения действия (включая эффект, примененный к другому элементу страницы) после завершения эффекта.

ПРИМЕЧАНИЕ

Файл *callback.html* в папке *глава06* демонстрирует представленный код в действии.

Функции обратного вызова могут запутать, если вы хотите анимировать несколько элементов сразу: например, сделать так, чтобы сначала изображение передвинулось в центр экрана и появилась надпись, а затем они исчезли. Чтобы этого добиться, вам нужно передать функции обратного вызова другой функции обратного вызова:

```
$('#photo').animate(
{
left: '+=400px',
},
1000,
```

```
function() { // первая функция callback
$('#caption').fadeIn(1000,
function() { // вторая функция callback
$('#photo, #caption').fadeOut(1000);
} // окончание второй функции callback
); // окончание функции fadeIn
} // окончание первой функции callback
); // окончание функции animate
```

ПРИМЕЧАНИЕ

Файл *multiple-callbacks.html* в папке *глава06* демонстрирует представленный код в действии.

Тем не менее вам необязательно использовать функцию обратного вызова, если вы хотите добавить дополнительные анимационные эффекты к тому же элементу страницы.

Например, вам нужно переместить фотографию по экрану, а затем скрыть ее. В данном случае достаточно просто использовать функцию `animate()`, чтобы переместить фотографию, а затем заставить ее медленно исчезнуть. Вы можете достичь подобного эффекта следующим образом:

```
$('#photo').animate(
{
left: '+=400px',
},
1000
); // окончание функции animate
$('#photo').fadeOut(3000);
```

В данном случае браузер выполняет код сразу же, jQuery помещает каждый эффект в очередь, так что сначала выполняется анимация, а затем функция `fadeOut()`. Используя связывание (см. раздел «Связывание функций» главы 4), вы можете переписать представленный выше код следующим образом:

```
$('#photo').animate(
{
```

```
left: '+=400px',
},
1000).fadeOut(3000);
```

Если вы хотите, чтобы фотография появилась, исчезла и снова появилась, вы можете использовать связывание так:

```
$('#photo').fadeIn(1000).fadeOut(2000).fadeIn(250);
```

ПРИМЕЧАНИЕ

Более подробную информацию о том, как работает очередь эффектов, вы можете узнать на сайте: **api.jquery.com/jQuery.queue/**.

Еще одна jQuery-функция, которая может пригодиться при создании очереди эффектов — это `delay()`. Данная функция просто ожидает определенное количество миллисекунд, прежде чем запускать следующий эффект в очереди. Например, вы хотите, чтобы изображение исчезло через 10 секунд после своего появления. Вы можете использовать функцию `delay()` следующим образом:

```
$('#photo').fadeIn(1000).delay(10000).fadeOut(250);
```

Анимированная панель навигации на практике

В данном руководстве вы будете использовать функцию `animate()` для перемещения элемента `div` от левого края страницы. Элемент `div` абсолютно позиционирован (см. врезку «Абсолютное позиционирование с помощью CSS»), поэтому большая часть рамки находится у левого края страницы за пределами окна браузера (рис. 6.3 слева). Когда посетитель наводит указатель мыши на видимый край элемента `div`, он полностью выдвигается (рис. 6.3 справа). Чтобы сделать этот эффект более интересным, вы также будете использовать два плагина для анимации цвета фона элемента `div` и пару разных методов `easing`.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

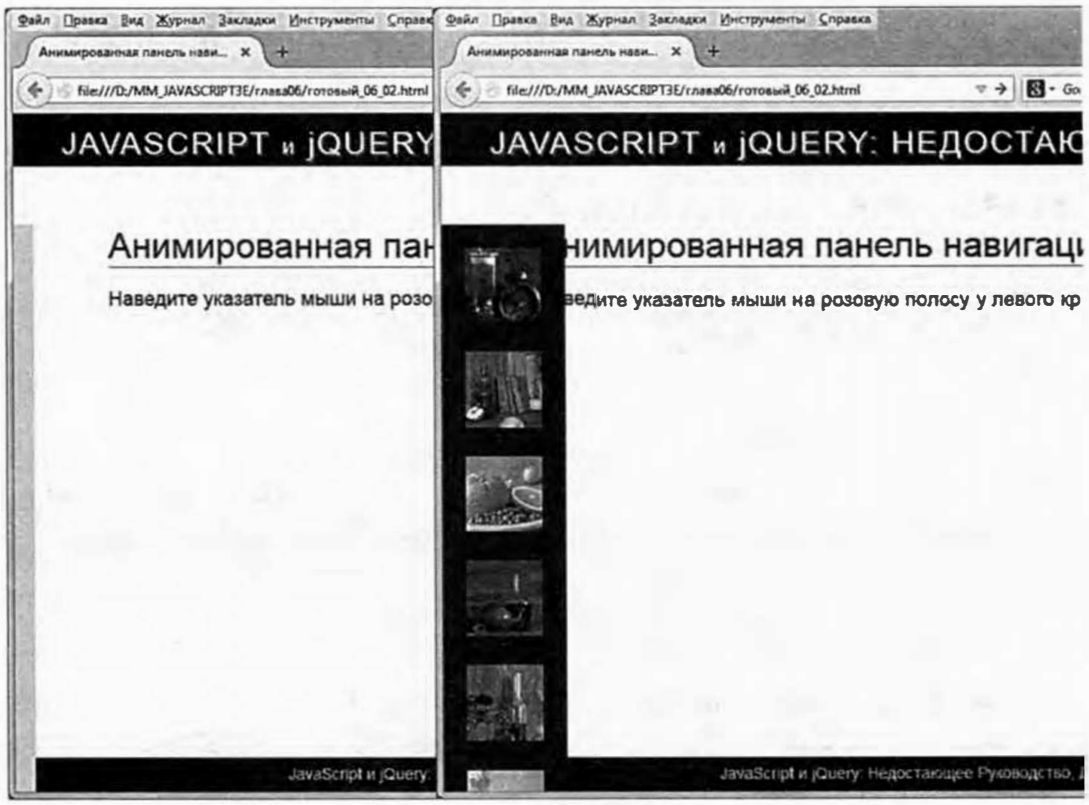

Рис. 6.3. Вы можете получить массу удовольствия, скрывая элементы страницы за одним из краев окна браузера (слева). Используя функцию `animate()`, можно полностью отобразить элемент страницы (справа)

Задача довольно простая:

1. **Выберите элемент `div`.**

   Помните, что большинство программ jQuery начинается с выбора элемента на странице, в данном случае — это элемент `div`, на который посетитель наводит указатель мыши.

2. **Прикрепите событие `hover`.**

   Событие `hover` (описанное в разделе «События наведения и смещения указателя мыши» главы 5) — это специальная jQuery-функция (а не событие языка JavaScript), которая позволяет вам производить одну серию действий, когда посетитель наводит указатель мыши на элемент, а затем следующую серию действий, когда посетитель смещает курсор с элемента (событие `hover` — это просто сочетания событий `mouseEnter` и `mouseLeave`).

3. **Добавьте функцию `animate()` для события `mouseEnter`.**

   Когда посетитель наводит указатель мыши на элемент `div`, вы будете анимировать свойство `left` элемента `div`, выдвигая его из-за

левого края окна браузера. Кроме того, вы станете анимировать цвет фона элемента `div`.

4. **Добавьте еще одну функцию `animate()` для события `mouseLeave`.**

   Когда посетитель смещает указатель мыши с элемента `div`, вы будете анимировать возврат элемента `div` к исходному состоянию и восстановление оригинального фонового цвета.

Верстка кода

1. **В текстовом редакторе откройте файл *06_02.html* в папке *глава06*.**

   Этот документ уже содержит ссылку на файл jQuery и функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5). Однако поскольку вы будете анимировать цвет фона элемента `div` и использовать два интересных метода `easing`, вам нужно прикрепить плагин jQuery UI.

2. **Щелкните в пустой строке после первого элемента `script` и добавьте код, выделенный полужирным шрифтом:**

```
<script src="../_js/jquery-1.7.2.min.js"></script>
<script src="../_js/jquery-ui.min.js"></script>
```

   jQuery UI — это плагин jQuery. В мире jQuery плагины представляют собой внешние файлы JavaScript, которые расширяют функциональность библиотеки jQuery и часто позволяют вам добавлять сложные эффекты на сайт без необходимости тратить много времени на программирование. Далее вам предстоит выбрать элемент `div` и добавить к нему функцию `hover()`.

3. **Щелкните в пустой строке внутри функции `$(document).ready()` и введите `$('#dashboard').hover(); // окончание функции hover`, чтобы ваш код выглядел так:**

```
$(document).ready(function() {
$('#dashboard').hover(); // окончание функции hover
}); // окончание функции ready
```

   Код `$('#dashboard')` выбирает элемент `div` (с идентификатором `dashboard`). Функция `hover()` принимает два аргумента — две функции, которые описывают, что необходимо сделать, когда посе-

титель наводит указатель мыши на элемент `div` и смещает его с этого элемента. Вместо того чтобы вводить весь код сразу, вы станете делать это постепенно: сначала добавите функцию `hover()`, а затем добавите «оболочку» для двух анонимных функций. Этот подход удобен, так как множество круглых и фигурных скобок, запятых и точек с запятой могут запутать, если вы не будете внимательны.

4. **Установите текстовый курсор между круглыми скобками функции `hover()` и добавьте две пустые анонимные функции:**

```
$(document).ready(function() {
$('#dashboard').hover(
function() {
},
function() {
        }
); // окончание функции hover
}); // окончание функции ready
```

JQuery-функция `hover()` выглядит довольно устрашающе после добавления программного кода (см. шаг 12). Тем не менее, по сути, она является просто функцией, принимающей в качестве аргументов две другие функции. Сначала следует добавить «оболочки» анонимных функций, чтобы вы могли гарантировать их правильную настройку, и только затем добавлять в них программный код.

▶ СОВЕТ

Вам следует часто тестировать свой код, чтобы удостовериться, что вы не допустили опечаток. В шаге 4 вы можете ввести код `console.log('mouseEnter')` внутри первой анонимной функции и код `console.log('mouseLeave')` внутри второй анонимной функции, а затем просмотреть страницу в браузере. Чтобы увидеть результаты, вам понадобится открыть консоль браузера: **F12** в Internet Explorer; **Ctrl+Shift+J** или **⌘+⌥+J** (OS X) в Chrome; **Ctrl+Shift+K** или **⌘+⌥+K** (OS X) в Firefox; и **⌘+⌥+C** в Safari. При наведении указателя мыши на элемент `div` в консоли должно появиться сообщение "mouseEnter", а при смещении указателя мыши — сообщение "mouseLeave". Если ни одно сообщение не появляется, то вы допустили ошибку. Перепроверьте свой код или используйте инструкцию по устранению ошибок при помощи консоли ошибок браузера.

5. **Внутри первой анонимной функции введите код `$(this).animate(); // окончание функции animate`.**

   Как говорилось в разделе «Ключевые слова `this` и `$(this)`» главы 4 в пределах события ключевое слово `$(this)` относится к элементу страницы, к которому прикреплено событие. В данном случае слово `$(this)` относится к элементу `div` с идентификатором `dashboard`. Другими словами, наведение указателя мыши на этот элемент анимирует его.

6. **Добавьте объектную константу со свойствами CSS, которые вы желаете анимировать:**

```
$(document).ready(function() {
$('#dashboard').hover(
function() {
$(this).animate(
{
left: '0',
backgroundColor: 'rgb(27,45,94)'
}
); // окончание функции animate
},
function() {
}
); // окончание функции hover
}); // окончание функции ready
```

   Первый аргумент для функции `animate()` — это объектная константа (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), содержащая свойства CSS. В данном случае элемент `div` в настоящий момент имеет значение свойства `left`, равное –92 пиксела, так что большая часть данного элемента скрыта за левым краем окна браузера. Анимируя изменение значения свойства `left` на 0, вы полностью отображаете этот элемент. А благодаря плагину jQueryUI, вы изменяете цвет фона с розового на темно-синий. Далее вы установите длительность анимации.

7. **Введите запятую после закрывающей фигурной скобки объектной константы, нажмите клавишу Enter и введите `500`.**

   Запятая означает окончание первого аргумента, переданного функции `animate()`, а значение 500 устанавливает длительность анимации на полсекунды (500 миллисекунд). Наконец, вы установите метод `easing`.

8. **Поставьте запятую после значения 500, нажмите клавишу Enter и введите `'easeInSine'`, чтобы ваш код выглядел так:**

```
$(document).ready(function() {
$('#dashboard').hover(
function() {
$(this).animate(
{
left: '0',
backgroundColor: 'rgb(27,45,94)'
},
500,
'easeInSine'
); // окончание функции animate
},
function() {
}
); // окончание функции hover
}); // окончание функции ready
```

   Последний аргумент функции `animate()` — `'easeInSine'` — позволяет функции использовать метод `easing`, который приводит к тому, что анимационный эффект начинается медленно, а затем ускоряется.

9. **Сохраните файл. Просмотрите его в браузере и наведите указатель мыши на элемент `div`.**

   Элемент должен полностью отобразиться. Если этого не происходит, используйте техники устранения ошибок, описанные в первой главе. Естественно, что при снятии указателя мыши с элемента ничего не происходит. Вам нужно добавить функцию `animate()` во вторую анонимную функцию.

10. Добавьте код во вторую анонимную функцию:

```
$(this).animate(
{
left: '-92px',
backgroundColor: 'rgb(255,211,224)'
},
1500,
'easeOutBounce'
); // окончание функции animate
```

Этот код обращает процесс первого анимационного эффекта, смещая элемент обратно за левый край окна и возвращая первоначальный розовый цвет фона. Длительность несколько иная — 1,5 секунды вместо полсекунды. Метод `easing` также иной.

11. Сохраните файл и просмотрите его в браузере. Наведите указатель мыши на элемент `div`, а затем уберите его с этого элемента.

Вы увидите, как элемент отображается и затем исчезает. Однако если вы будете быстро наводить и убирать курсор с элемента, то вы заметите странное поведение: элемент `div` продолжит появляться и исчезать еще некоторое время после того, как вы прекратите передвигать указатель мыши. Причина данной проблемы кроется в способе, с помощью которого библиотека jQuery создает очередь анимационных эффектов для элемента. Как уже говорилось, любой анимационный эффект, который вы добавляете к элементу, помещается в некую очередь для данного элемента. Например, если вам нужно, чтобы элемент появился, исчез и снова появился, библиотека jQuery запускает каждый эффект по порядку, один за другим.

Выполнение кода данного руководства приводит к тому, что каждый раз при наведении и смещении указателя мыши с элемента `div` анимационный эффект ставится в очередь, поэтому быстрое многократное наведение курсора на элемент создает длинный список эффектов, которые необходимо выполнить: отображение элемента, скрытие элемента, отображение элемента, скрытие элемента, и т. д. Решением данной проблемы является остановка всех анимационных эффектов перед запуском новой анимации. Другими словами, когда вы наводите указатель мыши на элемент `div`, который находится в процессе анимации, библиотека jQuery должна остановить выполняемую в данный момент анимацию и запустить эффект, со-

ответствующий событию наведения. К счастью, библиотека jQuery предусматривает функцию `stop()` для такого рода задач.

12. **Добавьте фрагмент `.stop()` между `$(this)` и `.animate` в двух анонимных функциях. Итоговый код должен выглядеть так (дополнения выделены полужирным шрифтом):**

```
$(document).ready(function() {
$('#dashboard').hover(
function() {
$(this).stop().animate(
{
left: '0',
backgroundColor: 'rgb(27,45,94)'
},
500,
'easeInSine'
); // окончание функции animate
},
function() {
$(this).stop().animate(
{
left: '-92px',
backgroundColor: 'rgb(255,211,224)'
},
1500,
'easeOutBounce'
); // окончание функции animate
}
); // окончание функции hover
}); // окончание функции ready
```

Функция `stop()` просто завершает любой анимационный эффект, прежде чем запускать новый и предотвращает создание очереди эффектов.

Сохраните страницу и просмотрите ее в браузере. Вы можете найти законченную версию данного руководства в файле *готовый_06_02.html* в папке *глава06*.

Библиотека jQuery, а также переходы и анимация CSS3

Если вы следите за последними и самыми передовыми техниками CSS, вам, вероятно, не ясно, зачем нужно использовать jQuery. В конце концов, с помощью одних только переходов CSS можно создавать анимацию между двумя разными стилями CSS, а анимация CSS позволяют обеспечить сложные анимированные эффекты (рис. 6.4).

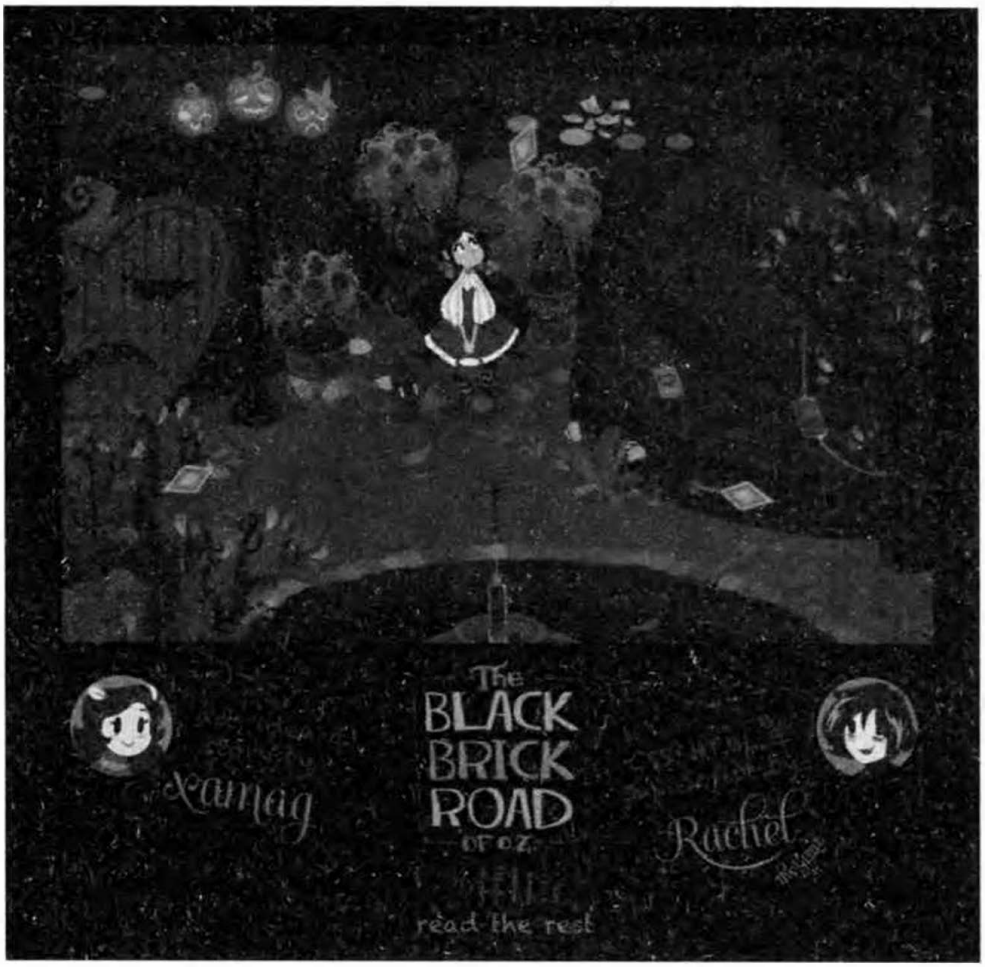

Рис. 6.4. Карикатурист, эксперт JavaScript и аниматор Рэйчел Нэйборс использует анимацию CSS3 и jQuery для создания целых анимированных приключенческих историй, которые можно найти по адресу: **codepen.io/rachelnabors/full/lqswg**. Она использует jQuery для запуска анимации, созданной с помощью CSS. К сожалению, этот передовой метод работает не во всех браузерах

В CSS появились потрясающие новые анимационные эффекты, однако они работают не во всех браузерах. Две по-прежнему широко распространенные версии Internet Explorer 8 и 9 не поддерживают ни переходы, ни анимацию CSS. Поэтому, если вы учитываете эти браузеры при планировании сайта, вам потребуется еще один способ для создания анимации.

В этом случае, обсуждаемые ранее в этой главе jQuery-функции являются вашим лучшим вариантом для обеспечения кроссбраузерной анимации.

ПРИМЕЧАНИЕ

Браузеры не могут анимировать все свойства CSS. Например, вы не можете анимировать свойство `font-family` путем изменения шрифта текста с одного на другой. Однако вы можете использовать многочисленные свойства CSS при создании переходов и анимации. Полный список анимируемых свойств приведен на странице: **developer.mozilla.org/ru/docs/Web/CSS/CSS_animated_properties**.

Тем не менее, даже если вы хотите использовать переходы и анимацию CSS, библиотека jQuery по-прежнему будет для вас большим подспорьем. В отличие от JavaScript, в CSS на самом деле нет никаких событий. Там предусмотрен псевдокласс `:hover`, который позволяет применять один стиль при наведении указателя мыши на элемент и возвращаться в предыдущему стилю при смещении его с элемента. А псевдокласс `:active` обеспечивает своего рода имитацию щелчка. Однако для таких событий, как двойной щелчок, прокрутка и нажатие определенных клавиш эквивалентов CSS не существует. Это означает, что вы не можете использовать только CSS для запуска анимации, когда посетитель вводит данные в текстовое поле или дважды щелкает по кнопке. Кроме того, невозможно только с помощью CSS запустить анимацию на одном элементе, когда посетитель взаимодействует с другим элементом в другом месте страницы: например, нельзя отобразить элемент `div` в другом месте страницы, щелкнув по кнопке «Отобразить настройки» в ее верхней части.

ПРИМЕЧАНИЕ

В этом пособии нет детального описания переходов и анимации CSS. Если вы хотите узнать больше, прочтите работу «Большая книга CSS3» Дэвида МакФарланда. Для быстрого ознакомления с переходами и анимацией CSS посетите страницы **www.css3files.com/transition/** и **www.css3files.com/animation/**.

Библиотека jQuery и переходы CSS

Переход CSS позволяет анимировать изменение свойств CSS. Самый простой способ сделать это — применить к элементу новый стиль,

а затем анимировать это изменение. Например, вы можете создать стиль класса для кнопки — `.button`, которая имеет синий фон. С помощью псевдокласса `:hover` — `.button:hover` — несложно изменить цвет этой кнопки на желтый. Добавляя свойство перехода к стилю `.button`, вы даете браузеру задание анимировать изменение цвета с синего на желтый, когда посетитель наводит указатель мыши на кнопку, а также анимировать изменение цвета с желтого на синий, когда посетитель смещает его с этой кнопки.

Вы можете добавить переход, чтобы анимировать изменение одного свойства CSS на другое. Например, вы хотите обеспечить исчезновение картинок на странице с помощью переходов CSS. В этом случае вы бы начали с создания стиля для изображения следующим образом:

```
img {
opacity: 1;
}
```

Свойство CSS `opacity` контролирует степень непрозрачности элемента. Значение `1` соответствует абсолютной непрозрачности, а значение `0` — полной прозрачности. Далее можно создать класс CSS, который устанавливает значение непрозрачности равным `0`:

```
img.faded {
opacity: 0;
}
```

Для анимации изменения между этими двумя стилями используется свойство CSS `transition`. Вам следует добавить это свойство к исходному стилю (тому, который первым был применен к элементу на странице). В данном случае это стиль `img`. Кроме того, чтобы убедиться, что это работает во всех браузерах, поддерживающих переходы CSS, вам потребуется использовать префиксы вендоров:

```
img {
opacity: 1;
-webkit-transition: opacity 1s;
-moz-transition: opacity 1s;
-o-transition: opacity 1s;
transition: opacity 1s;
}
```

В этом примере вы создаете инструкцию для анимации любых изменений в степени непрозрачности. Вы также хотите, чтобы анимация длилась 1 секунду, на что указывает фрагмент кода `1s`.

ПРИМЕЧАНИЕ

Когда производители браузеров добавляют в свои программы новые и инновационные свойства CSS, они обычно начинаются с префикса вендора, указанного перед именем свойства, например, `-webkit-transition`. Этот префикс позволяет производителю браузера безопасно протестировать новую функцию и исправить все ее недочеты. Как только будет достигнуто согласие в отношении принципа работы нового свойства CSS, обновленные версии браузеров станут использовать стандартное имя свойства CSS без префикса, например, `transition` вместо `-webkit-transition`.

Теперь, когда переход добавлен, при назначении для изображения класса `faded` степень его непрозрачности будет изменена со 100% до 0% за 1 секунду. Это эквивалентно использованию jQuery-функции `fadeOut()` со значением продолжительности 1. Главное заключается в присвоении изображению класса `faded`. Вот где может помочь jQuery. Если вы хотите применить стиль в ответ на щелчок посетителя по картинке, используйте функцию `click()` следующим образом:

```
$('img').click(function() {
$(this).addClass('faded');
}
```

В данном случае, когда посетитель щелкнет по картинке, библиотека jQuery просто присвоит ему класс. После этого веб-браузер сделает всю тяжелую работу по анимации изменения степени непрозрачности (рис. 6.5). Если вы хотите, чтобы при повторном щелчке изображение вернулось к исходному состоянию, вы можете использовать функцию `toggleClass()`:

```
$('img').click(function() {
$(this).toggleClass('faded');
}
```

Функция `toggleClass()` добавляет класс, если он еще не применен к элементу, и удаляет его, если он применен. Поскольку в CSS нет селектора, предусмотренного на случай, когда посетитель щелкает по чему-либо, библиотека jQuery предлагает простой способ запуска переходов CSS.

ПРИМЕЧАНИЕ

Установка степени непрозрачности элемента не удаляет этот элемент со страницы. Он по-прежнему занимает место на странице (и в DOM), но его не видно. Вот почему работает пример с функцией `toggleClass()`. Вы по-прежнему можете щелкнуть по невидимому изображению, чтобы удалить класс `faded`. Тем не менее, если элемент на самом деле скрыт, например, при использовании `display: hidden`, то он удален со страницы, и по нему нельзя щелкнуть второй раз.

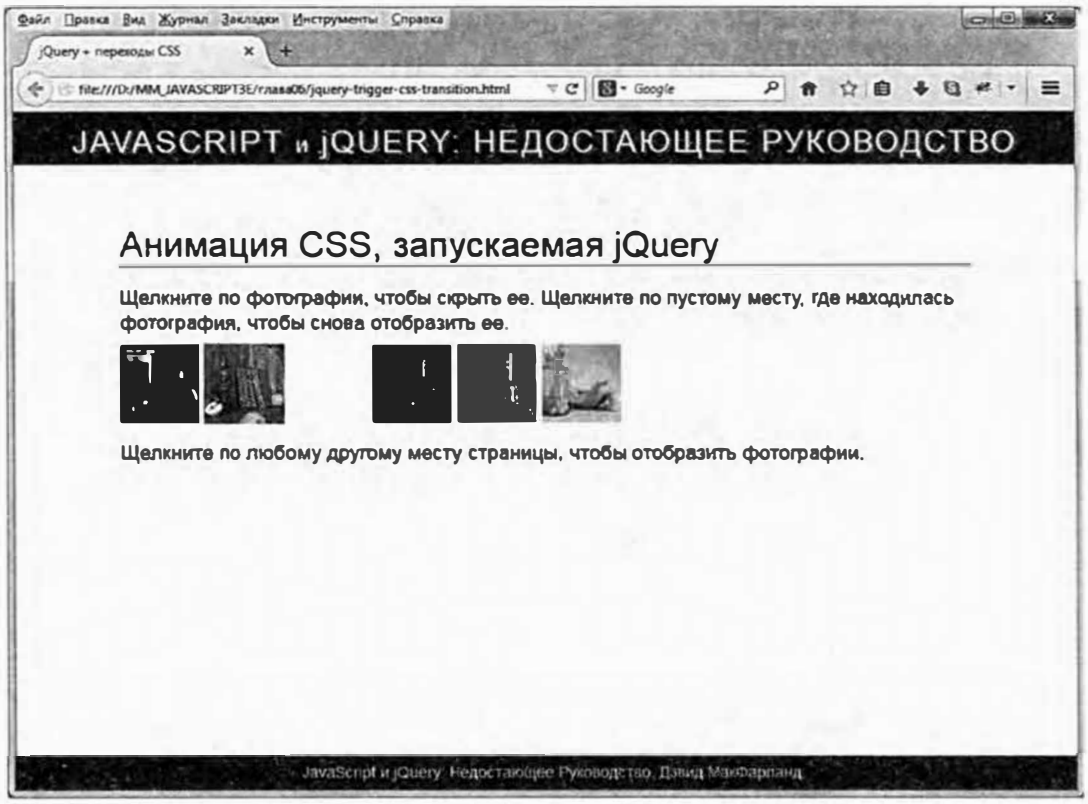

Рис. 6.5. jQuery и переходы CSS хорошо сочетаются. Используя обработчики событий jQuery, вы можете запускать анимацию переходов CSS. В данном примере щелчок по изображению приводит к его исчезновению из вида. Щелчок по пустому месту заставляет картинку снова появиться. Этот код не будет работать в программе Internet Explorer 9 или в более ранней версии, поскольку эти браузеры не поддерживают переходы CSS. Вы можете найти работающую версию этой страницы в файле *jquery-trigger-cssanimation.html*, в папке *глава06*

jQuery и анимация CSS

Возможности анимации CSS обеспечивают гораздо больший контроль по сравнению с простыми переходами. С их помощью вы создаете ключевые кадры, которые определяют свойства CSS на определенных этапах анимации. Например, вы можете сделать так, чтобы цвет кнопки

менялся много раз — с синего на красный, потом на оранжевый, затем на зеленый, или переместить элемент `div` в верхнюю часть экрана, затем в нижнюю, влево и, наконец, вправо. Другими словами, хотя переходы CSS позволяют вам указать начальный и конечный стили анимации, анимационные возможности CSS позволяют вам указать любое количество промежуточных стилей, используемых от начала анимации и до ее окончания.

CSS-анимации могут быть довольно сложными и быстрыми. Вы можете найти соответствующие ресурсы, чтобы подробнее узнать о том, как их создавать. В качестве введения данный раздел содержит небольшой пример использования jQuery для запуска анимации. Допустим, вам нужно, чтобы при нажатии кнопки воспроизведения элемент `div` изменил свой цвет и стал шире. (Возможно, вам нужно выделить и отобразить скрытый текст внутри этого элемента.)

Первым шагом является создание анимации. Это делается с помощью директивы `@keyframes`:

```
@keyframes growProgressBar{
0% {
width: 0%;
background-color: red;
}
50% {
background-color: yellow;
}
100% {
width:88%;
background-color: green;
}
}
```

Этот код может показаться вам незнакомым, однако он просто создает анимацию под названием `growProgressBar` с серией ключевых кадров. Каждый ключевой кадр определяет одно или несколько свойств CSS, которые будут изменяться в процессе воспроизведения анимации. Например, первый ключевой кадр в приведенном выше коде — `0%` — это начало анимации, он указывает на то, что значение ширины элемента должно быть равно 0%, а цветом фона должен быть красный.

ПРИМЕЧАНИЕ

Браузеры Chrome и Safari требуют использования префиксов вендоров — `@webkit-keyframes`, чтобы этот код работал. Кроме того, в Internet Explorer 9 и более ранних версиях этой программы он не станет работать вообще.

Между ключевыми кадрами этот элемент будет изменять цвет с красного на желтый, а затем на зеленый. Процентное значение для каждого ключевого кадра соответствует моменту, в который должно появиться указанное значение CSS. А теперь представьте, что эта анимация будет длиться 10 секунд (вы укажите продолжительность анимация отдельно чуть позднее). В начале анимации (0% от 10 секунд) цвет фона окажется красным, а ширина элемента будет равна 0%. Спустя 5 секунд (в середине анимации) цвет станет желтым. Наконец, спустя 10 секунд, когда анимация подойдет к концу, элемент будет зеленым, а его ширина — равна 88%. Поскольку в приведенном выше примере ширина задана только в первом и последнем ключевых кадрах, эта ширина станет меняться с 0% до 88% в течение всего времени воспроизведения эффекта анимации.

После создания ключевых кадров разрешается применить этот анимационный эффект к любому количеству элементов. Например, у вас есть следующий HTML-элемент: `<div class="progressBar">`. Вы можете применить данный анимационный эффект к этому элементу `div` таким образом:

```
.progressBar {
animation-name: growProgessBar;
animation-duration 10s;
animation-fill-mode: forwards;
}
```

Приведенный код применяет к этому элементу анимационный эффект продолжительностью в 10 секунд. Последняя строка CSS — `animation-fill-mode: forwards;` — просто гарантирует то, что по окончании воспроизведения анимации элемент сохранит свойства последнего ключевого кадра. То есть ширина элемента `div` должна быть равна 88%, а цвет фона должен быть зеленым. (Без этого параметра элемент вернется к тем стилям, которые были до анимации.)

Тем не менее воспроизведение анимации в данном коде начнется сразу после загрузки веб-страницы. Вам нужно сделать так, чтобы ани-

мация запускалась в ответ на щелчок по кнопке. Вы можете приостановить воспроизведение анимации (то есть остановить его прежде, чем оно начнется) с помощью другого свойства анимации: `animation-play-state`. Чтобы предотвратить немедленное воспроизведения анимации, добавьте это свойство к стилю со значением `paused`:

```
.progressBar {
animation-name: growProgessBar;
animation-duration 10s;
animation-fill-mode: forwards;
animation-play-state: paused;
}
```

Добавление jQuery для запуска анимации — это несложно. Все, что вам нужно сделать для запуска анимации, — это изменить значение свойства `animation-play-state` на `running`. Вы можете легко это сделать с помощью функции `css()`. Например, можно создать кнопку с идентификатором `start`, нажатие которой станет запускать анимацию. Вот, как будет выглядеть код jQuery:

```
$('#start').click(function() {
$('.progressBar').css('animation-play-state', ↵
'running');
});
```

Если у вас есть кнопка паузы с идентификатором `pause`, вы также могли бы использовать ее для приостановки воспроизведения анимация:

```
$('#pause').click(function() {
$('.progressBar').css('animation-play-state', ↵
'paused');
});
```

К счастью, jQuery «знает» о префиксах вендоров. Когда вы указываете значение свойства CSS, требующего наличия префиксов, библиотека jQuery также устанавливает версии свойств CSS с префиксами вендоров. Спасибо тебе, jQuery!

Другой подход заключается в создании ключевых кадров и отдельного стиля класса, в котором определены все свойства анимации. Это выглядит примерно так:

```
.animateDiv {
animation-name: growProgessBar;
animation-duration 10s;
animation-fill-mode: forwards;
}
```

При загрузке страницы к элементу, который вы хотите анимировать, не будет применен стиль класса `.animateDiv`. В результате анимация этого элемента не станет запускаться сразу, а это именно то, что вам нужно. Далее вы используете jQuery, чтобы присвоить этот класс данному элементу. Как только элемент получит новый класс, начнется воспроизведение анимации. Этот подход позволяет вам обойтись без свойства `animation-play-state`:

```
$('#start').click(function() {
$('.progressBar').addClass('animateDiv');
});
```

Вы найдете примеры использования обоих подходов в файлах *jquerytrigger-css-animation1.html* и *jquery-trigger-css-animation2.html* в папке *глава06*.

На данный момент с использованием CSS-анимации есть некоторые сложности. Как уже упоминалось, программа Internet Explorer 9 и более ранние версии ее вообще не поддерживают. Кроме того, в jQuery нелегко отследить прогресс CSS-анимации.

ПРИМЕЧАНИЕ

В конце концов, разработчики, вероятно, придут к использованию CSS-анимации в сочетании с JavaScript. Консорциум W3C и производители браузеров работают над многочисленными способами, позволяющими контролировать CSS-анимацию с помощью JavaScript путем добавления новых событий, которые отслеживают прогресс CSS-анимации. Если вы хотите запустить код jQuery после окончания воспроизведения CSS-анимации, обратитесь к статье: **blog.teamtreehouse.com/using-jquery-to-detect-when-css3-animations-and-transitions-end**.

Глава 7

РАСПРОСТРАНЕННЫЕ ЗАДАЧИ, РЕШАЕМЫЕ С ПОМОЩЬЮ JQUERY

Веб-дизайнеры используют изображения для усовершенствования дизайна страницы, украшения навигационных панелей и выделения элементов. Ссылки являются основой Всемирной паутины, они позволяют посетителям переходить от одного фрагмента информации к другому, кроме того, вы можете контролировать работу этих ссылок, будь то в этом же окне браузера или в новом. При наличии многочисленных ссылок вам нужен способ, позволяющий сгруппировать их в хорошей навигационной панели. Библиотека jQuery может сделать элементы страницы более интерактивными и интересными. В данной главе вы узнаете, как использовать библиотеку jQuery для повышения эффективности работы изображений, ссылок, окон и навигационных панелей.

Смена изображений

Обычно язык JavaScript применяется при работе с *изображениями, сменяющими друг друга*: когда вы наводите указатель мыши на изображение, оно сменяется другим. Эта простейшая техника использовалась с начала времен JavaScript для создания интерактивных навигационных панелей, в которых вид кнопок изменяется в зависимости от того, наведен ли на них указатель мыши.

Но в последнее время очень многие дизайнеры обратились к языку CSS для достижения такого же эффекта. Однако даже если для создания интерактивных навигационных панелей применять CSS, вам все же придется понять, как использовать язык JavaScript для изменения одной картинки на другую при создании слайд-шоу, галерей изображений, а также для добавления других типов интерактивных графических эффектов на веб-страницу.

Изменение атрибута src изображения

Каждое изображение, отображенное на веб-странице, имеет атрибут `src` (сокращенно от *source*), обозначающий путь к графическому файлу, другими словами, путь к файлу на веб-сервере. Если вы меняете это свойство, чтобы указать на другой графический файл, то браузер покажет иную картинку. Используя библиотеку jQuery, вы можете динамически изменять атрибут изображения `src`.

Например, на странице есть картинка, которой вы присвоили идентификатор `photo`. HTML-код будет выглядеть примерно так:

```
<img src="images/image.jpg" width="100" height="100" ↵
id="photo">
```

Чтобы заменить файл изображения, вы используете функцию `attr()` (см. раздел «Чтение, установка и удаление атрибутов HTML» главы 4) для установки атрибута `src` нового файла:

```
$('#photo').attr('src','images/newImage.jpg');
```

ПРИМЕЧАНИЕ

Если вы поменяете атрибут `src` изображения, используя язык JavaScript, то путь к иллюстрации будет основан на местоположении страницы, а не JavaScript-кода. Это может запутать, если вы работаете с внешним файлом JavaScript (см. раздел «Внешние файлы JavaScript» главы 1), размещенным в ином каталоге. В примере, приведенном выше, браузер попытается загрузить файл *newImage.jpg* из каталога *images*, который находится в той же папке, что и веб-страница. Данный прием работает, даже если код сохранен во внешнем файле в другом каталоге в любом месте на сайте. Соответственно, часто бывает проще использовать ссылки относительно корневого каталога во внешних файлах JavaScript (см. врезку «Типы URL-адресов» главы 1, чтобы узнать о различных типах ссылок).

Изменение атрибута `src` изображения не затрагивает всех остальных атрибутов элемента `img`. Например, если атрибут `alt` установлен в HTML-коде, то измененное изображение будет иметь тот же атрибут `alt`, что и оригинал. Дополнительно, если атрибуты `width` и `height` установлены в HTML-коде, то изменение свойства изображения `src` подгонит новую картинку под размер оригинала. Если два рисунка имеют различные размеры, то новое изображение будет искажено.

В ситуации со сменяемыми картинками на навигационной панели лучше всего, чтобы оба изображения были одного и того же размера и имели один и тот же атрибут `alt` — тогда проблемы не будет. Вы можете избежать искажения изображения, просто опустив свойства `width` и `height` оригинальной иллюстрации в вашем HTML-коде. Тогда при смене картинки на новую браузер отобразит ее на экране в том размере, который задан в файле.

Другое решение — это сначала загрузить новое изображение, получить его размеры, а затем изменить атрибуты `src`, `width`, `height` и `alt` для элемента `img`:

```
1 var newPhoto = new Image();
2 newPhoto.src = 'images/newImage.jpg';
3 var photo = $('#photo');
4 photo.attr('src',newPhoto.src);
5 photo.attr('width',newPhoto.width);
6 photo.attr('height',newPhoto.height);
```

ПРИМЕЧАНИЕ

Числа слева — это просто нумерация строк, которая облегчает чтение кода. Не записывайте эти числа в ваш код!

Ключом к данной технике является строка 1, создающая новый объект изображения. Код `new Image()` сообщает браузеру: «Я собираюсь добавить на страницу новое изображение, так что будь готов». В следующей строке содержится сама команда для браузера — загрузить новую картинку. В строке 3 дается ссылка на текущее изображение на странице, а в строках 4—6 осуществляется переход к новому изображению, меняются высота и ширина для соответствия новой иллюстрации.

СОВЕТ

JQuery-функция `attr()` может одновременно устанавливать несколько атрибутов HTML. Просто передайте ей объектную константу (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), содержащую имя и новое значение каждого атрибута. Вы можете написать код jQuery, приведенный выше, более кратко:

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
```

```
$('#photo').attr({
src: newPhoto.src,
width: newPhoto.width,
height: newPhoto.height
});
```

Обычно вы использовали бы эту технику смены изображения вместе с обработчиком событий. Например, вы можете запрограммировать смену картинки при наведении на нее указателя мыши. Этот эффект переключения часто применяется при создании навигационных панелей. Однако допустимо изменять изображение в ответ на любое событие: например, заставить новую картинку появляться каждый раз при щелчке по стрелке на странице, как в случае со слайд-шоу.

Смена изображений с помощью jQuery

Существует еще один способ смены изображений, который не подразумевает изменения атрибута `src` или отдельного изменения других атрибутов изображения. Как вы помните из раздела «Добавление содержимого на страницу» главы 4, библиотека jQuery позволяет производить быстрые изменения HTML-страницы. Вы можете использовать jQuery для добавления, удаления и изменения HTML-кода. Простой способ замены одного изображения другим заключается в замене элемента `img` исходной картинки новым элементом `img` с помощью метода jQuery в `replaceWith()`.

Например, на вашей странице присутствует такое изображение:

```
<img src="sad.png" alt="Sad Face" height="50" ↵
width="50" id="swap">
```

Вы можете заменить его другим элементом `img` так:

```
$('#swap').replaceWith('<img src="happy.png" ↵
alt="Радостное лицо" height="100" width="150" ↵
id="swap">');
```

Метод jQuery `replaceWith()` заменяет выбранный в данный момент элемент любым HTML-кодом, который вы ему передаете. С помощью этого метода можно устанавливать различные атрибуты `src`, `alt`, `width` и `height` в строке, которую вы предоставляете методу `replaceWith()`,

например, <img src="happy.png" alt="Радостное лицо" height="100" width="150" id="swap">. Это, как правило, проще, чем менять каждый из этих атрибутов по отдельности, как вы знаете из предыдущего раздела.

СОВЕТ

Метод `replaceWith()` возвращает HTML-код, который заменяет jQuery. Другими словами, вы можете сохранить удаляемый HTML-код. Например, если вы хотите заменить изображение, но сохранить его HTML-код для последующего использования, вы можете сделать так:

```
var oldImage = $('#swap').replaceWith('<img ↵
src="happy.png" alt="Радостное лицо" height="100" ↵
width="150" id="swap">');
```

Теперь переменная `oldImage` содержит HTML-код, который был заменен. В случае с приведенным выше примером эта переменная содержит код:

```
<img src="sad.png" alt="Грустное лицо" height="50" ↵
width="50" id="swap">
```

Теперь вы можете использовать переменную `oldImage` снова, например, для возврата картинки на место.

Предварительная загрузка изображений

Есть одна проблема с переключением на новое изображение с помощью описанного выше способа. Когда вы вставляете новый путь к файлу в атрибут `src`, браузеру необходимо загрузить файл. Если кто-нибудь подведет указатель мыши к изображению до того, как загрузилась новая картинка, возникнет неприятная задержка, прежде чем появится изображение. В случае с навигационной панелью данный эффект приведет к ее замедленной работе.

Чтобы избежать такой задержки, необходимо предварительно загрузить изображения, чтобы они появлялись непосредственно в ответ на действие. Например, когда посетитель наводит указатель мыши на кнопку навигационной панели, картинка должна появиться сразу же. *Предварительная загрузка* изображения означает приказ браузеру загрузить картинку до того, как вы планируете ее отобразить. Когда изображение загружено, оно сохраняется в кэше браузера, чтобы все последующие запросы обслуживались с жесткого диска посетителя, а не загружались с сервера заново.

Предварительная загрузка картинки не сложнее, чем создание нового объекта изображения и установки для него свойства `src`. На самом деле вы уже знаете, как это делается:

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
```

Предварительная загрузка заключается в том, что вы выполняете ее до того, как вам необходимо будет заменить текущее изображение. Один из способов предварительной загрузки — это создание в начале сценария массива (см. раздел «Массивы» главы 2), содержащего пути ко всем рисункам, которые вы хотите предварительно загрузить, а затем — проработка этих путей в цикле с созданием для каждого из них нового объекта-изображения:

```
var preloadImages = ['images/roll.png',
'images/flower.png',
'images/cat.jpg'];
for (var i=0; I < preloadImages.length; i++) {
new Image().src = preloadImages[i];
}
```

В строках 1—3 размещается единая инструкция языка JavaScript, создающая массив `preloadImages`, в котором содержатся три значения — пути к каждому из графических файлов, предназначенных для предварительной загрузки (как было указано в разделе «Создание массива» главы 2, массив часто бывает проще читать, если каждый из его элементов разместить в отдельной строке). В строках 4–6 показан простейший цикл JavaScript `for` (см. раздел «Циклы `for`» главы 3), запускаемый один раз для каждого элемента массива `preloadImages`. В строке 5 создается новый объект изображения и устанавливается его атрибут `src`, соответствующий пути к файлу из массива `preloadImages` — и все это «волшебство» заставляет картинку загружаться.

Вы можете использовать библиотеку jQuery, чтобы обеспечить этот же эффект, и обойтись при этом без функции `new Image()`:

```
var preloadImages = ['images/roll.png',
'images/flower.png',
'images/cat.jpg'];
for (var i=0; i < preloadImages.length; i++) {
```

```
$('<img>').attr('src',preloadImages[i]);
}
```

В строке 5 с помощью jQuery вы создаете новый элемент `img`. Этот новый для вас метод может показаться несколько сложным. Передавая jQuery тег `<img>` (вместе с символами < и >), вы создаете новый HTML-элемент. Обычно в jQuery вы опускаете символы < и >, как в случае с `$('img')`. Это позволяет jQuery выбрать элементы присутствующих на странице изображений. Как видите, jQuery может не только выбрать существующие HTML-элементы, но и создать новые.

В оставшейся части кода — `.attr('src', preloadImages[i])` — используется функция `attr()` (см. раздел «Чтение, установка и удаление атрибутов HTML» главы 4). Она устанавливает значение свойства `src`, соответствующее пути к файлу нового изображения, что позволяет веб-браузеру предварительно загружать его.

Оба способа предварительной загрузки изображений (с использованием и без использования jQuery) работают одинаково хорошо, поэтому вы можете выбрать тот, который подходит вам лучше всего.

Сменяемые изображения

Сменяемое изображение (или *ролловер*) — это просто два изображения, которые сменяют друг друга (см. раздел «Смена изображений» в начале данной главы), когда посетитель наводит указатель мыши на картинку. Иными словами, вы присваиваете событию наведения курсора изменение картинки. Например, на странице есть изображение с идентификатором `photo`. Когда на него наводится указатель мыши, вы хотите, чтобы появилась новая картинка. Этого можно достичь с помощью библиотеки jQuery:

```
1  <script src="js/jquery.min.js"></script>
2  <script>
3  $(document).ready(function() {
4     var newPhoto = new Image();
5     newPhoto.src = 'images/newImage.jpg';
6     $('#photo').mouseover(function() {
7           $(this).attr('src', newPhoto.src);
8  }); // окончание события mouseover
```

```
9  }); // окончание функции ready
10 </script>
```

Строка 3 приказывает подождать, пока загрузится HTML-код, чтобы JavaScript получил доступ к HTML текущего фото. В строках 4 и 5 происходит предварительная загрузка изображения, которым вы хотите заменить имеющееся. В оставшемся коде изображению присваивается событие `mouseover`, а также функция, изменяющая атрибут `src` для соответствия новой фотографии.

Поскольку сменяемые картинки обычно возвращаются в исходное состояние, как только вы отводите указатель мыши в сторону, вы также должны добавить событие `mouseout` для возврата к изначальному изображению. Как обсуждалось в разделе «События наведения и снятия указателя мыши» главы 5, библиотека jQuery предусматривает собственное событие `hover()`, которое отвечает за оба события наведения (`mouseover`) и смещения (`mouseout`) указателя мыши:

```
1  <script src="js/jquery.min.js"></script>
2  <script>
3  $(document).ready(function() {
4     var newPhoto = new Image();
5     newPhoto.src = 'images/newImage.jpg';
6     var oldSrc=$('#photo').attr('src');
7     $('#photo').hover(
8  function() {
9     $(this).attr('src', newPhoto.src);
10 },
11 function() {
12    $(this).attr('src', oldSrc);
13 }); // окончание события hover
14 }); // окончание функции ready
15 </script>
```

Функция `hover()` принимает два аргумента: первый — это функция, сообщающая браузеру, что делать, когда указатель мыши наведен на картинку; второй — это функция, сообщающая браузеру, что делать, если указатель смещается с изображения. В данном коде также добавляется переменная `oldSrc` для отслеживания исходного атрибута `src` — пути к файлу, появляющемуся при загрузке страницы.

Вы не ограничены сменой только изображений. Вы можете добавить функцию `hover()` к любому элементу: ссылке, элементу формы и даже абзацу.

Таким образом, каждый элемент на странице может инициировать изменение любого изображения. Например, вы хотите, чтобы фото сменялось другим при наведении указателя мыши на элемент `h1` страницы.

Допустим, целевое изображение то же, что и в предыдущем примере. Просто измените код, выделенный полужирным шрифтом:

```
<script src="js/jquery.min.js"></script>
<script>
$(document).ready(function() {
   var newPhoto = new Image();
   newPhoto.src = 'images/newImage.jpg';
   var oldSrc = $('#photo').attr('src');
   $('h1').hover(
         function() {
               $('#photo').attr('src', newPhoto.src);
         },
         function() {
         $('#photo').attr('src', oldSrc);
}); // окончание события hover
}); // окончание функции ready
</script>
```

Добавление сменяемых изображений на практике

В данном примере вы добавите эффект переката серии изображений (рис. 7.1), а также программный код для предварительной загрузки сменяемых картинок с целью устранения задержки между наведением указателя мыши на иллюстрацию и просмотром сменяемого изображения. Дополнительно вы освоите новую технику повышения результативности процесса предварительной загрузки и добавления эффекта переката.

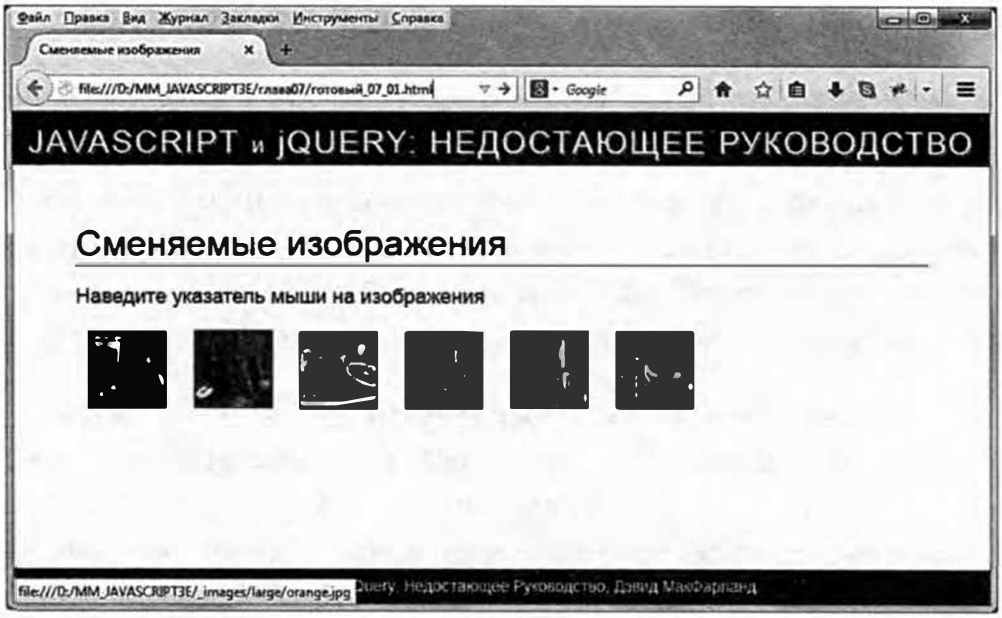

Рис. 7.1. Сделайте навигационную панель, ссылку или просто фотографию визуально более интерактивными с помощью сменяемых изображений

Обзор задачи

Файл примера *07_01.html* из каталога *глава07* содержит серию из шести изображений (рис. 7.2). Каждая картинка заключена в ссылку, указывающую на большую версию фотографии, все изображения содержатся в элементе `div` с идентификатором `gallery`.

Рис. 7.2. HTML-код для данного руководства содержит элемент `div`, включающий серию ссылок, содержащих изображения. Чтобы упростить смену картинки, имя нового изображения является вариантом имени исходного изображения

Вы пытаетесь решить две основные задачи.

- Предварительно загрузить сменяемое изображение, связанное с каждым изображением внутри элемента `div`.
- Прикрепить функцию `hover()` к каждой из картинок в рамках элемента `div`. Функция `hover()` меняет переключаемое изображение, когда на него наведен указатель мыши, а затем возвращает исходную картинку, когда курсор смещается с изображения.

Из этого описания вы можете заключить, что оба шага связаны с изображениями внутри элемента `div`, то есть один из подходов к этой задаче — сначала выбрать изображения внутри элемента `div`, а затем проработать выборку в цикле, предварительно загружая сменяемый вариант для каждого из изображений и прикрепляя к нему функцию `hover()`.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Верстка кода

1. **В текстовом редакторе откройте файл *07_01.html* из каталога *глава07*.**

   В этом документе уже содержатся ссылка на файл jQuery и функция `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5). В первом шаге выбираются все изображения в элементе `div` и задается цикл с jQuery-функцией `each()`, которая обсуждалась в разделе «Работа с каждым элементом выборки» главы 4.

2. **Щелкните мышью в пустой строке после функции `$(document).ready()` и введите код `$('#gallery img').each(function() {`.**

   Селектор `#gallery img` выбирает все элементы `img` в элементе, имеющем идентификатор `gallery`. JQuery-функция `each()` обеспечивает быстрый способ проработки в цикле группы элементов страницы, осуществляя над каждым из элементов серию действий. Функция `each()` принимает в качестве аргумента анонимную функцию (см. раздел «Анонимные функции» главы 4). Хорошая идея — закрыть анонимную функцию и функцию `each()` перед не-

посредственным написанием кода, который выполняется внутри функции, и это будет сделано далее.

3. **Дважды нажмите клавишу Enter и введите код `});  // конец each` для закрытия анонимной функции, завершите вызов функции `ready()` и закончите инструкцию JavaScript. Ваш код должен выглядеть так:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
      $('#gallery img').each(function() {

      }); // конец each
}); // конец ready
```

Здесь сценарий просматривает в цикле каждое из изображений галереи, но не делает ничего более. Первый пункт на повестке дня — взять свойство изображения `src` и сохранить его в переменной, которая будет использоваться далее в сценарии.

ПРИМЕЧАНИЕ

Комментарии JavaScript `// конец each` и `// конец ready` необязательны для работы этого сценария. Однако они помогают понять, с какой из частей сценария связана та или иная строка.

4. **Щелкните мышью в пустой строке (строка 5 в шаге 3) и напечатайте код:**

```
var imgFile = $(this).attr('src');
```

Как было сказано в разделе «Ключевые слова `this` и `$(this)`» главы 4, вы можете использовать ключевое слово `$(this)` для ссылки на элемент, обрабатываемый в цикле в настоящий момент. Другими словами `$(this)` будет относиться к каждому из элементов изображений по очереди. JQuery-функция `attr()` (см. раздел «Чтение, установка и удаление атрибутов HTML» главы 4) возвращает указанный атрибут HTML. В данном случае она возвращает свойство `src` изображения и сохраняет его в переменной `imgFile`. Например, для первого изображения свойство `src` — *images/small/blue.jpg*, что является путем к изображению, появляющемуся на странице.

Вы можете использовать данное значение `src` для предварительной загрузки изображения.

5. **Нажмите клавишу Enter и создайте пустую строку, а затем добавьте следующие строки кода:**

```
var preloadImage = new Image();

var imgExt = /(\.\w{3,4}$)/;

preloadImage.src = imgFile.replace(imgExt,'_h$1');
```

Для предварительной загрузки изображения вы должны сначала создать объект изображения. В данном случае переменная `preloadImage` создается для сохранения объекта изображения. Далее предварительная загрузка происходит при задавании свойства `src` объекта изображения.

Один из способов предварительной загрузки изображений (описанный в разделе «Предварительная загрузка изображений») — это создать массив изображений, которые вы желаете предварительно загрузить, а затем проработать в цикле каждый из элементов массива, создавая объект изображения и добавляя к нему источник изображения. Однако данный подход требует очень много труда, так как вам необходимо знать путь к каждому из сменяющихся изображений и вводить эти пути в массив.

В данном примере вы будете использовать более эффективный (и менее трудоемкий) метод предварительной загрузки изображений. Вы лишь должны убедиться, что сохранили сменяемое изображение там же, где и исходное изображение, и назвали его похожим именем. Для данной веб-страницы каждое изображение имеет соответствующее сменяемое изображение с символами *_h*, добавленными к концу имени файла. Например, для изображения *blue.jpg* сменяемое изображение называется *blue_h.jpg*. Оба файла сохранены в одном и том же каталоге, поэтому путь к ним один и тот же.

А теперь — творчество. Вместо набора вручную атрибута `src` сменяемого изображения для его предварительной загрузки (`preloadImage.src='_images/small/blue_h.jpg'`), вы можете позволить языку JavaScript выяснить атрибут `src`, просто изменив имя источника оригинального изображения так, чтобы оно отражало имя сменяемого изображения. Другими словами, если вы знаете путь к изображению на странице, то имя сменяющего изображения просто будет содержать символы *_h* перед расширением

.jpg. Так *_images/small/blue.jpg* становится *_images/small/blue_h.jpg*, а *_images/small/orange.jpg* становится *_images/small/orange_h.jpg*.

Именно это делают две оставшиеся строки кода. Первая (`var imgExt = /(\.\w{3,4}$)/;`) создает *регулярное выражение*. Регулярное выражение (о котором вы узнаете в разделе «Поиск по маске в строках» главы 16) — это последовательность символов, которую вы можете искать в строке, например, три цифры подряд. Регулярные выражения могут вызывать сложности, но, в сущности, данное выражение находит точку, за которой следуют три или четыре символа в конце строки, например, *.jpg* в */images/small/blue.jpg* и *.png* в */images/orange.png*.

Метод `replace()` используется в следующей строке (`preloadImage.src = imgFile.replace(imgExt,'_h$1');`) (см. раздел «Замена текста» главы 16) для замены соответствующего текста чем-то другим. Окончание имени файла будет заменено на *_h*, поэтому *images/small/blue.jpg* и изменится на *images/small/blue_h.jpg*. Данная техника слегка замысловата, поскольку использует подмаску регулярного выражения (см. врезку «Использование подмасок для замены текста» главы 16), поэтому не переживайте, если вам не вполне ясно, как она работает.

Теперь, когда переключаемое изображение уже предварительно загружено, вы можете присвоить ему событие `hover()`.

6. **Нажмите клавишу Enter и введите код, приведенный ниже в строках 9—11:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4 $('#gallery img').each(function() {
5  var imgFile = $(this).attr('src');
6  var preloadImage = new Image();
7  var imgExt = /(\.\w{3,4}$)/;
8  preloadImage.src=imgFile.replace(imgExt,'_h$1');
9  $(this).hover(
10
11 ); // конец hover
12 }); // конец each
13 )); // конец ready
```

Функция `hover()` — краткий метод применения событий `mouseenter` и `mouseleave` к элементу. Чтобы она заработала, ей в качестве аргументов передаются две функции.

Первая функция запускается при наведении указателя мыши на элемент — тогда изображение изменяется на его альтернативный вариант.

Вторая функция работает, когда указатель мыши смещается с элемента — сменяемое изображение возвращается в исходное состояние.

7. **В пустую строку (строка 10 в шаге 6) добавьте следующие три строки кода:**

```
function() {
$(this).attr('src', preloadImage.src);
},
```

Первая функция просто изменяет свойство `src` текущего изображения на `src` альтернативного изображения. В конце последней строки необходима запятая, поскольку только что добавленная вами функция работает как первый аргумент при вызове функции `hover()` — запятая разделяет аргументы, переданные функции.

8. **Наконец, добавьте вторую функцию (строки 13—15 ниже). Итоговый сценарий должен выглядеть так:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4 $('#gallery img').each(function() {
5  var imgFile = $(this).attr('src');
6  var preloadImage = new Image();
7  var imgExt = /(\.\w{3,4}$)/;
8  preloadImage.src=imgFile.replace(imgExt,'_h$1');
9  $(this).hover(
10       function() {
11       $(this).attr('src', preloadImage.src);
```

```
12        },
13        function() {
14              $(this).attr('src', imgFile);
15        }
16 ); // конец hover
17 }); // конец each
18 }); // конец ready
```

Вторая функция просто меняет атрибут `src` на атрибут исходного изображения. В строке 5 путь к изображению, первоначально находящемуся на странице, сохраняется в переменной `imgFile`. В этой функции (строка 14) вы получаете доступ к данному значению вновь, чтобы вернуть `src` в исходное состояние. Сохраните страницу и просмотрите ее в браузере, проведите указателем мыши по каждому из черно-белых изображений, чтобы увидеть, как они становятся цветными.

ПРИМЕЧАНИЕ

Вы можете добиться того же эффекта смены изображения, используя CSS. За подробной информацией обратитесь к статье **kyleschaeffer.com/development/pure-css-image-hover/**. Тем не менее, вам следует разобраться в способе замены одного изображения другим средствами JavaScript. Поскольку CSS предусматривает лишь несколько состояний, вроде `:hover` или `:active`, применение JavaScript позволяет использовать для смены изображений различные события, например, двойной щелчок или нажатие определенных клавиш. Кроме того, вы можете запускать данный эффект, используя не само изображение, а другой элемент. Например, у вас может быть кнопка «Сменить изображения», щелчок по которой будет приводить к замене всех изображений на странице другими.

Фотогалерея с эффектами на практике

Давайте усложним последнее руководство, чтобы создать фотогалерею на одной странице. Вы сможете загружать на страницу крупное изображение при помощи щелчка по его эскизу (рис. 7.3). Дополнительно вы используете пару функций эффектов jQuery, чтобы сделать переход между большими изображениями более зрелищным.

Рис. 7.3. Законченная страница фотогалереи. Щелчок по эскизу заставляет появляться крупное изображение, а рисунок, имеющийся на странице, — постепенно исчезать. Полная версия данного руководства находится в файле *готовый_07_02.html* в каталоге *глава07*

Обзор задачи

Принцип работы галереи достаточно прост — щелчок по эскизу вызывает появление крупного изображения. Однако в данном руководстве рассказано, как добавить некоторые возможности, которые сделают представление более интересным благодаря использованию эффектов проявления для смены больших изображений на странице.

Другая представленная здесь техника — *«ненавязчивый» JavaScript*, то есть пользователь, у которого отключен JavaScript, также будет иметь доступ к крупным версиям фотографий. Чтобы достичь этого, эскиз изображения должен быть заключен в ссылку, указывающую на крупный рисунок (рис. 7.4). Для тех, у кого нет JavaScript, щелчок по ссылке означает уход с данной страницы и следование по ссылке для загрузки более крупного изображения. Это не так зрелищно, поскольку посетитель должен уйти со страницы галереи, чтобы увидеть страницу с крупным изображением, но, самое главное, фотографии будут доступны. Для тех, у кого JavaScript включен, щелчок по ссылке заставляет изображение медленно появляться на странице.

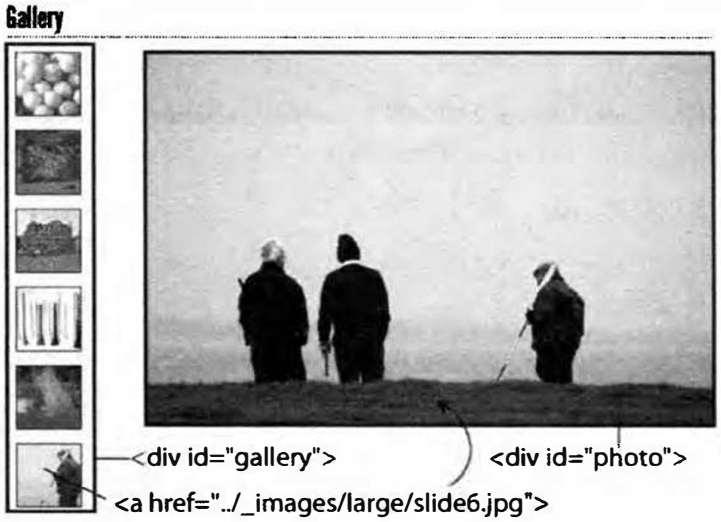

Рис. 7.4. Простейшая структура фотогалереи. Все эскизы изображений заключены в ссылки, указывающие на крупные версии фотографий. При щелчке по каждой из ссылок загружается более крупная версия рисунка, находящаяся внутри элемента `div` с идентификатором `photo`

Действие начинается при щелчке по ссылке, то есть данный сценарий использует событие щелчка по ссылке, чтобы выполнить следующие задачи.

- **Воспрепятствовать стандартному поведению ссылки.** Как правило, щелчок по ссылке означает переход на другую страницу. На данной странице щелчок по ссылке, включающей эскиз, означает покидание страницы и отображение более крупного изображения. Поскольку вы используете JavaScript для показа изображения, можете добавить небольшой код, чтобы воспрепятствовать переходу по ссылке в браузере.
- **Получить значение `href` ссылки**, которая указывает на более крупное изображение. Возвратив значение ее `href`, вы одновременно получите путь к более крупному изображению.
- **Создать новый элемент изображения для вставки на страницу.** Этот элемент изображения будет включать путь из значения `href`.
- **Скрыть старый рисунок и заставить появиться новый.** Текущее изображение медленно исчезает из виду, а крупная картинка — проявляется.

В руководстве есть еще несколько нюансов. В приведенных выше четырех шагах описан основной процесс.

Верстка кода

Данное руководство является расширенной версией предыдущего, но начальная веб-страница немного реорганизована: здесь новые эскизы находятся в левой колонке; на страницу добавлен элемент `div` с идентификатором `photo` (см. рис. 7.4).

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *07_02.html* из каталога *глава07*.**

   В этом файле содержится программный код из предыдущего руководства плюс новый элемент `div` для отображения крупной версии каждого эскиза. Поскольку процесс отображения рисунка инициируется щелчком по ссылке, включающей эскиз, первый шаг — это создание выборки таких ссылок и присвоение каждой из них события щелчка.

2. **Найдите комментарий JavaScript, который гласит «введите новый программный код под этой строкой» и добавьте следующий код:**

   ```
   $('#gallery a').click(function(evt) {

   }); // конец click
   ```

   Селектор `#gallery a` выбирает элементы ссылок внутри элемента с идентификатором `gallery`. `.click` — это функция библиотеки jQuery для добавления обработчика событий (см. раздел «Использование событий: подход jQuery» главы 5, если вам нужно освежить знания о событиях). В коде обработчику события `click` передается анонимная функция (как было сказано в разделе «События мыши» главы 5, функциям, выполняемым в качестве ответа на событие, обязательно передается объект события). В данном случае переменная `evt` сохраняет объект события. Вы воспользуетесь им в следующем шаге, чтобы предотвратить переход по ссылке при щелчке по ней кнопкой мыши.

3. **Между двумя строками кода, добавленными в шаге 2, напечатайте:**

   ```
   evt.preventDefault();
   ```

Как правило, щелчок по ссылке приказывает браузеру загрузить документ, на который она указывает (веб-страницу, графический файл, PDF-документ и т. д.). В данном случае ссылка присутствует для того, чтобы люди, у которых не включен JavaScript, смогли перейти к крупной версии изображения. Чтобы помешать браузеру проследовать по ссылке у тех посетителей, у которых JavaScript включен, запустите функцию объекта события `preventDefault()` (см. раздел «Отмена обычного поведения событий» главы 5).

Далее мы получим атрибут `href` для ссылки.

4. **Нажмите клавишу Enter для создания новой пустой строки и введите:**

```
$('#gallery a').click(function(evt) {
evt.preventDefault();
var imgPath = $(this).attr('href');
}); // конец click
```

Здесь слово `$(this)` относится к элементу, по которому был совершен щелчок, иными словами, к ссылке. Атрибут `href` указывает на страницу или ресурс, к которому ведет ссылка. В данном случае в каждой из ссылок содержится путь к более крупному изображению. Это важная информация, поскольку вы можете использовать ее для добавления элемента изображения, указывающего на его файл. Но прежде чем это сделать, вам необходимо получить ссылку на изображение, которое в данный момент отображено на странице. В конце концов, вам необходимо знать, что конкретно требуется скрыть.

СОВЕТ

Вы увидите, что в каждой из строк кода внутри события `click()` сделан отступ. Это необязательно, но помогает сделать код более удобным для чтения (см. врезку «Пробелы, табуляция и возврат каретки в языке JavaScript» в главе 2). Многие программисты используют два пробела или табуляцию для отступа на каждом уровне.

5. **Нажмите клавишу Enter и введите код:**

```
var oldImage = $('#photo img');
```

Переменная `oldImage` содержит выборку jQuery, в которой находится элемент `img` внутри элемента `div` фотографии (см. рис. 7.4). Пришло время создать элемент для нового изображения.

6. **Снова нажмите клавишу Enter и добавьте в сценарий следующий код:**

```
var newImage = $('<img src="' + imgPath +'">');
```

Здесь происходит следующее: библиотека jQuery позволяет выбрать элемент, находящийся в HTML-коде страницы. Например, часть `$('img')` выбирает все изображения на странице. Дополнительно объект jQuery может добавлять *новый* элемент в документ. Например, фрагмент кода `$('<p>Привет</p>')` создает новый абзац, в котором содержится слово «Привет». В данной строке создается новый элемент `img` и сохраняется в переменной `newImage`.

Поскольку объект jQuery в качестве аргумента должен иметь строку (например, `'<p>Привет</p>'`), в данной строке несколько строк кода соединяются (или комбинируются) в одну. Первая строка (в одиночных кавычках) — `<img src='';` вторая строка сохранена в переменной `imgPath` (созданной в шаге **4**), она является путем к файлу изображения (например, `../_images/large/slide1.jpg`); третья строка (также в одинарных кавычках) — `">`. Взятые вместе они добавляются к HTML-тегу: `<img src="../_images/large/slide1.jpg">`. Когда сценарий передает данную информацию объекту jQuery, например, `$('<img src="../_images/large/slide2.jpg">')`, браузер создает элемент страницы. Он пока еще не отображен, но браузер готов добавить его на страницу в любое время.

7. **Введите код, выделенный полужирным шрифтом в строках 6–8, чтобы итоговый код выглядел так:**

```
1 $('#gallery a').click(function(evt) {
2   evt.preventDefault();
3   var imgPath = $(this).attr('href');
4   var oldImage = $('#photo img');
5   var newImage = $('<img src="' + imgPath +'">');
6   newImage.hide();
7   $('#photo').prepend(newImage);
8   newImage.fadeIn(1000);
9 }); // конец click
```

В строке 6 только что созданное изображение (сохраненное в переменной `newImage`) скрыто с использованием функции `hide()`, описанной в разделе «Основы отображения и сокрытия» главы 6.

Данный шаг необходим, так как если вы добавите только элемент изображения, созданный в строке 5, рисунок сразу же окажется отображен на странице, и зрелищного эффекта постепенного проявления видно не будет. Итак, сначала вы скрываете изображение, а потом добавляете его на страницу в элементе `div` фотографии (строка 7).

Функция `prepend()` (описанная в разделе «Добавление содержимого на страницу» главы 4) добавляет HTML-код в элемент (а именно, в его начало). Теперь на странице в элементе `div` фотографии присутствуют два изображения. На рис. 7.5 показано, как одно изображение может находиться поверх другого. Верхнее изображение невидимо, но в строке 8 функция `fadeIn()` заставляет изображение медленно проявляться в течение 1000 миллисекунд (одной секунды).

Теперь нужно, чтобы исходное изображение исчезло из виду.

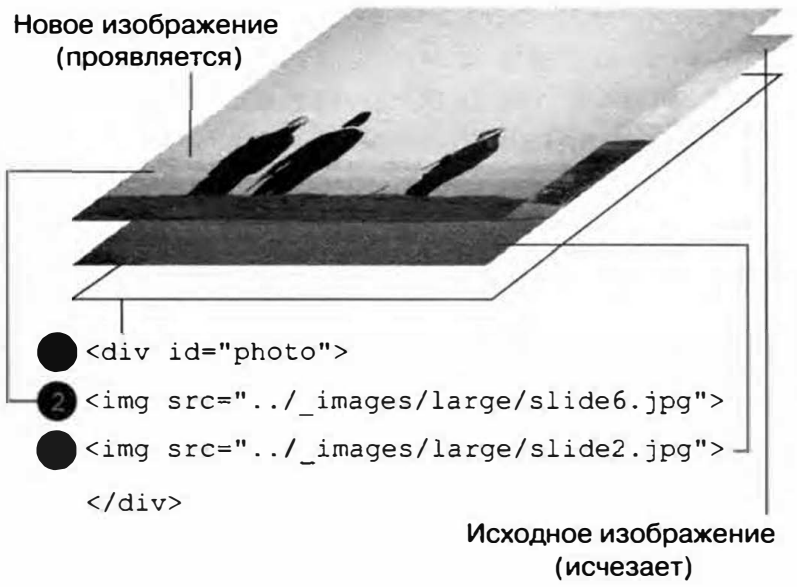

Рис. 7.5. Для достижения эффекта, при котором два фото появляются в одном и том же месте на странице (но чтобы одно фото появлялось, а другое исчезало), вам понадобится творчески использовать код CSS. Абсолютное позиционирование позволяет элементу находиться поверх страницы и даже поверх другого элемента. В данном случае оба элемента абсолютно позиционированы внутри элемента `div`, что позволяет им находиться друг над другом. Таблица стиля, внедренная в раздел заголовка файла *07_02.html*, включена в каталог *глава07* и содержит весь необходимый код CSS (см. стиль *#photo img*). Кроме того, элемент, содержащий фотографии, должен иметь значение `relative` свойства `position`, чтобы позволить фотографиям быть позиционированными относительно конкретного места на странице

8. Нажмите клавишу Enter и добавьте следующие строки кода:

```
oldImage.fadeOut(1000,function(){
$(this).remove();
}); // конец fadeout
```

В шаге 5 вы создали переменную `oldImage` и сохранили в ней ссылку на исходное изображение. Этот рисунок должен быть скрыт, поэтому вы применяете функцию `fadeOut()`. Вы передаете функции два аргумента: первый — длительность эффекта (1000 миллисекунд или одна секунда), а второй — функция обратного вызова (описанная в разделе «Выполнение действия после завершения эффекта» главы 6), которая выполняется *после* завершения эффекта исчезновения и удаляет элемент `img` для данного изображения.

ПРИМЕЧАНИЕ

Функция `remove()` обсуждается в разделе «Замена и удаление выборок» главы 4. Она удаляет элемент из объектной модели документа, то есть стирает HTML-код из памяти браузера, высвобождая ресурсы компьютера. Если этого не сделать, то при каждом щелчке по эскизу станет добавляться новый элемент `img` (см. шаг 7), однако прежний просто будет скрыт, но не удален. В итоге вы получите огромное множество скрытых элементов `img`, по-прежнему присутствующих на веб-странице, что замедлит отклик браузера.

Последний шаг — это загрузка первого изображения. В настоящее время элемент `div`, в котором должна находиться фотография, пуст. Вы можете добавить сюда элемент `img`, и при загрузке страницы станет появляться крупное изображение, например, для первого эскиза. Но зачем беспокоиться — ведь у вас есть JavaScript!

9. Добавьте последнюю строку после функции `click()` (строка 13 ниже), чтобы ваш окончательный код выглядел так:

```
1 $('#gallery a').click(function(evt) {
2  evt.preventDefault();
3  var imgPath = $(this).attr('href');
4  var oldImage = $('#photo img');
5  var newImage = $('<img src="' + imgPath +'">');
6  newImage.hide();
```

```
7  $('#photo').prepend(newImage);
8  newImage.fadeIn(1000);
9  oldImage.fadeOut(1000,function(){
10       $(this).remove();
11 }); // конец fadeout
12 }); // конец click
13 $('#gallery a:first').click();
```

Последняя инструкция состоит из двух частей. Сначала селектор `#gallery a:first` выбирает первую ссылку только в элементе `div` галереи. Далее следует функция `click()`. Мы использовали ее, чтобы присвоить функцию, которая выполняется, когда происходит событие. Однако если вы не передадите функции события никаких аргументов, то библиотека jQuery просто запустит событие, что приведет к запуску заранее определенных обработчиков событий. Итак, в данной строке запускается событие щелчка по первой ссылке, в связи с чем браузер запускает функцию, созданную вами ранее в строках 1—11. То есть данная строка кода заставляет крупную версию первого эскиза появиться при загрузке страницы.

Сохраните страницу и просмотрите ее в браузере. Когда вы наводите указатель мыши на эскизы изображений, они не просто меняют цвет. Щелчок по эскизу заставляет медленно появляться связанную с ним большую картинку (если с вашим кодом возникли проблемы, то обратитесь к файлу *готовый_07_02.html*, в котором содержится рабочая копия сценария).

Управление поведением ссылок

Ссылки позволяют перемещаться по Всемирной паутине. Без мгновенного доступа к информации, который обеспечивается ссылками, ведущими со страницы на страницу и с сайта на сайт, Всемирная паутина не ушла бы далеко. Она бы просто не была *паутиной*. Поскольку ссылки — это один из наиболее обычных и мощных компонентов HTML, само собой разумеется, что существует множество техник JavaScript для улучшения их работы. В данном разделе вы изучите основы использования языка JavaScript для управления ссылками, узнаете, как открывать ссылки в новом окне.

Несомненно, вы уже многое знаете о ссылках. Прежде всего, они — сердце Всемирной паутины, и скромный элемент `a` — это одна из первых частей HTML-кода, изучаемая веб-дизайнером.

Добавление на страницу JavaScript может превратить обычную ссылку в мощный интерактивный шлюз, но только, если вы знаете, как использовать язык JavaScript для контроля над своими ссылками. Для начала вам нужно изучить основы работы со ссылками, после чего мы перейдем к реальным способам управления ими с помощью языка JavaScript.

Выборка ссылок с помощью языка JavaScript

Чтобы выполнить любую операцию со ссылкой на странице, сначала эту ссылку нужно выбрать. Вы можете выбрать все ссылки на странице, только одну ссылку или отдельную группу взаимосвязанных ссылок, например, ссылки, сгруппированные в одной из частей страницы, ссылки, объединенные определенными характеристиками (например, внешние ссылки, ведущие за пределы вашего сайта).

Библиотека jQuery предоставляет большую гибкость при выборе элементов документа. Код `$('a')` создает выборку jQuery, в которую входят все ссылки страницы. Более того, библиотека jQuery обеспечивает более точные выборки, например, вы можете быстро выбрать все ссылки с отдельного участка страницы.

Допустимо выбрать все ссылки, содержащиеся в маркированном списке, имеющем идентификатор `mainNav`, вот так: `$('#mainNav a')`. Вы также можете использовать селекторы атрибутов (см. раздел «Специальные селекторы» главы 4), чтобы выбрать ссылки, чьи значения `href` (пути к файлам, на которые они указывают) соответствуют определенному образцу, например, ссылки, указывающие на другие сайты или на документы PDF (см. раздел «Специальные селекторы» главы 4).

После того как вы выбрали ссылки, вы можете использовать jQuery-функции для работы с этими ссылками. Например, проработать каждую ссылку в цикле с помощью функции `each()` (см. раздел «Работа с каждым элементом выборки» главы 4), присвоить класс этим ссылкам, используя функцию `addClass()` (см. раздел «Классы» главы 4), или добавить к ним функции событий (см. раздел «Использование событий: способ jQuery» главы 5). Вы увидите множество примеров того, что вы можете сделать со ссылками далее в этой главе.

Определение направления ссылки

После того как вы выбрали одну или более ссылок, вы можете поинтересоваться, куда они ведут. Например, в галерее, которую вы создали ранее в этой главе, каждая ссылка указывала на крупное изображение. Находя путь, вы использовали язык JavaScript для отображения этого большого изображения. Иными словами, вы извлекали значение ссылки `href` и использовали его для создания на странице нового элемента `img`. Таким же образом несложно извлечь значение ссылки `href`, ведущей на другую веб-страницу, и вместо того чтобы перейти по ссылке, вы можете отобразить новую веб-страницу поверх текущей веб-страницы.

В каждом случае вам необходим доступ к атрибуту `href`, чего несложно добиться с помощью jQuery-функции `attr()` (см. раздел «Чтение, установка и удаление атрибутов HTML» главы 4). Например, вы присвоили идентификатор ссылке, которая ведет на домашнюю страницу сайта. Вы можете найти путь этой ссылки таким способом:

```
var homePath = $('#homelink').attr('href');
```

Подобная информация пригодится вам во многих случаях. Например, вам нужно добавить полный URL-адрес ссылки, ведущей за пределы вашего сайта, после самого текста ссылки. Допустим, у вас есть ссылка с текстом «Подробнее о короедах», которая указывает на сайт www.barkbeetles.org/. Теперь предположим, что вы хотите изменить текст на «Подробнее о короедах (www.barkbeetles.org)», чтобы, щелкая по ссылке, люди точно знали, куда она ведет.

Вы легко можете сделать это с помощью следующего кода JavaScript:

```
1 $('a[href^="http://"]').each(function() {
2     var href = $(this).attr('href');
3     href = href.replace('http://','');
4     $(this).after(' (' + href + ')');
5 });
```

ПРИМЕЧАНИЕ

Числа слева — просто нумерация строк, которая облегчает чтение кода. Не записывайте эти числа в ваш код!

В строке 1 выбираются все внешние ссылки, затем запускается функция `each()` (см. раздел «Работа с каждым элементом выборки» главы 4),

которая просто применяет функцию к каждой ссылке. В данном случае тело функции содержится в строках 2—4. Строка 2 извлекает атрибут `href` ссылки (например, http://www.barkbeetles.org). Строка 3 необязательна, она лишь упрощает URL-адрес, удаляя часть `http://`, так что в переменной `href` теперь находится следующее: www.barkbeetles.org (вы можете узнать о методе JavaScript `replace()` подробнее в разделе «Замена текста» главы 16). Наконец, в строке 4 содержимое переменной `href` (в круглых скобках) добавляется к окончанию ссылки: (www.barkbeetles.org). Строка 5 закрывает функцию.

Вы можете также создать библиографический список внизу страницы, в котором будут перечислены все ссылки, упоминаемые в статье. Вместо того чтобы добавлять веб-адрес после каждой ссылки, вы можете перечислить адреса внизу страницы в отдельном элементе `div`.

Не переходите по этой ссылке

Добавляя к ссылке событие `click`, вы можете не захотеть, чтобы браузер, как обычно, уходил с текущей страницы и загружал ту страницу, на которую указывает ссылка. Например, в галерее изображений в разделе «Фотогалерея с эффектами на практике» ранее в этой главе, когда вы щелкаете по эскизу, на странице появляется большое изображение. Как правило, щелчок по ссылке эскиза уводит вас с текущей страницы и отображает большое изображение на пустой странице. Однако в нашем случае вместо перехода по ссылке к более крупному изображению мы остаемся на прежней странице, на которую загружается большая картинка.

Есть пара способов предотвратить переход по ссылке — вы можете возвратить значение `false` или использовать jQuery-функцию `preventDefault()` (см. раздел «Отмена обычного поведения событий» главы 5). Например, у вас есть ссылка, приводящая посетителя на страницу авторизации. Чтобы сделать ваш сайт более интерактивным, вы хотите использовать язык JavaScript для отображения формы авторизации при щелчке по ссылке. Иными словами, если в браузере посетителя включен JavaScript, то форма появится на странице; если JavaScript выключен, то щелчок по ссылке «доставит» пользователя на страницу авторизации.

Чтобы достичь данной цели, необходимо выполнить несколько шагов:

1. **Выберите ссылку на страницу авторизации.**

   Обратитесь к первой части этого раздела, если вам нужна подсказка, как это сделать.

2. **Прикрепите событие `click`.**

   Чтобы сделать это, можно использовать jQuery-функцию `click()`, которая принимает в качестве аргумента другую функцию. Во второй функции содержатся действия, производящиеся, когда пользователь щелкает по ссылке. В данном примере необходимо всего два действия.

3. **Покажите форму авторизации.**

   Ее можно скрыть из виду, пока страница загружается, например, с помощью абсолютно позиционированного элемента `div` прямо под ссылкой. Вы можете отобразить форму, используя функцию `show()` или другой эффект отображения, предусмотренный библиотекой jQuery (см. врезку «Основы отображения и сокрытия» в главе 6).

4. **Остановите ссылку!**

   Это самый важный шаг. Если вы не остановите ссылку, браузер просто покинет данную страницу и перейдет на страницу с формой авторизации.

   Рассмотрим, как остановить ссылку с помощью инструкцию возврата. Предположим, ссылка имеет идентификатор `showForm`, а скрытый элемент `div` с формой авторизации — `loginForm`:

```
1 $('#showForm').click(function() {
2   $('#loginForm').fadeIn('slow');
3   return false;
4 });
```

   В строке 1 выполняются этапы 1 и 2, описанные выше. Строка 2 отображает скрытую форму. Строка 3 — это часть, говорящая браузеру: «Стоп! Не переходи по этой ссылке». Вам следует использовать инструкцию `return false;` в качестве последней строки функции, поскольку как только интерпретатор JavaScript встретит инструкцию возврата, он прекратит выполнение функции.

   Можно также использовать jQuery-функцию `preventDefault()` следующим образом:

```
1 $('#showForm').click(function(evt) {
2  $('#loginForm').fadeIn('slow');
3  evt.preventDefault();
4 });
```

Основные детали этого сценария не отличаются от деталей предыдущего. Главное отличие заключается в том, что теперь событие щелчка принимает аргумент `evt`, представляющий само событие (объект события описывается в разделе «Объект события» главы 5). Событие имеет собственный набор свойств и функций — функция `preventDefault()` просто останавливает поведение по умолчанию, связанное с событием щелчка, то есть загрузку новой страницы в ответ на щелчок по ссылке.

Открытие внешних ссылок в новом окне

Потеря посетителей — одна из самых страшных историй, которая может приключиться с сайтами, зависящими от аудитории. Онлайн-журналы, зарабатывающие деньги на рекламе, стараются сделать все, чтобы люди не уходили с их сайта. Электронная коммерция не желает терять потенциальных клиентов, просто позволив им щелкнуть по ссылке и уйти с сайта. Показывая портфолио законченных сайтов, вебдизайнер также не может отпустить потенциального клиента во время просмотра законченных проектов.

В большинстве случаев решение подобных проблем заключается в открытии каждого сайта в новом окне. Получается, что, когда посетитель прекратит просмотр другого сайта и закроет его окно, первоначальный сайт по-прежнему останется на виду. Язык HTML давно предложил метод с использованием атрибута ссылки `target`. Если вы установите для него значение `_blank`, то браузер откроет ссылку в новом окне (или на новой вкладке, если такая возможность предусмотрена).

ПРИМЕЧАНИЕ

Среди экспертов по удобству использования Всемирной паутины ведутся дискуссии относительно того, хороша или плоха идея открытия новых окон. Например, посетите сайт: **www.useit.com/alertbox/990530.html**.

Вручную добавлять фрагмент кода `target="_blank"` к каждой внешней ссылке долго и чревато ошибками. К счастью, язык JavaScript

и библиотека jQuery предоставляют быстрый способ заставить браузеры открывать внешние ссылки (и вообще любые ссылки) в новом окне или на новой вкладке браузера. Этот процесс достаточно прост:

1. **Определите ссылки, которые вы хотите открыть в новом окне.**

   Для этого в данной главе будет использоваться селектор jQuery (см. раздел «Выбор элементов страницы: подход jQuery» главы 4).

2. **Добавьте к ссылке атрибут `target` со значением `_blank`.**

   Вы можете подумать: «Но это же неправильный HTML-код, я не могу так сделать». Ну, во-первых, он не подходит только для строгих версий HTML 4.01 и XHTML 1.0, но подойдет для любых других типов документов, включая новый HTML5. Во-вторых, ваша страница пройдет проверку, так как HTML-валидатор (например, **validator.w3.org/**) проанализирует только код в файле веб-страницы, а не тот HTML, который добавляет JavaScript. И, наконец, любому браузеру понятен атрибут `target`, то есть он откроет ссылку в новом окне независимо от строгих стандартов типов документов.

Используя библиотеку jQuery, вы можете выполнить два предыдущих действия с помощью одной строки кода:

```
$('a[href^="http://"]').attr('target','_blank');
```

Селектор jQuery `$('a[href^="http://"]')` использует селектор атрибутов (см. раздел «Специальные селекторы» главы 4) для нахождения элементов `a`, которые начинаются с `http://` (например, *http://www.yahoo.com*). Селектор идентифицирует все ссылки этого типа, а затем применяет jQuery-функцию `attr()` (см. раздел «Чтение, установка и удаление атрибутов HTML» главы 4), чтобы присвоить атрибуту `target` значение `_blank` для каждой ссылки. И все!

Если вы собираетесь использовать безопасные веб-адреса, начинающиеся с `https://`, то вам следует использовать код:

```
$('a[href^="http://"], a[href^="https://"]')↵
.attr('target','_blank');
```

В данном случае используется групповой селектор для выбора URL-адресов, которые начинаются с http:// или https://.

Наконец, если для ссылок на файлы на вашем сайте используются абсолютные пути, то вам нужен еще один шаг. Например, если адрес ва-

шего сайта *www.your_site.com* и вы ссылаетесь на другие страницы или файлы следующим образом: *http://www.your_site.com/a_page.html*, то предыдущий код открывает в новом окне и такие ссылки. Если вы не хотите открывать в новом окне каждую страницу вашего сайта, то вам нужен такой код:

```
var myURL = location.protocol + '//' + location.↵
hostname; $('a[href^="http://"], a[href^="https://"]').↵
not('[href^="'+myURL+'"]').attr('target','_blank');
```

ПРИМЕЧАНИЕ

Символ ↵ в конце первой строки означает, что следующую строку на самом деле нужно напечатать как часть первой. Но так как длинная строка кода JavaScript не поместится на странице этой книги, пришлось разбить ее на две части.

Данный код сначала определяет URL-адрес вашего сайта и присваивает данный адрес переменной — `myURL`. URL-адрес сайта требует небольшой помощи со стороны объекта браузера `window`. Браузеру известен протокол для доступа к URL — `http://` (для безопасных сайтов — `https://`). Эта информация сохраняется в свойстве объекта `protocol`. Подобным образом, если сайт называется, например, *www.sawmac.com*, то это название сохраняется в свойстве `hostname`. Так JavaScript-код `location.protocol + '//' + location.hostname` генерирует строку, которая выглядит так: *http://www.sawmac.com*. Конечно, в данном случае имя хоста изменяется в зависимости от того, где расположена страница с данным кодом JavaScript. Например, если вы поместите код на страницу, находящуюся на *http://www.your_site.com*, и кто-нибудь просмотрит ее с этого сайта, значением `location.hostname` будет *www.your_site.com*.

Вторая строка кода начинается с селектора jQuery, который находит все ссылки, начинающиеся с `http://`. Затем функция `not()` удаляет все ссылки, начинающиеся с вашего URL-адреса, — в данном случае ссылки, указывающие на сайт *http://www.sawmac.com*. (Функция `not()` полезна при исключении определенных элементов из выборки jQuery. Чтобы изучить ее, посетите сайт: **api.jquery.com/not**.)

Итак, для использования этого кода на странице вы просто ссылаетесь на файл jQuery, добавляете функцию `$(document).ready()`

(см. раздел «Больше концепций для событий jQuery» главы 5), а затем вставляете в нее предыдущий код следующим образом:

```
<script src="js/jquery.min.js"></script>
<script>
$(document).ready(function() {
var myURL = location.protocol + '//' + location↵
.hostname;
$('a[href^="http://"], a[href^="https://"]')↵
.not('[href^="'+myURL+'"]').attr('target','_blank');
});
</script>
```

Другой способ заключается в создании внешнего файла JavaScript (см. раздел «Внешние файлы JavaScript» главы 1). Добавьте в него функцию, запускающую код, который заставляет внешние ссылки открываться в новом окне. Прикрепите этот файл к странице, а затем вызовите функцию.

Например, можно создать файл с именем *open_external.js* с помощью такого кода:

```
function openExt() {
var myURL = location.protocol + '//' + location ↵
.hostname;
$('a[href^="http://"], a[href^="https://"]') ↵
.not('[href^="'+myURL+'"]').attr('target','_blank');
}
```

Затем добавьте следующий код на каждую страницу вашего сайта, к которой вы хотели бы применить данную функцию:

```
<script src="js/jquery.min.js"></script>
<script src="js/open_external.js"></script>
<script>
$(document).ready(function() {
openExt();
// добавьте любой код JavaScript на страницу
});
</script>
```

Польза от использования внешних файлов заключается в том, что если вы применили данную функцию на сотнях страниц вашего сайта, то вы можете легко обновить сценарий, например, чтобы сделать их изящнее.

Позже вы сможете изменить функцию `openExt()` так, чтобы она открывала внешние страницы в рамке на данной странице (в разделе «Открытие страниц в окне на текущей странице» данной главы рассказано, как это сделать). Другими словами, внешний файл *.js* упрощает задачу обеспечения единообразия ваших сценариев по всему сайту.

Создание новых окон

Браузеры позволяют открывать новые окна и настраивать их, например, регулировать высоту, ширину, размещение на экране и даже отображение полос прокрутки, меню или адресной строки. Для этого используется метод `open()`, имеющий следующую структуру:

```
open(URL, name, properties)
```

Метод `open()` принимает три аргумента. Первый — URL-адрес страницы, которая должна появиться в открывшемся окне, то же значение следует использовать для атрибута `href` ссылок (например, *http://www.google.com*, */pages/map.html* или *../../portfolio.html*). Второй аргумент — имя окна, оно может быть любым, на ваше усмотрение, но вам следует помнить о правилах, приведенных в разделе «Создание переменной» главы 2. Наконец, в качестве третьего параметра вы можете передать методу строку, в которой содержатся настройки для нового окна (например, его высота и ширина).

Дополнительно при открытии нового окна обычно создается переменная для хранения ссылки на него. Например, если вы хотите открыть домашнюю страницу сервиса Google в новом окне площадью 200 квадратных пикселов, напишите код:

```
var newWin= open('http://www.google.com/','theWin', ↵
'height=200,width=200');
```

Данный код открывает новое окно и сохраняет ссылку на него в переменной `newWin`. В подразделе «Использование ссылки на окно» далее в главе описывается, как применять такую ссылку для управления открытым окном.

ПРИМЕЧАНИЕ

Имя, которое вы присваиваете новому окну (в рассматриваемом случае `'theWin'`), делает не так много. Однако после присвоения окну имени, если вы попытаетесь открыть новое окно, используя то же имя, то вы не получите нового окна. Вместо этого, веб-страница, запрошенная посредством метода `open()`, загрузится в созданном ранее окне, которому присвоено данное имя.

Свойства окна

Окна браузера включают множество различных компонентов: полосы прокрутки, маркеры масштабирования, панели инструментов и т. д. (рис. 7.6). Кроме того, они имеют определенные ширину, высоту и положение на экране. Большую часть этих свойств можно определить при создании нового окна с помощью строки, которая содержит разделенный запятыми список свойств и их значений, передаваемой в качестве третьего аргумента метода `open()`. Например, для установки ширины и высоты нового окна, а также отображения адресной строки напишите следующее:

```
var winProps = 'width=400,height=300,location=yes';
var newWin = open('about.html','aWin',winProps);
```

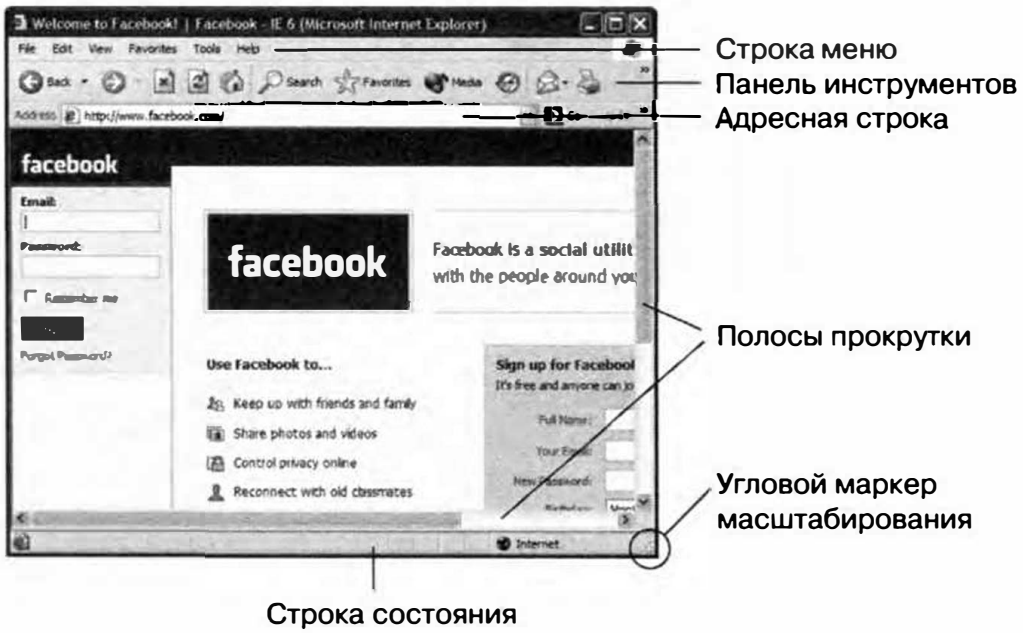

Рис. 7.6. Совокупность свойств окна браузера (полосы прокрутки, панели инструментов и угловые маркеры масштабирования) называется «хромом». Каждый браузер имеет свою собственную версию этих свойств, и веб-разработчики мало что могут сделать в смысле контроля над их работой и внешним видом. Тем не менее при создании нового окна с помощью языка JavaScript вы можете включать и отключать некоторые из данных характеристик

При настройке свойств, определяющих размеры или положение окна, вы пользуетесь значениями в пикселах, тогда как другие свойства регулируются логическими значениями `yes` (включить свойство) или `no` (отключить свойство). Если для такого свойства логическое значение не задано, то браузер отключит его по умолчанию (например, если вы не задали свойство `location`, то браузер скрывает адресную строку, которая обычно появляется в верхней части окна). Только свойства `height`, `width`, `left`, `top` и `toolbar` работают одинаково во всех браузерах. Как указано в следующем списке, некоторые браузеры полностью игнорируют некоторые из этих свойств, так что если вы создаете всплывающие окна с помощью языка JavaScript, то обязательно протестируйте их во всех браузерах.

- **`height`** задает высоту окна в пикселах. Вы не можете устанавливать иные единицы измерения, кроме пикселов. Если значение не задано, браузер исходит из высоты текущего окна.
- **`width`** указывает ширину окна. Здесь также допустимы только значения в пикселах, а если значение не задано, то браузер исходит из ширины текущего окна.
- **`left`** задает позицию в пикселах от левого края монитора.
- **`top`** задает позицию в пикселах от верхнего края монитора.
- **`scrollbars`** задает полосы прокрутки, которые находятся в правой и нижней частях окна браузера, если размер страницы превышает размер окна. Чтобы скрыть полосы прокрутки, установите для этого свойства значение `no`. Вы не можете выбирать, какую из полос скрыть (либо обе отображены, либо обе скрыты), а некоторые браузеры, например, Chrome и Safari не позволяют скрыть полосы прокрутки.
- **`status`** контролирует отображение строки состояния в нижней части окна. Браузеры Firefox и Internet Explorer обычно не позволяют скрывать строку состояния, так что в этих браузерах она видна всегда.
- **`toolbar`** определяет видимость панели инструментов, содержащей кнопки навигации, вкладки и другие элементы, доступные в данном браузере. В программе Safari настройки панели инструментов и адресной строки одинаковы: включение любого из параметров активирует и панель инструментов, и адресную строку.
- **`location`** определяет отображение поля навигации, которое также называют адресной строкой. Это поле содержит URL-адреса

веб-страниц и дает возможность перехода к другой странице путем ввода нового адреса. Браузеры Opera, Internet Explorer и Firefox не позволяют скрывать адресную строку. Это свойство предотвращает неподобающее использование языка JavaScript, например, для открытия нового окна и перехода на другой сайт, похожий на тот, который вы только что покинули. В браузере Safari включение этого свойства активизирует панель инструментов и поле навигации.

- **`menubar`** относится к браузерам, имеющим меню в верхней части окна (например, **Файл** (File) и **Правка** (Edit), обычные для большинства программ). Это свойство применимо только к браузерам, работающим в среде Windows; в операционной системе OS X меню расположено в верхней части экрана, а не в индивидуальном окне. Данное свойство не применимо к браузеру Internet Explorer 7 и более поздним версиям, которые не содержат меню.

ПРИМЕЧАНИЕ

Чтобы увидеть удивительные примеры программ на языке JavaScript, использующих метод `window.open()`, посетите сайты: **experiments.instrum3nt.com/markmahoney/ball/** и **thewildernessdowntown.com/**.

Использование ссылки на окно

Открыв новое окно, вы можете использовать ссылку на него для контроля над ним. Например, вы открываете новое окно с помощью следующей команды:

```
var newWin = open('products.html','theWin','width=300,↵
height=300');
```

Переменная `newWin` содержит ссылку на новое окно. Вы можете применить к этой переменной любой из методов окна браузера для контроля над окном. Например, если вы хотите закрыть его, то воспользуйтесь методом `close()`:

```
newWin.close();
```

Браузеры поддерживают различные методы для объекта окна, среди них можно выделить наиболее часто используемые:

- **`close()`** закрывает указанное окно. Например, команда `close()` закрывает текущее окно. Однако данный метод можно применить

и к ссылке на окно: например, `newWin.close()`. Вы можете назначить выполнение этой инструкции любому событию, например, такому, как щелчок по кнопке, говорящей «Закрой это окно».

ПРИМЕЧАНИЕ

Если вы используете любую из этих команд отдельно, то она применяется к окну, в котором выполняется сценарий. Например, добавив в сценарий инструкцию `close();`, вы можете закрыть содержащее его окно. Однако если вы открыли окно и имеете ссылку на него (например, переменную, которую вы создали при открытии окна, вроде `newWin`), тогда вы можете закрыть его со страницы, которая первоначально создала окно, используя ссылку следующим образом: `newWin.close()`.

- **`blur()`** выводит окно «из фокуса». Это означает, что окно перемещается на задний план под все открытые окна. Это способ скрыть открытое окно, его используют специалисты по веб-рекламе, чтобы создавать рекламные объявления типа «pop under» — окна, скрытые под текущими, таким образом, что закрыв все окна, посетитель страницы обнаружит надоедливую рекламу.
- **`focus()`** противоположен методу `blur()`, помещает окно поверх всех других окон.
- **`moveBy()`** позволяет перемещать окно на заданное число пикселов вправо и вниз. Вы должны передать методу два аргумента. Первый указывает число пикселов, на которое окно следует передвинуть вправо, а второй — число пикселов, на которое окно следует передвинуть вниз. Например, код `newWin.moveBy(200,300);` перемещает окно, на которое ссылается переменная `newWin`, на 200 пикселов вправо и на 300 пикселов вниз. Вы также можете использовать отрицательные значения, таким образом, для передвижения окна на 100 пикселов вверх и на 300 пикселов влево, вы можете использовать следующий код:

```
newWin.moveBy(-100,-300);
```

- **`moveTo()`** позволяет перемещать окно в заданное место на экране. Эта команда эквивалентна использованию свойств `left` и `top` (см. предыдущий подраздел) при открытии нового окна. Например, чтобы поместить окно строго в левом верхнем углу монитора, напишите следующий код: `moveTo(0,0);`.

ПРИМЕЧАНИЕ

Чтобы увидеть многие из этих методов в действии, посетите вебсайт: **www.allisnotlo.st**.

- **`resizeBy()`** изменяет ширину и высоту окна. Первый из двух аргументов определяет, на сколько пикселов должна увеличиться ширина окна, а второй — на сколько пикселов должна стать больше высота. Например, код `resizeBy(100,200);` делает текущее окно на 100 пикселов выше и на 200 пикселов шире. Для уменьшения размеров окна используйте отрицательные значения.
- **`resizeTo()`** изменяет размеры окна на заданные. Например, код `resizeTo(200,400);` задает для текущего окна ширину 200 пикселов и высоту 400 пикселов.
- **`scrollBy()`** прокручивает документ внутри окна на заданное количество пикселов вправо и вниз. Например, код `scrollBy(100,200);` прокручивает документ вниз на 200 пикселов и вправо на 100 пикселов. При невозможности прокрутки документа (когда он целиком помещается внутри окна или достигнут его конец) функция недоступна.
- **`scrollTo()`** прокручивает документ внутри окна на заданную позицию по отношению к левому и верхнему краям. Например, код `scrollTo(100,200);` прокручивает текущий документ на 200 пикселов от верхнего края окна и на 100 пикселов от левого. При невозможности прокрутки документа (когда он целиком помещается внутри окна или достигнут его конец) функция недоступна.

ПРИМЕЧАНИЕ

Плагин jQuery ScrollTo обеспечивает простой способ контроля над прокручиванием документа с помощью языка JavaScript. Информацию о нем можно найти по адресу: **github.com/flesler/jquery.scrollTo**.

События, открывающие новое окно

За время короткой истории Всемирной паутины всплывающие (popup) окна завоевали дурную репутацию. К сожалению, многие сайты злоупотребляют методом `open()` для внезапного открытия окон перед глазами ничего не подозревающих посетителей. В настоящее время

большинство браузеров снабжены возможностью блокировки всплывающих окон, так что, даже если вы добавили код JavaScript для открытия нового окна при загрузке или закрытии страницы, браузер не позволит ему появиться. Посетитель либо увидит сообщение о том, что всплывающее окно заблокировано, либо вообще не узнает об этом событии.

На деле многие браузеры не позволяют открывать новое окно в ответ на большинство событий вроде передвижения указателя мыши или нажатия клавиш. Единственный надежный способ применить команды JavaScript для открытия окон состоит в запуске действия после щелчка по ссылке или отправки формы. Для этого вы добавляете событие `click` к любому элементу HTML-кода (не обязательно к ссылке) и открываете новое окно. Например, вам нужно открыть некоторые ссылки на странице в новом окне площадью 300 квадратных пикселов с полосами прокрутки и маркерами масштабирования, но без хрома браузера вроде панели инструментов. Вы можете добавить имя класса — например, `popup`, к каждой из ссылок, и затем добавить на страницу следующий код jQuery:

```
$('.popup').click(function() {
var winProps='height=300,width=300,resizable=yes, ↵
scrollbars=yes';
var newWin=open($(this).attr('href'),'aWin',winProps);
}
```

Знакомство с плагинами jQuery

С помощью JavaScript и jQuery вы уже способны обеспечивать интерактивность веб-страниц, используя всего несколько строк кода. Однако вам может захотеться создать более интересные и сложные дополнения пользовательского интерфейса. Например, как насчет слайдера? Это распространенный элемент домашней страницы, который позволяет один за другим отображать несколько слайдов на большой площади страницы, как показано на рис. 7.7. Слайды могут быть фотографиями, видео или просто элементами `div`, включающими HTML-контент. Посетители могут перемещаться между слайдами, щелкая по кнопкам, а с помощью щелчка по слайду могут перейти на другую страницу сайта.

Программирование слайдеров (также называемых «каруселями») может быть сложным и трудоемким. К счастью, одно из преимуществ jQuery состоит в том, что эта библиотека предусматривает большое ко-

личество плагинов. Плагин jQuery — это файл JavaScript, который работает совместно с jQuery. Плагины позволяют добавлять на веб-страницу слайдеры, помогают проверить введенные в форму данные и многое другое. jQuery UI, о котором вы узнаете в части III книги, является большим и сложным плагином, позволяющим добавлять элементы управления, необходимые для создания крупномасштабных веб-приложений. Тем не менее большинство плагинов представляет собой небольшие файлы JavaScript, которые очень хорошо справляются с одной конкретной задачей.

Слайдер WOW Slider, изображенный на рис. 7.7, — это коммерческий плагин (то есть за его использование на коммерческом веб-сайте необходимо платить). Однако многие плагины распространяются бесплатно и имеют открытый исходный код, это значит, что вы можете использовать их, не платя за это, и, что еще лучше, открыть файлы JavaScript и посмотреть, как эти плагины написаны. Изучение кода, созданного другими разработчиками, является отличным способом, позволяющим научиться программировать плагины, кроме того, это дает возможность усовершенствовать плагин путем изменения его кода.

Рис. 7.7. Wow Slider (**wowslider.com**) является полноценным плагином jQuery, который позволяет очень легко создавать анимированные слайдеры. Он предусматривает многочисленные варианты отображения слайдов и может анимировать переход между слайдами разнообразными впечатляющими способами

В этой книге вы узнаете о нескольких полезных плагинах, однако во Всемирной паутине их буквально тысячи. Начнем с того, что веб-сайт jQuery содержит каталог плагинов: **plugins.jquery.com**. Вы можете найти еще больше плагинов, посетив такие сайты, как **Sitepoint.com** или **WebDesignerDepot.com**, или просто осуществив поиск в Интернете. Например, если вы введете в поисковую строку что-то вроде «Google Maps jQuery плагин», то получите список плагинов, которые могут помочь вам добавить на сайт Google Map.

На что обратить внимание в плагине jQuery?

Перед загрузкой и использованием первого попавшегося плагина вы должны подумать о том, действительно ли это то, что вам нужно. Вы столкнетесь с большим количеством классных плагинов, позволяющих обеспечить интересные эффекты, однако легко соблазниться плагином, который здорово выглядит и делает что-то по-настоящему удивительное. Проблема в том, что вы легко можете добавить на сайт десятки плагинов, вынуждая посетителей тоже их загружать. Это будет снижать скорость загрузки вашего сайта и может также привести к замедлению работы веб-браузеров посетителей, которым для отображения ваших веб-страниц придется обрабатывать несколько программ JavaScript.

Кроме того, при добавлении плагина вы становитесь зависимым от его разработчика. Если при выпуске плагина в нем содержалась необнаруженная ошибка, то вам или разработчику придется ее исправить, иначе вам потребуется удалить плагин с вашего сайта.

Таким образом, вам следует ограничить количество плагинов только теми, которые вам действительно нужны. Кроме того, вам нужно выбирать плагины, имеющие хороший послужной список. Если плагин был размещен на веб-сайте только сегодня, и это версия 0.0.0.1, подумайте дважды, прежде чем его использовать. Вы можете определить «зрелость» плагина по номеру версии. То есть версия 0.0.1 означает, что плагин является совершенно новым, в то время как версия 4.1.10 говорит о том, что он пережил несколько доработок.

Точно так же вам следует посмотреть на дату выхода плагина. Каталог плагинов jQuery предоставляет эту информацию для каждого плагина. Например, плагин Chosen (рис. 7.8) впервые появился на сайте jQuery 5 марта 2013 года. На таких сайтах, как Github (**github.com**), на котором содержится множество проектов с открытым исходным кодом, в том числе

тысячи плагинов jQuery, также указана дата создания проекта. Также важно знать дату последнего обновления плагина. Плагин, имеющий длинную историю обновлений, исправлений и добавления новых функций, вероятно, просуществует еще длительное время и продолжит обновляться при возникновении проблем. Если плагин обновлялся 4 года назад, держитесь от него подальше, поскольку он может не работать с последней версией jQuery, и, конечно, он не был протестирован в новейших браузерах.

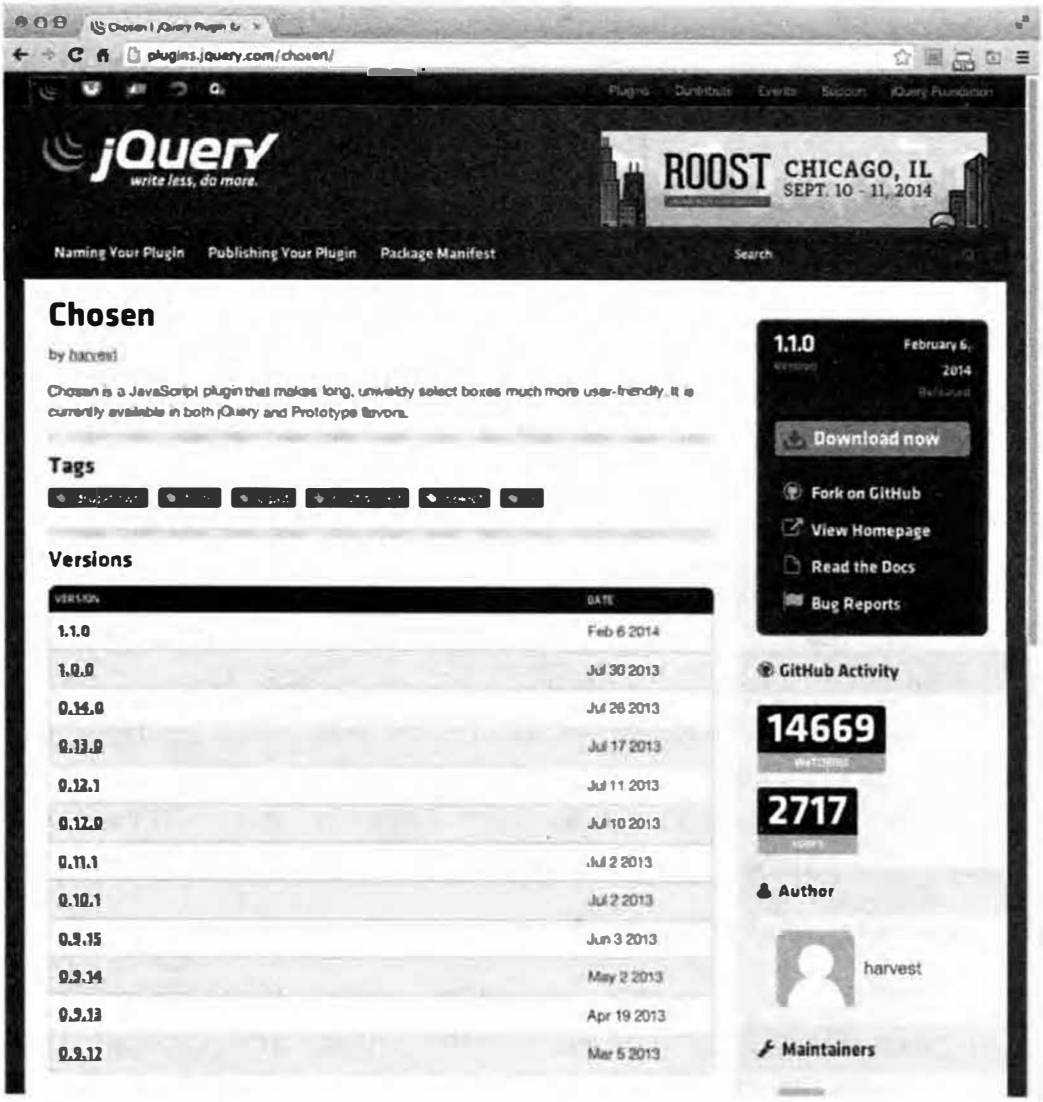

Рис. 7.8. На сайте **jQuery.com** содержится список бесплатных плагинов (**plugins.jquery.com**). Каждый плагин имеет свою собственную страницу, на которой показана история его развития (даты появления, пересмотров и обновлений). Вы также можете увидеть, сколько людей (так называемых наблюдателей) следят за ним. Чем больше у плагина наблюдателей, тем больше его популярность. Например, у приведенного на данном рисунке плагина 14 669 наблюдателей!

Вы также можете поискать плагины в поисковых системах Google или Yandex и прочитать отзывы или обсуждения на форумах разработ-

чиков. Если среди результатов поиска вы увидите много таких фраз, как «Я не могу заставить плагин XYZ работать в WordPress» или «Помогите! Из-за плагина XYZ мой сайт не работает», то у вас также могут возникнуть проблемы.

Основы работы с плагинами jQuery

Хотя плагины jQuery различаются по сложности и качеству, они, как правило, имеют аналогичные требования, касающиеся установки. При загрузке плагина вы получаете сам файл плагина с расширением *.js*. Кроме того, часто с этим файлом поставляется CSS-файл, который отвечает за визуальное форматирование HTML-кода, добавляемого плагином на страницу. Например, полезный плагин Datepicker, предусмотренный в jQuery UI (см. раздел «Стильный способ выбора даты» главы 10), который добавляет всплывающий календарь в виде элемента формы, что облегчает посетителю процесс выбора даты, форматируется сопровождающим файлом CSS. Кроме того, плагины часто включают графические файлы, которые добавляют визуальные компоненты в HTML-код плагина.

Основной процесс таков:

1. **Загрузите файлы.**

   Вы можете загрузить файлы с сайта, посвященного плагину, или очень часто с ресурса Github. Загрузка часто включает в себя дополнительные файлы, которые были использованы при разработке плагина, например, демонстрационные страницы, специальные тесты и дополнительные файлы. В данный момент вам нужен только файл *.js*, файл *.css* и любые изображения, используемые в плагине.

2. **Переместите необходимые файлы в папку вашего сайта.**

   Вы можете сделать это несколькими способами. Вы можете переместить файл плагина *.js* в тот же самый каталог, где содержатся другие файлы JavaScript, а файл *.css* переместить в каталог с другими файлами CSS. Тем не менее часто лучше хранить графические файлы для плагина с файлом *.css*, поскольку обычно файл CSS плагина ссылается на них.

   Другой подход позволяет намного легче отслеживать (и удалять) плагины. Он заключается в том, чтобы поместить все компоненты плагина в папку с его названием, а затем поместить эту папку в каталог, содержащий файлы JavaScript, или просто в папку с названием *plugins* в корневом каталоге вашего сайта. Например, допустим,

вы загрузили плагин Super Plugin. Он может включать в себя файл с именем *jquery.super-plugin.min.js*, CSS-файл с именем *super-plugin.css* и папку с изображениями *images*. Вы можете создать папку с именем *super_plugin* и поместить в нее все эти файлы. Затем поместите папку *super_plugin* на свой сайт (например, в папку *scripts* или в специальную папку под названием *plugins*).

3. **Прикрепите CSS-файл плагина к веб-странице.**

   Этот код помещается в разделе заголовка веб-страницы непосредственно под ссылкой на другие таблицы стилей, которые вы используете:

```
<link href="css/site.css" rel="stylesheet">

<link href="plugins/super_plugin/super-plugin.css" ↵
rel="stylesheet">
```

4. **Создайте ссылку на JavaScript-файл плагина.**

   Плагины jQuery зависят от этой библиотеки, так что сначала вам необходимо создать ссылку на файл jQuery, а затем — на файлы плагинов:

```
<script src="js/jquery.min.js"></script>

<script src="plugins/super_plugin/jquery. ↵
super-plugin.min.js"></script>
```

5. **Измените HTML-код вашей страницы.**

   Да, эта инструкция немного расплывчата. Тем не менее каждый плагин имеет свои особенности и правила использования. В большинстве случаев для того, чтобы плагин заработал, вы должны добраться до HTML-кода страницы и добавить некоторое содержимое. Это может быть простое добавление нескольких классов к уже существующим элементам. Эти классы служат своеобразными «маркерами» плагина, определяющими фрагменты HTML-кода, которые плагин должен выбрать и изменить.

   В других случаях вам может потребоваться добавить специально структурированный HTML-код. Например, виджет Accordion, предусмотренный jQuery UI (см. раздел «Экономия пространства с помощью аккордеонов» главы 9), требует, чтобы вы создали особую структуру: HTML-элемент, который выступает в качестве «контейнера» для аккордеона, а также ряд HTML-элементов `h`, за которыми следуют элементы `div`:

```
<div id="accordion">
<h3>Первый заголовок</h3>
<div>Первая панель с содержимым</div>
<h3>Второй заголовок</h3>
<div>Вторая панель с содержимым</div>
</div>
```

Плагин будет некоторым образом изменять представление HTML-кода: отображать его в виде панели-аккордеона с областями, которые посетитель может скрывать и отображать; создавать набор движущихся слайдеров (как в случае с Wow Slider на рис. 7.6); или отображать всплывающие подсказки.

6. **Вызов функции плагина.**

   Точный метод варьируется от плагина к плагину. Тем не менее многие плагины работают аналогичным образом: создают выборку jQuery, а затем вызывают функцию плагина. Выборка jQuery часто представляет собой селектор идентификатора или класса, который вы добавили на свою HTML-страницу. Например, в случае с jQuery UI Accordion все сводится к выбору HTML-элемента, который содержит элементы аккордеона, и к вызову функции `accordion()`:

   ```
   $('#accordion').accordion();
   ```

   Да, часто можно обойтись одной строкой кода. Большинство плагинов также позволяет изменять принцип своей работы путем передачи дополнительных инструкций, часто в форме литерала объекта JavaScript. Вы можете настраивать виджет jQuery UI Datepicker различными способами, просто передавая ему различную информацию. Например, чтобы отобразить бок о бок три календаря на разные месяцы, а также панель под ними, содержащую кнопку для выбора текущей даты, вам пришлось бы предоставить плагину литерал объекта следующим образом:

   ```
   $('#date').datepicker({
   numberOfMonths: 3,
   showButtonPanel: true
   });
   ```

Каждый плагин имеет свои особенности, однако большинство из них подчиняется общей схеме, описанной выше. В следующем разделе вы получите некоторый опыт работы с конкретным плагином.

Создание отзывчивого меню навигации

По мере расширения сайта становится все труднее обеспечить доступ к каждому его разделу и при этом избежать переполнения страницы ссылками. Для облегчения навигации в этих условиях многие веб-дизайнеры используют выпадающие меню, позволяющие скрывать ссылки, пока они не востребованы (рис. 7.9). Решения подобной проблемы на базе CSS не всегда подходят. Во-первых, меню, созданное только на основе CSS, капризно: если вы уберете с него указатель мыши, хоть на мгновение, меню исчезнет. Кроме того, код CSS часто довольно сложен, и если вы не являетесь экспертом, то вам лучше использовать JavaScript.

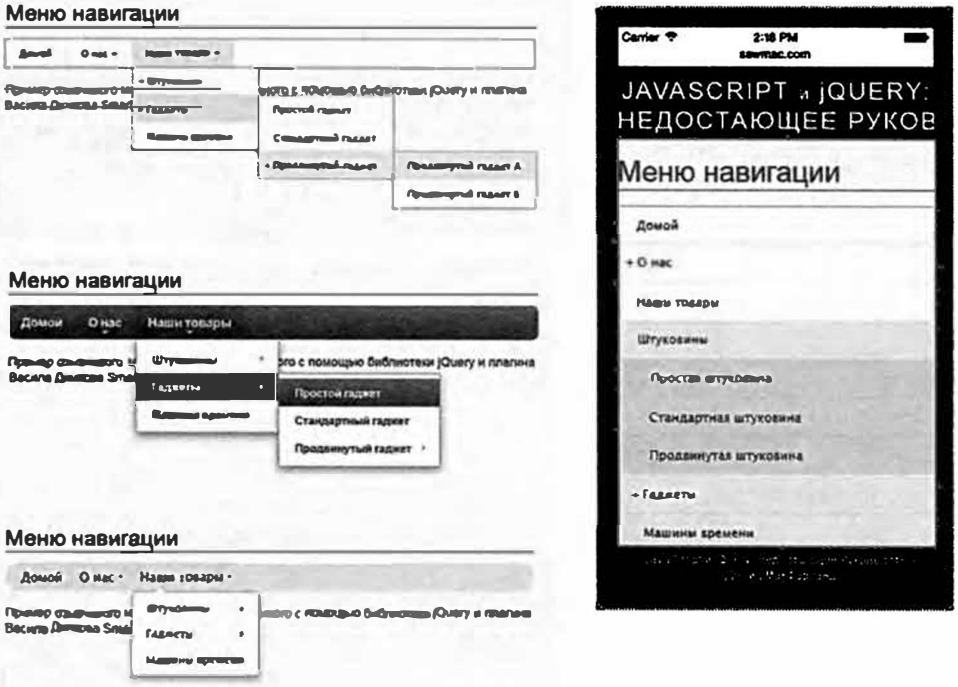

Рис. 7.9. Навигация по сайту, имеющему множество страниц и разделов, может оказаться затруднительной. Панель навигации с раскрывающимися меню является элегантным способом упростить представление ссылок (слева). Плагин jQuery SmartMenus (**www.smartmenus.org**) позволяет вам легко создать множество навигационных элементов и включить несколько уже созданных визуальных тем: простую (вверху слева), синюю (в середине слева) и чистую (внизу слева). При просмотре через телефон это меню превращается в простой для навигации список ссылок (справа)

К счастью, с помощью языка JavaScript вы можете создать систему анимированных меню, работающую во всех браузерах. В этом разделе вы будете использовать плагин jQuery SmartMenus, упрощающий про-

цесс создания выпадающего меню. Кроме того, этот плагин создает отзывчивую навигационную панель, которая автоматически подстраивается под размер окна браузера и уменьшается при просмотре страницы через телефон (рис. 7.9 справа).

Она базируется в большей степени на методах HTML и CSS, чем на других приемах JavaScript, которые вы до сих пор изучали. HTML понадобится вам для создания вложенного набора ссылок, а CSS — для форматирования этих ссылок, чтобы они выглядели как панель навигации, а также для отображения и сокрытия подменю. Затем вы воспользуетесь возможностями языка JavaScript для анимации изменения вида меню при наведении указателя мыши на кнопки панели навигации.

HTML-код

HTML-код для меню навигации — это простой маркированный список, созданный посредством элемента `ul`. Каждый из элементов `li` верхнего уровня представляет главные кнопки на панели навигации. Для создания подменю добавьте вложенный элемент `ul` в элемент `li`, которому принадлежат меню. К примеру, HTML-код для меню, приведенного на рис. 7.9, выглядит так:

```
<ul id="navigation">
<li><a href="#">Домой</a></li>
<li><a href="#">О нас</a>
<ul>
<li><a href="#">Наша история</a></li>
<li><a href="#">Как нас найти</a></li>
<li><a href="#">Часы работы</a></li>
</ul>
</li>
<li><a href="#">Наши товары</a>
<ul>
<li><a href="#">Штуковины</a>
<ul>
<li><a href="#">Простая штуковина</a></li>
<li><a href="#">Стандартная штуковина</a></li>
<li><a href="#">Продвинутая штуковина</a></li>
```

```
</ul>
</li>
<li><a href="#">Гаджеты</a>
<ul>
<li><a href="#">Простой гаджет</a></li>
<li><a href="#">Стандартный гаджет</a></li>
<li><a href="#">Продвинутый гаджет</a>
<ul>
<li><a href="#">Продвинутый гаджет А</a></li>
<li><a href="#">Продвинутый гаджет Б</a></li>
</ul>
</li>
</ul>
</li>
<li><a href="#">Машины времени</a></li>
</ul>
</li>
</ul>
```

ПРИМЕЧАНИЕ

Чтобы данный пример выглядел проще, в HTML-код помещен знак # вместо конкретного URL-адреса для свойства `href` каждого элемента `a` — `<a href="#">`, например. В настоящей панели навигации каждый элемент будет указывать на реальную веб-страницу следующим образом: `<a href="products/gadgets/basic.html">`.

Тремя главными кнопками навигации являются «Домой», «Наши товары» и «О нас». «О нас» — это меню, представленное вложенным списком, включающим разделы «Наша история», «Как нас найти» и «Часы работы». Кнопка «Наши товары» содержит меню с вариантами «Штуковины», «Гаджеты» и «Машины времени». Кнопки «Штуковины» и «Гаджеты» предусматривают свои собственные меню (два других вложенных списка), а под вариантом «Продвинутый гаджет» есть еще два пункта (еще один вложенный список). Вложенный список представляет список другого уровня. Данный HTML-код можно представить так:

- Домой
- О нас
- Наша история
- Как нас найти
- Часы работы
- Наши товары
- Штуковины
- Простая штуковина
- Стандартная штуковина
- Продвинутая штуковина
- Гаджеты
- Простой гаджет
- Стандартный гаджет
- Продвинутый гаджет
- Продвинутый гаджет А
- Продвинутый гаджет Б
- Машины времени

Помните, что вложенный список в действительности находится внутри элемента `li` (родительского списка). Например, элемент `ul`, содержащий элементы «Штуковины», «Гаджеты» и «Машины времени», находится внутри элемента `li` элемента списка «Наши товары» (если вам требуется освежить знания о создании списков HTML, то посетите сайт: **www.htmldog.com/guides/htmlbeginner/lists/**).

ПРИМЕЧАНИЕ

Убедитесь, что ссылки верхнего уровня (в нашем примере «Домой», «Наши товары» и «О нас») указывают на страницу, ссылающуюся на подстраницы в своем разделе (например, ссылки «Наша история», «Как нас найти» и «Часы работы» должны быть связаны со страницей «О нас»). В этом случае, даже если в браузере не задействован JavaScript, он все же сможет получить доступ к ссылкам в подменю.

Каскадная таблица стилей

Плагин jQuery SmartMenus, созданный Василом Динковым (**www.smartmenus.org**), делает основную часть работы по размещению эле-

ментов списка, так что вам не придется волноваться о создании каскадной таблицы стилей, необходимой для создания кнопок и выпадающих меню. Главный CSS-файл под названием *sm-core.css* отвечает за размещение навигационных кнопок.

Затем вы сможете создать собственные стили для настройки внешнего вида кнопок или использовать одну из предоставленных тем. Каждая тема поставляется в собственном CSS-файле: *sm-simple.css*, *sm-clean.css*, *sm-blue.css*. Вы узнаете о том, как использовать тему и настраивать внешний вид меню в следующем руководстве.

JavaScript

Основная концепция использования языка JavaScript для управления видом меню проста. Указатель мыши наводится на элемент списка, и если тот имеет вложенный список (раскрывающееся меню), то этот список отображается; затем указатель мыши смещается со списка, и все вложенные списки скрываются.

Существует ряд тонкостей, несколько усложняющих эту основную идею. В частности, раскрывающиеся меню, исчезающие тотчас, как только курсор оказывается вне родительского списка, требуют точной техники управления мышью.

При навигации по раскрывающимся меню легко сдвинуть курсор с элемента списка. Если меню неожиданно исчезает, посетитель вынужден возвращать указатель мыши к исходному элементу, чтобы вновь открыть меню. И если существует несколько уровней, то очень легко пропустить цель и потерять нужное меню.

Для решения этой проблемы в большинство сценариев добавляется свойство таймера с целью отсрочить исчезновение меню. Это свойство позволяет учесть не совсем точную технику управления мышью и сделать так, чтобы раскрывающиеся меню не казались столь «хрупкими».

Руководство

Теперь, когда вы поняли принципы создания меню навигации, посмотрим, как это происходит на практике. В этом руководстве вы с помощью CSS и JavaScript преобразуете простой HTML-список меню, показанный в разделе «HTML-код» ранее в главе, в панель навигации.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *07_03.html* в каталоге *глава07*.**

   Этот файл содержит маркированный список ссылок, подлежащий преобразованию в панель навигации. Чтобы увидеть, как он выглядит без применения кода JavaScript, откройте файл в браузере. Первый шаг при создании меню заключается в присоединении предоставленного CSS-файла.

2. **Щелкните мышью в пустой строке после кода `<link href="../_css/site.css" rel="stylesheet">` и введите:**

   ```
   <link href="smartmenus/sm-core-css.css" ↵
   rel="stylesheet" > <link href="smartmenus/ ↵
   sm-simple.css" rel="stylesheet" >
   ```

   Первый элемент `link` добавляет основную таблицу стилей. Эта таблица нужна всегда, поскольку она обеспечивает базовое форматирование вне зависимости от цветовой схемы или шрифтов, которые вы хотите использовать. Этот файл содержит стили для «простой» схемы, изображенной на рис. 7.9.

   Теперь вам предстоит внести в HTML-код небольшие изменения.

3. **Найдите элемент `ul`, который определяет начало навигационной панели (непосредственно под элементом `h1`), и добавьте атрибут класса с двумя значениями:**

   ```
   <ul class="sm sm-simple">
   ```

   Имя класса `sm` необходимо для всех меню SmartMenus. Этот плагин требует наличия данного класса, поскольку он помогает ему творить его «волшебство» и применять к меню стили CSS. Второе имя класса `sm-simple` указывает на то, какую тему вы используете. Если в шаге 2 вы присоединили один из других CSS-файлов (*sm-clean.css* или *sm-blue.css*), то вы будете использовать соответствующее имя класса. Другими словами, замените фрагмент `sm-simple` фрагментом `sm-clean` или `sm-blue`.

   Теперь пришло время добавить код JavaScript.

4. **Щелкните мышью в пустой строке после кода `<script src="../_js/jquery.min.js"></script>` и введите:**

```
<script src="smartmenus/jquery.smartmenus.min.js"> ↵
</script>
```

Этот код загружает плагин SmartMenus. Помните о том, что, поскольку для работы этому плагину необходима библиотека jQuery, вы должны поместить эту строку после элемента `script`, указывающего на jQuery. Кроме того, обратите внимание на то, что файл JavaScript и два CSS-файла в предыдущем шаге находятся в папке под названием *smartmenus*. Как говорилось ранее в главе, хорошей идеей является хранение всех файлов, необходимых плагину, в отдельной папке с одноименным названием. В этом случае вы сможете легко найти все файлы плагина при необходимости обновления (например, при выходе новой версии) или удаления (если этот плагин вам надоел или перестал работать).

Теперь вы добавите программный код. Рабочий файл уже содержит набор элементов `script` и функцию `$(document).ready()`.

5. **Щелкните в пустой строке внутри функции `$(document).ready()` и добавьте код, выделенный полужирным шрифтом:**

```
$(document).ready(function() {
$('.sm').smartmenus();
}); // окончание функции ready
```

Для активации меню сначала используйте библиотеку jQuery, чтобы выбрать элемент `ul`, применяемый для главной панели навигации, — в данном примере, этому тегу присвоен класс `sm`, так что код `$('.sm')` выбирает этот элемент. Функция `.smartmenus()` применяет к меню плагин jQuery SmartMenu. Да, все действительно так просто!

6. **Сохраните файл и просмотрите его в браузере. Наведите указатель мыши на пункты меню и максимально уменьшите ширину окна браузера.**

Вы заметите, что теперь выпадающие меню появляются при нажатии основных кнопок, а вылетающие кнопки находятся в подменю. Довольно круто.

Однако вы можете сделать еще больше. Передавая плагину литерал объекта (см. раздел «Одновременное изменение нескольких свойств

CSS» главы 4), содержащий различные варианты, вы способны контролировать работу плагина. Например, в данный момент подменю проявляется при наведении на кнопку указателя мыши, а при его смещении оно исчезает из вида. Однако с помощью определенного кода вы можете заставить подменю выдвигаться и задвигаться. Этот код несколько сложен, поэтому добавляйте его по одному фрагменту за раз.

7. **Отредактируйте код, который вы только что написали, добавив фигурные скобки (выделены полужирным шрифтом):**

```
$(document).ready(function() {
$('.sm').smartmenus({ });
}); // окончание функции ready
```

Фигурные скобки передают функции литерал объекта. Как вы помните, литерал объекта — это просто способ отправки функции одной или нескольких пар имя/значение. Например, вот простой объект, содержащий имя автора:

```
{ name : 'Dave' }
```

В данном примере `name` — это имя (или ключ), а `'Dave'` — это значение. Пришло время освободить место для значений, которые вы собираетесь добавить.

8. **Добавьте пустую строку между фигурными скобками литерала объекта, а затем добавьте комментарий в конце:**

```
$(document).ready(function() {
$('.sm').smartmenus({
}); // окончание smartmenus
}); // окончание функции ready
```

Комментарий позволяет вам определить, где заканчивается меню. Теперь добавьте первую пару имя/значение.

9. **Добавьте код, содержащийся в строках 3–5:**

```
$(document).ready(function() {
$('.sm').smartmenus({
showFunction: function($ul, complete) {
$ul.slideDown(250, complete);
```

```
}
}); // окончание smartmenus
}); // окончание функции ready
```

В данном случае именем (или ключом) является значение `showFunction`. Плагин SmartMenus имеет множество заранее запрограммированных параметров, один из них — `showFunction`. Если это имя передается плагину, то функция, передающаяся в качестве значения, используется для анимации внешнего вида подменю.

В данном фрагменте происходит очень многое, вам должна быть знакома функция `slideDown()` в строке 4. (Впервые вы прочитали об анимационных эффектах jQuery в разделе «Скользящие элементы» главы 6.) Вы можете заменить функцию `slideDown()` любым анимационным эффектом jQuery, например, `.hide()` или `.fadeIn()`, или создать собственный анимационный эффект, используя функцию `.animate()`, описанную в разделе «Анимация» главы 6.

Числовое значение соответствует продолжительности анимационного эффекта, в данном случае — 250 миллисекунд. Вы можете уменьшить это значение, чтобы скольжение происходило быстрее, или увеличить значение, чтобы его замедлить.

Теперь пришло время заставить подменю задвигаться вверх при смещении с него указателя мыши.

10. Отредактируйте код, чтобы он выглядел так (дополнения выделены полужирным шрифтом):

```
$(document).ready(function() {
$('.sm').smartmenus({
showFunction: function($ul, complete) {
$ul.slideDown(250, complete);
},
hideFunction: function($ul, complete) {
$ul.slideUp(250, complete);
}
}); // окончание smartmenus
}); // окончание функции ready
```

Не забудьте поставить запятую после фигурной скобки в строке 5. Запятая используется для отделения пар имя/значение в литерале объекта. В данном случае вы отправляете плагину SmartMenus второй вариант: `hideFunction`.

ПРИМЕЧАНИЕ

Плагин SmartMenus предусматривает много способов управления принципом своей работы. Для получения более подробной информации обратитесь на сайт: **www.smartmenus.org/docs/**.

11. Сохраните страницу и просмотрите ее в браузере.

Теперь у вас должна быть полностью функциональная навигационная панель. Обратитесь к таблице стилей в файле, чтобы разобраться с форматированием кнопок, и попробуйте настроить различные свойства CSS для изменения вида меню. Полная версия данного руководства находится в файле *готовый_07_03.html* в каталоге *глава07*.

Чтобы познакомиться с другими темами, замените ссылку на таблицу стилей `sm-simple.css`, а также класс `sm-simple` элемента `ul` навигационной панели.

Например, для использования «синей» темы следует заменить ссылку на CSS-файл следующим образом:

```
<linkhref="smartmenus/sm-blue.css" rel="stylesheet" >
```

Обновите элемент `ul`, чтобы он выглядел так:

```
<ul class="sm sm-blue">
```

Вы можете найти готовую версию данного руководства в файле *готовый_07_03.html*. Кроме того, вы можете увидеть примеры двух других тем SmartMenu в файлах *готовый_07_03_blue.html* и *готовый_07_03_clean.html*.

Настройка внешнего вида плагина SmartMenu

Самый простой способ изменения внешнего вида навигационной панели SmartMenu заключается в редактировании CSS-файла. Предоставленные CSS-файлы содержат множество полезных комментариев,

объясняющих принципы работы стилей. Например, в файле *sm-simple.css* содержится групповой селектор `sm-simple, .sm-simple ul`. Данное правило отвечает за форматирование всей навигационной панели (внешний элемент `ul`) и каждое подменю (то есть каждый элемент `ul` в навигационной панели).

Вы даже можете создать собственные стили следующим образом.

1. **Сохраните копию одной таблицы стилей, форматирование которой вам больше всего нравится, и присвойте ей имя.**

   Например, сохраните файл *sm-simple.css* под именем *sm-mymenu.css*.

2. **Замените префикс таблицы стилей, например, `.sm-simple`, собственным префиксом.**

   Например, замените префикс `.sm-simple` значением `.sm-mymenu` (или любым другим). Это легко сделать, используя функцию поиска и замены, предусмотренную в текстовом редакторе. Кроме того, этот шаг не является обязательным. При желании вы можете оставить прежнее имя класса, например, `.sm-simple`, которое уже присутствует в таблице стилей.

3. **Внесите изменения в стили.**

   Отредактируйте стили любым способом. Стиль `.sm-simple a` контролирует внешний вид каждой кнопки, а стиль `.sm-simple a span.sub-arrow` определяет внешний вид индикатора, появляющегося рядом с кнопками, к которым прикреплены подменю.

4. **Прикрепите к веб-странице новую таблицу стилей.**

   Этот шаг предусматривает изменение ссылки так, чтобы она указывала на новую таблицу стилей:

   ```
   <link rel="stylesheet" href="smartmenus/sm-mymenu.css">
   ```

5. **Если вы изменили префикс, использованный в таблице стилей, например, заменили `.sm-simple` на `.sm-mymenu`, то вам нужно отредактировать имя класса в элементе `ul` навигационной панели. Например:**

   ```
   <ul class="sm sm-mymenu">
   ```

   Плагин SmartMenus прост в использовании и позволяет легко добавлять отзывчивые выпадающие меню.

ПАРА СЛОВ О ПЛАГИНАХ

Другие плагины библиотеки jQuery для улучшения навигации

Плагин jQuery SmartMenus прост и эффективен, однако существует множество других плагинов для создания расширенной навигации.

- Плагин jPanel (**jpanelmenu.com**) создает меню в стиле панелей, которые выезжают при щелчке по значку. Подобная навигация используется на сервисах Facebook и Google, а также во многих приложениях для смартфонов.
- Плагин Multi-level Push Menu (**multi-levelpush-menu.make.rs**) — это еще одна система меню, которая располагается на краю экрана. Она предусматривает множество уровней навигации, представленных небольшими вкладками, которые посетитель может открывать и закрывать. Это хороший способ для организации и предоставления доступа к большой коллекции ссылок.

Если ни один из этих плагинов не заинтересовал вас, обратитесь к большому списку, состоящему из 15 плагинов навигации для библиотеки jQuery на сайте **speckyboy.com/2013/08/01/15-responsive-navigation-jqueryplugins/**.

Глава 8
УЛУЧШЕНИЕ ВЕБ-ФОРМ

С первых дней существования Всемирной паутины формы служат для сбора информации от посетителей сайтов: электронных адресов для рассылки новостей, данных, необходимых для покупки товара или просто обратной связи с клиентами. Формы также заставляют пользователя *думать*: читать надписи, вводить информацию, делать выбор и т. д. Поскольку некоторые сайты целиком зависят от получения данных с помощью форм (интернет-магазин Ozon не оставался бы в бизнесе столько лет, если бы потребители не могли пользоваться формами для заказа товаров), веб-дизайнеры работают над тем, чтобы они были как можно более удобными. К счастью, язык JavaScript способен придать формам интерактивность и помочь вам сделать их более легкими в использовании и точно отвечающими запросам посетителей сайта.

Структура форм

Язык HTML предлагает разнообразные HTML-элементы для построения веб-форм, подобных той, которая изображена на рис. 8.1. Наиболее важен элемент `form`, определяющий начало (открывающий тег `<form>`) и конец (закрывающий тег `</form>`) формы. Он также указывает на метод, который форма использует для передачи данных (отправка (*post*) или получение (*get*)), и определяет, в какое место во Всемирной паутине должны быть отправлены данные формы.

Вы создаете *элементы формы* — кнопки, текстовые поля и раскрывающиеся списки — с помощью элементов `input`, `textarea` и `select`, соответственно. Большинство элементов форм используют HTML-элемент `input`. Например, текстовые поля, поля для ввода пароля, переключатели, флажки и кнопки подтверждения используют один и тот же элемент `input`. Вы указываете, к какому элементу формы относится `input`, используя атрибут `type`. Например, вы создаете текстовое поле с помощью элемента `input` и присваиваете атрибуту `type` значение `text`:

```
<input name="user" type="text">
```

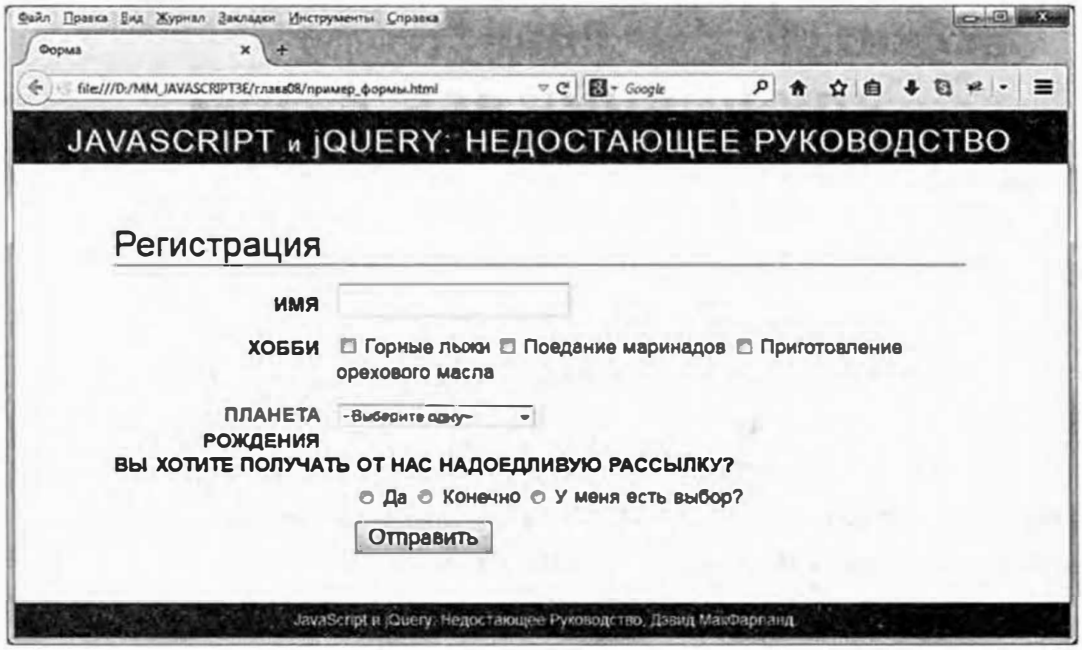

Рис. 8.1. Простая форма может включать разнообразные типы элементов управления, в том числе текстовые поля, переключатели, флажки, раскрывающиеся списки, кнопки подтверждения и т. д. Для получения более полной информации о HTML-формах и способах их использования посетите страницу: **developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms**

Ниже представлен HTML-код для создания формы, приведенной на рис. 8.1; элемент `form` и элементы формы выделены полужирным шрифтом:

```
<form action="process.php" method="post" ↵
name="signup" id="signup">
<div>
<label for="username" class="label">Имя</label>
<input name="username" type="text" id="username" ↵
size="36">
</div>
<div><span class="label">Хобби</span>
<input type="checkbox" name="hobby" id="heliskiing" ↵
value="heliisking">
<label for="heliskiing">Горные лыжи</label>
<input type="checkbox" name="hobby" id="pickle" ↵
value="pickle">
<label for="pickle">Поедание маринадов</label>
```

```
<input type="checkbox" name="hobby" id="walnut" ↵
value="walnut">
<label for="walnut">Приготовление орехового масла ↵
</label>
</div>
<div>
<label for="planet" class="label">Планета ↵
 рождения</label>
<select name="planet" id="planet">
<option>Земля</option>
<option>Марс</option>
<option>Альфа Центавра</option>
<option>Вы о такой не слышали</option>
</select>
</div>
<div class="labelBlock">Вы хотите получать от нас ↵
 надоедливую рассылку?</div>
<div class="indent">
<input type="radio" name="spam" id="yes" value="yes" ↵
checked="checked">
<label for="yes">Да</label>
<input type="radio" name="spam" id="definitely" ↵
value="definitely">
<label for="definitely">Конечно</label>
<input type="radio" name="spam" id="choice" ↵
value="choice">
<label for="choice">У меня есть выбор?</label>
</div>
<div>
<input type="submit" name="submit" id="submit" ↵
value="Отправить">
</div>
</form>
```

ПРИМЕЧАНИЕ

`label` в этом примере является еще одним элементом, обычно используемым в коде форм. Он не создает элемент формы, вроде кнопки, а добавляет видимую на странице текстовую надпись, которая поясняет назначение конкретного элемента формы.

Выбор элементов формы

Как вы уже не раз видели в настоящей книге, прежде чем начать работу с элементами страницы, следует сначала их выбрать. Например, для определения значения, хранящегося в элементе формы, вы должны выбрать его. Подобным образом, если вы хотите скрыть или показать элементы формы, вы должны использовать код JavaScript для идентификации данных элементов.

Как вы знаете, библиотека jQuery может использовать практически любой селектор CSS для выбора элементов страницы. Простейшим способом выбора элемента формы является присвоение ему идентификатора:

```
<input name="user" type="text" id="user">
```

Затем допустимо использовать функцию выбора jQuery:

```
var userField = $('#user');
```

Выбрав элемент формы, вы можете с ним что-нибудь сделать. Например, вы хотите определить значение, введенное посетителем в элемент формы. Если он имеет идентификатор `user`, вы можете использовать jQuery для доступа к значению элемента формы:

```
var fieldValue = $('#user').val();
```

ПРИМЕЧАНИЕ

JQuery-функция `val()` описывается в следующем разделе.

А если вы желаете выбрать все элементы формы определенного типа? Например, вам нужно добавить события щелчка для всех переключателей на странице.

Поскольку элемент `input` используется не только переключателями, но и текстовыми полями, полями ввода пароля, флажками, кноп-

ками подтверждения и сброса, а также скрытыми элементами, то вы не можете просто выбрать элемент `input`. Вместо этого вы должны иметь возможность найти конкретный тип элемента `input`.

Вы могли бы использовать селектор атрибута CSS следующим образом:

```
$('input[type="radio"]')
```

Однако библиотека jQuery предусматривает более простой способ выбора конкретных типов элементов формы (табл. 8.1). Используя один из селекторов jQuery, вы легко можете идентифицировать все элементы определенного типа и работать с ними. Например, после заполнения посетителем формы вам нужно проверить, во все ли поля введено значение. Вам необходимо выбрать все текстовые поля, а затем проверить каждое на наличие значения. Библиотека jQuery позволяет выполнить данное действие таким образом:

```
$(':text')
```

Затем вы просто прорабатываете результаты с помощью функции `.each()` (см. раздел «Работа с каждым элементом выборки» главы 4), чтобы убедиться, что каждое поле содержит значение (подробнее о проверке элементов формы вы узнаете в разделе «Проверка формы» далее в этой главе).

Табл. 8.1. Библиотека jQuery располагает целым рядом селекторов, облегчающих работу с конкретными типами элементов формы

Селектор	Пример	Описание
`:input`	`$(':input')`	Выделяет все элементы типа *input*, *textarea*, *select* и *button*, то есть все элементы формы
`:text`	`$(':text')`	Выбирает все текстовые поля
`:password`	`$(':password')`	Выбирает все поля для ввода пароля
`:radio`	`$(':radio')`	Выбирает все переключатели
`:checkbox`	`$(':checkbox')`	Выбирает все флажки
`:submit`	`$(':submit')`	Выбирает все кнопки подтверждения (отправки)
`:image`	`$(':image')`	Выбирает все кнопки-рисунки
`:reset`	`$(':reset')`	Выбирает все кнопки сброса
`:button`	`$(':button')`	Выбирает все кнопки
`:file`	`$(':file')`	Выбирает все поля файла (применяется для выгрузки файлов на сайт)
`:hidden`	`$(':hidden')`	Выбирает все скрытые элементы

Вы можете комбинировать селекторы формы с другими селекторами. Например, у вас на странице есть две формы, и вы хотите выбрать текстовые поля только в одной из них.

Исходя из того что форма с полями, которые вам нужны, имеет идентификатор `signup`, вы можете выбрать поля только данной формы следующим образом:

```
$('#signup :text')
```

Кроме того, библиотека jQuery предлагает весьма полезные фильтры, которые отбирают элементы формы, соответствующие определенному состоянию.

- `:checked` выбирает все отмеченные или установленные элементы формы, то есть флажки и переключатели. Например, если вам нужно найти все элементы такого рода, используйте следующий код:

```
$(':checked')
```

Этот фильтр можно применять для выяснения того, какой переключатель внутри группы был отмечен. Например, у вас есть группа переключателей («Выберите способ доставки») с различными значениями (например, «Почта», «Курьер» и «Самовывоз»), и вы хотите найти значение переключателя, которое выбрал ваш посетитель. Данная группа использует один и тот же HTML-атрибут `name`; допустим, ваша группа переключателей использует для этого атрибута имя `shipping`. Применение селектора атрибута jQuery (см. раздел «Специальные селекторы» главы 4) в сочетании с фильтром `:checked` позволяет найти значение отмеченного переключателя:

```
var checkedValue = $('input[name="shipping"] ↵
:checked').val();
```

Селектор `$('input[name="shipping"]')` выбирает все элементы ввода с именем `shipping`, но после добавления части `:checked` — `$('input[name="shipping"]:checked')` — только отмеченные элементы. Функция `val()` возвращает значение, хранящееся в отмеченном элементе, например «Почта».

- `:selected` выбирает все отмеченные элементы *option* внутри списка или меню, позволяя вам узнать, какой выбор сделал посетитель (элемент `select`). Например, у вас есть элемент `select` с идентификатором `region`, в котором перечислены все области России. Для

выяснения того, какую область выбрал ваш посетитель, напишите следующее:

```
var selectedRegion=$('#region :selected').val();
```

Заметьте, что, в отличие от примера с фильтром `:checked`, между идентификатором и фильтром есть пробел (`'#region :selected'`). Это объясняется тем, что этот фильтр выбирает элементы `option`, а не `select`. Говоря проще, данный выбор jQuery означает: «Найди все выбранные варианты, находящиеся внутри элемента `select` и имеющие идентификатор `region`». Пробел уподобляет работу данного метода действию дочернего селектора CSS: сначала он находит элемент с нужным идентификатором, а затем внутри него ищет все выбранные элементы.

ПРИМЕЧАНИЕ

Вы можете задействовать несколько селекторов для меню `select`. Это означает, что фильтр `:selected` может возвращать больше одного элемента.

Получение и установка значений элементов форм

Иногда возникает необходимость проверить значение элемента формы. Например, вы хотите убедиться, что электронный адрес посетителя введен в соответствующее поле. Или вам нужно знать значение поля для подсчета общей стоимости заказа. С другой стороны, вам может понадобиться задать значение элемента формы. Скажем, у вас есть форма, запрашивающая информацию как по платежу, так и по доставке заказа. Было бы полезно с помощью флажка «Аналогично информации по платежу» предоставить вашим клиентам возможность заполнять элементы формы для сведений о доставке автоматически на основе данных, взятых из элементов, содержащих платежную информацию.

Библиотека jQuery предлагает простую функцию для выполнения обеих задач. Функция `val()` может как задавать, так и считывать значения элементов формы. Если вы вызываете ее без передачи аргументов, она будет считывать значения; при передаче функции значения она введет его в элемент формы. Например, у вас есть поле для ввода электронного адреса клиента с идентификатором `email`. Его значение можно выяснить, введя следующий код:

```
var fieldValue = $('#email').val();
```

Задать значение элементу формы можно просто передав это значения функции `val()`. Например, у вас есть форма для заказа товаров и вы хотите автоматически рассчитать общую стоимость заказа, исходя из количества единиц товара, заявленного клиентом (рис. 8.2). Вы можете *получить* количество заказанных единиц, умножить его на цену товара и затем *установить* значение в поле с итоговой стоимостью.

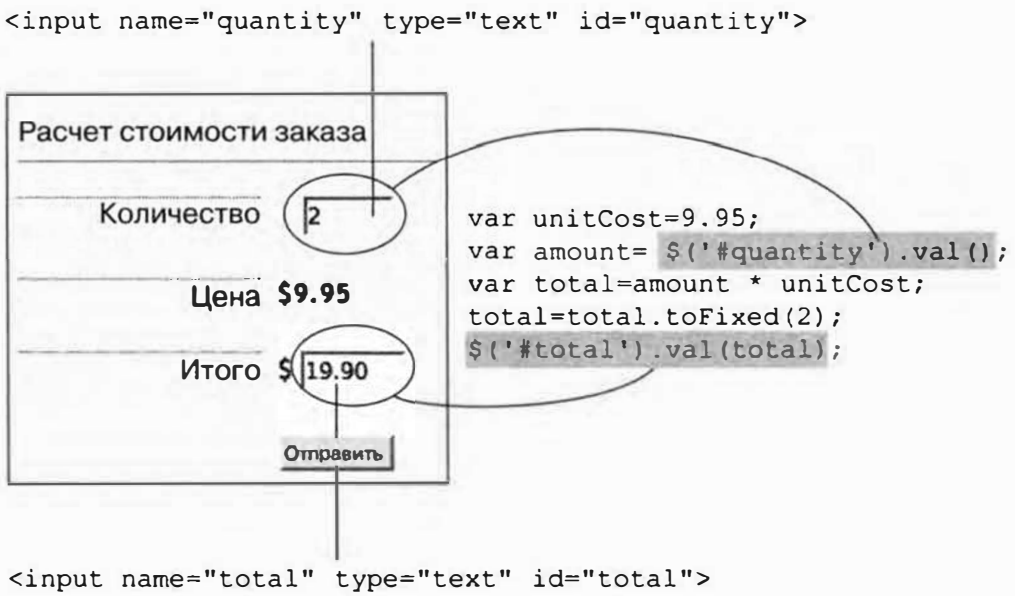

Рис. 8.2. Библиотека jQuery облегчает задачу получения и установки значения элемента формы

Код для выполнения этой задачи выглядит так:

```
1 var unitCost=9.95;
2 var amount=$('#quantity').val(); // получить значение
3 var total=amount*unitCost;
4 total=total.toFixed(2);
5 $('#total').val(total); // задать значение
```

Строка 1 кода создает переменную, которая хранит значение цены товара. Строка 2 создает еще одну переменную и извлекает значение, введенное посетителем в поле с идентификатором `quantity`, соответствующее количеству заказанного товара. Строка 3 определяет общую стоимость, полученную путем умножения цены на количество, а строка 4 форматирует результат, включая в него два знака после запятой (см. раздел «Форматирование значений в валюте» главы 16 для получения информации о методе `toFixed()`). Наконец, строка 5 вводит итоговое

значение в поле с идентификатором `total` (как задействовать этот код, используя событие, вы узнаете далее).

Проверка активации кнопок и флажков

Хотя функция `val()` полезна для получения значений любого элемента формы, иногда подобное значение имеет смысл лишь в том случае, если посетитель выбрал какой-то конкретный элемент. Например, переключатели и флажки требуют от посетителя сделать выбор определенного значения. Вы уже видели, как использовать фильтр `:checked` для поиска отмеченных переключателей или флажков, но как только вы их найдете, вам понадобится способ для определения статуса конкретного переключателя или флажка.

В языке HTML атрибут `:checked` определяет, помечен ли данный элемент. Например, для установки флажка при загрузке веб-страницы следует добавить атрибут `checked` следующим образом (для XHTML):

```
<input type="checkbox" name="news" id="news" ↵
checked="checked"/>
```

А для HTML5:

```
<input type="checkbox" name="news" id="news" checked>
```

Несмотря на то, что в первом примере — `checked="checked"` — кажется, что атрибут `checked` является атрибутом языка HTML, на самом деле он представляет собой свойство DOM (см. раздел «Объектная модель документа» главы 4). То есть это свойство элемента «флажок», которое может динамически изменяться, когда посетитель устанавливает и сбрасывает его. Многие другие элементы формы также обладают динамическими свойствами: например, текстовые поля предусматривают свойство `disabled`, определяющее возможность (значение `true`) или невозможность (значение `false`) ввода текста в это поле.

В случае со свойствами DOM вы используете метод jQuery `prop()` следующим образом:

```
if ($('#news').prop('checked')) {
// флажок установлен
} else {
// флажок сброшен
}
```

Код `$('#news').prop('checked')` возвращает значение `true`, если флажок установлен. Если он сброшен, возвращается значение `false`. Таким образом, данная управляющая инструкция позволяет выполнить один набор задач, если флажок установлен, либо другой набор задач, если флажок сброшен. Если вам нужно освежить знания об управляющих инструкциях, обратитесь к разделу «Управляющие инструкции» главы 3.

Свойство `checked` применимо и к переключателям. Вы можете использовать функцию `prop()` таким же образом, чтобы проверить, установлен ли атрибут `checked` для переключателя.

События формы

Как вы знаете из главы 5, события позволяют сделать вашу страницу интерактивной, реагируя на различные действия посетителя. Формы и их элементы способны отвечать на события, так что вы можете создавать формы, интеллектуально реагирующие на действия пользователя.

Подтверждение

Когда посетитель подтверждает заполнение формы нажатием кнопки **Подтвердить** (Submit) или клавиши **Enter**, происходит событие `submit`. Вы можете выполнить сценарий во время подтверждения формы. Таким образом, код JavaScript может верифицировать элементы формы на предмет их корректного заполнения. Когда форма подтверждена, программа JavaScript проверяет ее элементы и при обнаружении проблем останавливает передачу и сообщает посетителю причину отказа. Если все заполнено верно, то форма передается как обычно.

Для выполнения функции при наступлении события подтверждения формы сначала выберите форму, затем примените jQuery-функцию `submit()`, чтобы добавить ваш сценарий. Предположим, вам нужно убедиться, что при передаче формы поле с именем, показанное на рис. 8.1, заполнено, другими словами, посетитель не может оставить его незаполненным. Сделать это можно путем добавления события подтверждения в форму и проверки значения элемента перед ее отправкой. Если поле пусто, то вам нужно дать знать об этом посетителю и остановить процесс передачи формы; в противном случае форма будет передана с пустым полем.

Если вы взглянете на HTML-код формы в начале данной главы, то увидите, что форма имеет идентификатор `signup`, а поле ввода имени — идентификатор `username`. Верифицировать эту форму с помощью библиотеки jQuery можно так:

```
1 $(document).ready(function() {
2     $('signup').submit(function() {
3           if ($('#username').val() == '') {
4                 alert('Введите имя в поле "Имя".');
5                 return false;
6           }
7     }); // конец submit()
8 }); // конец ready()
```

Строка 1 вводит необходимую функцию `$(document).ready()`, так что код выполняется только после загрузки HTML-кода страницы (см. раздел «Больше концепций для событий jQuery» главы 5). Строка 2 присоединяет функцию к событию формы `submit`. Строки 3–6 выполняют проверку. Строка 3 проверяет, не является ли значение элемента формы пустой строкой (`''`). Если элемент пуст, то появляется окно оповещения, дающее посетителю понять, что он ошибся.

Строка 5 очень важна: она останавливает процесс передачи формы. Если пропустить этот шаг, то форма будет передана без имени пользователя. Строка 6 завершает управляющую инструкцию, а строка 7 является окончанием функции `submit()`.

ПРИМЕЧАНИЕ

Остановить отправку формы можно также с помощью функции объекта события `prevent-Default()`, описанной в разделе «Отмена обычного поведения событий» главы 5.

```
$('form').submit(function(evt) {
// остановить отправку формы
evt.preventDefault();
}
```

Событие `submit` применяется только к формам, так что вы должны выбрать форму, а затем присоединить к ней функцию события `submit`. Выбор формы осуществляется с помощью идентификатора, содержаще-

гося в HTML-элементе `form`, или, если страница имеет единственную форму, с использованием селектора элемента:

```
$('form').submit(function() {
// код, выполняемый во время отправки формы
});
```

Активное состояние

Когда текстовый курсор находится в поле (либо после щелчка по элементу формы кнопкой мыши, либо при переходе на него с помощью клавиши **Tab**), элемент формы переходит в активное состояние, называемое *focus*. Фокус представляет собой событие, запускаемое браузером и указывающее на то, что текстовый курсор находится в конкретном поле или выделен определенный элемент формы. Вы можете быть уверены, что именно здесь сосредоточено внимание посетителя. Вероятно, вы не будете использовать это событие слишком часто, но некоторые дизайнеры применяют его для удаления любого текста, который уже присутствует в поле. Например, у вас есть следующий HTML-код внутри формы:

```
<input name="username" type="text" id="username" ↵
value="Пожалуйста, введите ваше имя пользователя">
```

Этот код создает текстовое поле в форме с текстом «Пожалуйста, введите ваше имя пользователя» внутри него. Этот метод позволяет давать пользователю указания по заполнению формы. Посетитель не должен сам стирать ненужный текст, вы это делаете автоматически при активации поля:

```
1 $('#username').focus(function() {
2     var field = $(this);
3     if (field.val()==field.prop('defaultValue')) {
4           field.val('');
5     }
6 });
```

Строка 1 выбирает поле (имеющее идентификатор `username`) и присваивает функцию событию `focus`. Строка 2 создает переменную `field`, которая хранит ссылку на выборку jQuery; как сказано в разделе «Ключевые слова `this` и `$(this)`» главы 4, ключевое слово `$(this)` ссылается на выбранный элемент внутри jQuery-функции, в нашем случае — на элемент формы.

Строка 4 стирает содержимое поля. Она задает новое значение (пустую строку (`''`)) элемента формы, удаляя таким образом любое присутствующее в поле значение. Однако вам не нужно обнулять значение этого элемента всякий раз, когда он оказывается в фокусе. Например, пользователь выделил поле и вместо прежней надписи «Пожалуйста, введите ваше имя пользователя» ввел свое имя. Если затем текстовый курсор покинет данное поле, а потом вернется, то введенное имя не должно исчезнуть. Вот когда вступает в игру управляющая инструкция, содержащаяся в строке 3.

Текстовые поля имеют атрибут `defaultValue`, представляющий текст, содержащийся в поле при первоначальной загрузке страницы. Даже если вы стерли этот текст, браузер будет его помнить. Управляющая инструкция проверяет, соответствует ли текущий текст поля (`field.val()`) тому, который был первоначально (`field.prop('defaultValue')`). Если текст совпадает, то интерпретатор JavaScript стирает текст в поле.

Рассмотрим этот процесс подробнее. В прошлом примере при загрузке HTML-кода в текстовом поле был текст «Пожалуйста, введите ваше имя пользователя». Это первоначальное значение элемента формы, то есть `defaultValue`. Когда посетитель впервые переходит в поле, управляющая инструкция задает вопрос: «Совпадает ли текущее содержимое поля с тем содержимым, которое было в нем при загрузке страницы?» Другими словами, эквивалентна ли запись «Пожалуйста, введите ваше имя пользователя» записи «Пожалуйста, введите ваше имя пользователя». Ответ положителен, поэтому поле очищается.

Однако, если вы, к примеру, ввели в качестве имени пользователя *helloKitty*, затем перешли к следующему элементу формы, но вдруг осознали, что ошиблись с именем, то, вернувшись назад, вы вновь запускаете событие фокуса и функцию, присвоенную данному событию. Инструкция спрашивает: «Эквивалентна ли запись helloKitty записи “Пожалуйста, введите ваше имя пользователя?”». На этот раз ответ отрицателен и текст поля не стирается, а вы можете исправить свою ошибку.

ПРИМЕЧАНИЕ

Формы HTML5 предусматривают атрибут `placeholder`, который позволяет поместить временное сообщение в текстовое поле. Когда посетитель начинает ввод данных, этот заполнитель стирается:

```
<input name="username" type="text" id="username" ↵
placeholder="Пожалуйста, введите ваше имя пользователя">
```

Данная техника проще, чем вышеописанный способ использования библиотеки jQuery, однако атрибут `placeholder` не работает в браузере Internet Explorer 9 или более ранней версии.

Неактивное состояние

Когда вы покидаете элемент формы или щелкаете кнопкой мыши за его пределами, браузер запускает событие `blur`, которое обычно применяется к полям `text` и `textarea` для выполнения сценария проверки формы, когда кто-то покидает заполненное текстовое поле. Например, у вас есть длинная форма с большим количеством вопросов, многие из которых требуют определенных типов ответов (электронные адреса, телефоны, почтовые индексы и т. д.). Допустим, посетитель не смог заполнить все элементы формы правильно, но нажал на кнопку отправки формы и получил длинный список ошибок, указывающих на неправильное заполнение формы. Вместо того чтобы сваливать на него все сразу, вы можете проверять значения элементов формы параллельно их заполнению. В таком случае при первой же ошибке посетитель сразу получит предупреждение и сможет ее исправить.

Предположим, у вас есть поле для сбора информации о количестве товаров, нужных клиенту. HTML-код может иметь такой вид:

```
<input name="quantity" type="text" id="quantity">
```

Вы хотите убедиться, что поле содержит только числа (то есть 1, 2 или 9, но не «Один», «Два» или «Девять»). Вы можете это сделать после того, как посетитель покинул данное поле:

```
1 $('#quantity').blur(function() {
2     var fieldValue=$(this).val();
3     if (isNaN(fieldvalue)) {
4           alert('Пожалуйста, введите число');
5     }
6 });
```

Строка 1 присваивает событию `blur` функцию. Строка 2 извлекает значение элемента формы и сохраняет его в переменной `fieldValue`. Строка 3 проверяет числовую природу значения методом `isNaN()` (см. раздел «Тест на числа» главы 16). Если значение не является числом, выполняется строка 4 и выводится сообщение об ошибке.

При наличии формы с множеством элементов формы, принимающих числовые значения, вы можете назначить каждому из этих элементов одно и то же имя класса, например, `class="numOnly"`, и проверить каждое из них с помощью следующего кода:

```
1 $('.numOnly').blur(function() {
2     var fieldValue=$(this).val();
3     if (isNaN(fieldValue)) {
4           alert('Введите число');
5     }
```

Этот метод позволяет вам использовать всего несколько строк кода для проверки каждого числового элемента.

Щелчок мышью

Событие `click` происходит при щелчке кнопкой мыши по элементу формы. Оно особенно полезно для переключателей и флажков, поскольку вы можете добавлять функции, изменяющие форму на базе значений, выбираемых посетителем. Например, у вас есть форма заказа, предоставляющая отдельные элементы формы для информации по платежу и доставке. Для предотвращения повторного ввода одних и тех же данных в случаях, когда информация совпадает, следует установить флажок «Аналогично информации по платежу», что скрывает информацию по доставке и делает форму удобнее для чтения. (Вы увидите данный прием в действии в разделе «Сокрытие элементов формы» данной главы.)

Как и в случае с другими событиями, вы можете использовать jQuery-функцию `click()` для ее присвоения событию щелчка по элементу формы:

```
$(':radio').click(function() {
//функция будет применена к каждому переключателю ↵
при щелчке по нему
});
```

ПРИМЕЧАНИЕ

Событие `click` применимо и к текстовым полям, но не так же, как в случае с событием фокуса. Событие `focus` запускается при переходе (клавишей **Tab**) или при щелчке в текстовом поле, событие же щелчка происходит только при щелчке кнопкой мыши в текстовом поле.

Смена

Событие `change` применяется к раскрывающимся спискам формы (рис. 8.3). Сделав выбор в раскрывающемся списке, вы запускаете событие смены.

Вы можете использовать его для выполнения функции проверки; многие дизайнеры в качестве первого пункта раскрывающегося списка обычно добавляют инструкцию, например, «Выберите страну».

Чтобы убедиться, что посетитель выбрал страну, проверяйте значение всякий раз, когда кто-то делает новый выбор.

Рис. 8.3. Событие `change` раскрывающегося списка формы позволяет делать интересные вещи, когда пользователь выбирает пункт в раскрывающемся списке. Выбор пункта из верхнего раскрывающегося списка динамически изменяет список пунктов, представленных во втором раскрывающемся списке. Выбрав товар из верхнего раскрывающегося списка, во втором вы получите варианты доступных цветов для этого товара

Вы также можете запрограммировать изменение формы на основании выбора пункта раскрывающегося списка. Например, вы можете выполнить функцию таким образом, чтобы при выборе варианта из раскрывающегося списка, изменялись бы пункты второго раскрывающегося списка. На рис. 8.3 показана форма с двумя раскрывающимися списками; выбор пункта из верхнего раскрывающегося списка изменяет список доступных цветов в нижнем.

Чтобы применить к раскрывающемуся списку обработчик события `change`, используйте jQuery-функцию `change()`. Предположим, у вас есть раскрывающийся список с идентификатором `country` с названиями стран; всякий раз, когда делается новый выбор, вы хотите быть уверены, что новое значение не соответствует тексту «Выберите страну». В данном случае используйте следующий код:

```
$('#country').change(function() {
if ($(this).val()=='Выберите страну') {
alert('Пожалуйста, выберите страну из данного ↵
раскрывающегося списка.');
}
}
```

Усовершенствование форм

Веб-формы требуют от посетителя сайта определенных усилий: заполнить текстовые поля, выбрать пункт в раскрывающемся списке, установить флажки и т. д. Если вы хотите, чтобы люди делали все это, в ваших же интересах сделать формы как можно более простыми. И язык JavaScript поможет вам в этом. Например, вы можете скрыть элементы формы, пока они не нужны, отключить элементы, неприменимые в данном случае, или рассчитывать итоговые значения, основываясь на выбранных вариантах. Язык JavaScript предлагает множество способов повышения удобства форм.

Фокусировка на первом элементе формы

Обычно, для того чтобы начать заполнение формы, вы должны установить текстовый курсор в первое текстовое поле и начать вводить текст. Если страница предусматривает авторизацию, зачем заставлять

посетителей перемещать указатель мыши на нужное место и только потом вводить текст? Почему бы просто не поместить текстовый курсор сразу в нужное поле, подготовив его таким образом к вводу данных? С JavaScript сделать это проще простого.

Секрет заключается в фокусировке, являющейся не просто событием, на которое код JavaScript может отреагировать, но и командой, которую вы можете отдать, чтобы поместить текстовый курсор в текстовое поле. Вы просто выбираете текстовое поле, после чего выполняете jQuery-функцию `focus()`. Предположим, вы хотите, чтобы при загрузке страницы текстовый курсор был помещен в поле ввода имени, показанном на рис. 8.1. Если вы посмотрите на HTML-код этой формы (см. раздел «Структура форм» в начале данной главы), то увидите, что это поле имеет идентификатор `username`. Чтобы с помощью кода JavaScript поместить текстовый курсор в это поле, напишите следующее:

```
$(document).ready(function() {
$('#username').focus();
});
```

В данном примере текстовое поле имеет идентификатор `username`. Однако вы можете создать универсальный сценарий, который всегда будет вводить в фокус первое текстовое поле формы без необходимости присвоения ему идентификатора:

```
$(document).ready(function() {
$(':text:first').focus();
});
```

Как вы прочли ранее в главе, библиотека jQuery предлагает удобный способ выбора всех текстовых полей — `$(':text')`. Добавляя к селектору фрагмент кода `:first`, вы можете выбрать первый экземпляр данного элемента, так что селектор jQuery `$(':text:first')` выбирает первое текстовое поле на странице. Добавление части `.focus()` устанавливает текстовый курсор в данное поле, которое терпеливо ожидает ввода данных со стороны посетителя.

Если ваша страница содержит более одной формы (например, формы «Поиск по сайту» и «Подпишитесь на рассылку новостей»), то вам нужно уточнить селектор для идентификации формы, чье текстовое поле должно получить фокус. Например, вам нужно, чтобы первым полем, в котором окажется текстовый курсор, было поле формы для

подписки, хотя первым текстовым полем по очереди является поле поисковой формы. Чтобы задать фокус нужному полю, просто добавьте идентификатор (например, `signup`) в форму и примените этот код:

```
$(document).ready(function() {
$('#signup :text:first').focus();
});
```

Теперь селектор `$('#signup :text:first')` выбирает только первое поле внутри формы для подписки.

Отключение/включение элементов формы

Элементы формы обычно предназначены для заполнения. В конце концов, какой толк от элемента формы, которое нельзя заполнить? Однако бывает так, что вы *не* хотите, чтобы посетитель заполнял текстовые поля, устанавливал флажки или выбирал вариант из раскрывающегося списка. Скажем, у вас есть поле, которое следует заполнять, только если для предыдущего элемента формы установлен флажок. Например, форма 1040 для определения подоходного налога в США имеет поле ввода номера социального страхования супруги (супруга). Заполняется оно только в том случае, если вы женаты (замужем).

Чтобы «отключить» элемент формы, не предназначенный для заполнения, вы можете деактивировать его с помощью кода JavaScript. Деактивация означает, что возле пункта не может быть установлен флажок или переключатель, в текстовые поля нельзя ввести текст, в раскрывающемся списке — выбрать пункт, нельзя также нажать кнопку подтверждения.

Чтобы отключить элемент формы, задайте для атрибута `disabled` значение `true`. Например, для отключения всех полей ввода формы можно использовать следующий код:

```
$(':input').prop('disabled', true);
```

Обычно вы выключаете элемент формы в ответ на событие. Пользуясь примером с формой 1040, вы можете деактивировать поле ввода номера социального страхования супруга (супруги), когда выбран вариант «Холост». Исходя из того что переключатель для объявления себя холостым имеет идентификатор `single`, а поле для ввода номера социального страхования супруга (супруги) — идентификатор `spouseSSN`, код JavaScript будет выглядеть следующим образом:

```
$('#single').click(function() {
$('#spouseSSN').prop('disabled', true);
});
```

Разумеется, отключая элемент формы, вы можете в дальнейшем захотеть вновь включить его. Для этого присвойте атрибуту `disabled` значение `false`. Например, для активации всех элементов формы используйте код:

```
$(':input').prop('disabled', false);
```

ПРИМЕЧАНИЕ

При отключении элемента формы убедитесь, что вы применяете логические значения (см. раздел «Логический тип данных» главы 2) *true* или *false*, а не строки *'true'* или *'false'*. Например, следующий код ошибочен:

```
$(':input').prop('disabled', 'false');
```

А этот верен:

```
$(':input').prop('disabled', false);
```

Вернемся к примеру с налоговой формой. Если посетитель выбирает параметр «Женат», то он должен помнить, что поле для ввода номера социального страхования супруга (супруги) имеет активный статус. Полагая, что переключатель для параметра «Женат» имеет идентификатор `married`, вы можете добавить такой код:

```
$('#married').click(function() {
$('#spouseSSN').prop('disabled', false);
});
```

Вы разберетесь с этой техникой подробнее в руководстве далее в этой главе.

Сокрытие/отображение параметров формы

Помимо отключения, существует еще один способ не утруждать посетителей заполнением ненужных элементов формы — их просто можно скрыть. В примере с налоговой формой вы можете просто скрыть поле для ввода номера социального страхования супруга (супруги), если выбран пункт «Холост», или показать его, если активирован пункт «Женат». Сделать это можно следующим образом:

```
$('#single').click(function() {
$('#spouseSSN').hide();
});
$('#married').click(function() {
$('#spouseSSN').show();
});
```

ПРИМЕЧАНИЕ

Функции библиотеки jQuery `hide()` и `show()` (а также другие функции для отображения и сокрытия элементов) описаны в разделе «Основы отображения и сокрытия» главы 6.

К преимуществам сокрытия элементов формы (в противоположность их отключению) относится упрощение макета формы. В конце концов, отключенный элемент формы остается видимым, что может привлекать (а точнее, отвлекать) внимание пользователя.

Во многих случаях вам нужно скрыть или показать не просто элемент формы: возможно, вы захотите скрыть название этого элемента или любой другой связанный с ним текст. Один из способов — заключить код, который вы хотите скрыть (элемент, название и т. д.), в элемент `div`, присвоить ему идентификатор, а затем скрыть этот элемент `div`. Описание данного способа приведено в следующем руководстве.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

Предотвращение многократной отправки формы

Иногда получается так, что одна и та же информация отправляется несколько раз. Как этого избежать?

Веб-серверы не самые быстрые создания... это же можно сказать и об Интернете. Зачастую имеет место задержка по времени от момента, когда посетитель нажал кнопку отправки до появления сообщения «Ваша информация получена». Иногда такая задержка бывает достаточно долгой, чтобы нетерпеливые веб-серферы нажали кнопку отправки еще (и еще, и еще) раз, решив, что в первый раз система не сработала.

Это приводит к тому, что одна и та же информация может быть отправлена несколько раз.

А в случае с онлайновыми продажами это означает, что вы несколько раз расплатитесь с помощью своей кредитной карты! К счастью, язык JavaScript предоставляет легкий способ деактивировать кнопку подтверждения, как только начался процесс отправки. С помощью атрибута `disabled` вы можете сделать так, чтобы эта кнопка больше не нажималась.

Исходя из того что форма имеет идентификатор `formID`, а кнопка передачи — идентификатор `submit`, сначала добавьте в форму функцию `submit()`, а затем внутри функции отключите кнопку передачи:

```
$('#formID').submit(function() {
$('#submit').prop('disabled',true);
});
```

Если на странице присутствует только одна форма, то вам даже не нужно использовать идентификаторы для элементов:

```
$('form').submit(function() {
$('input[type=submit]').
prop('disabled',true);
});
```

Кроме того, разрешается изменять сообщение на кнопке передачи. Например, сначала кнопка называется «Передать», а после начала передачи она же гласит «Информация отправляется». Сделать это можно следующим образом:

```
$('#formID').submit(function() {
var subButton = $(this).find(':submit');
subButton.prop('disabled',true);
subButton.val('Информация отправляется');
});
```

Убедитесь, что данный код помещен внутри функции `$(document).ready()`, как описано в разделе «Больше концепций для событий jQuery» главы 5.

Усовершенствование простой формы на практике

Данное руководство позволит вам усовершенствовать простую форму заказа, состоящую из элементов для ввода информации по платежу и доставке. Во-первых, вы поместите текстовый курсор в первое текстовое поле формы при загрузке страницы. Во-вторых, отключите и вновь вклю-

чите элементы формы в соответствии с выбором посетителя. Наконец, вы скроете целый раздел формы, когда он окажется ненужным (рис. 8.4).

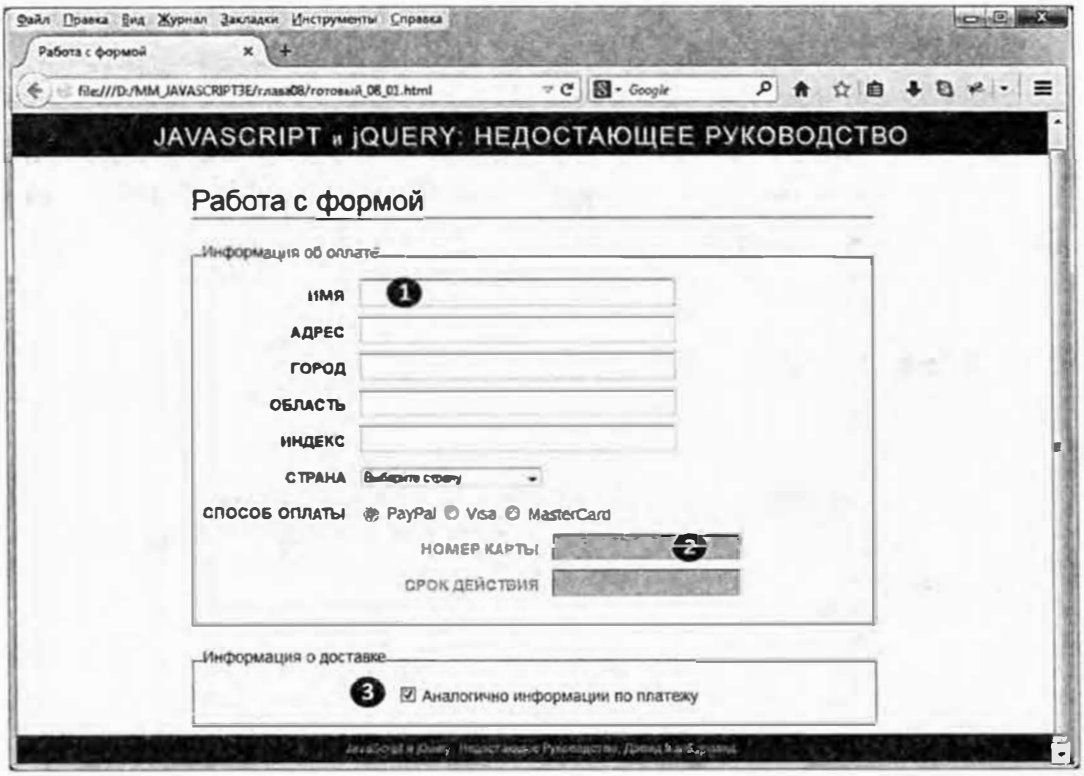

Рис. 8.4. С помощью языка JavaScript можно повысить удобство использования веб-формы и добавить интерактивные свойства: скрыть ненужные элементы формы или отключить те, которые не должны заполняться

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Фокусировка на элементе формы

В первое текстовое поле бланка-заказа в нашем примере вводится имя лица, размещающего заказ (см. рис. 8.4). Для удобства пользователя текстовый курсор должен находиться в этом поле при загрузке страницы.

1. **В текстовом редакторе откройте файл *08_01.html* в каталоге *глава08*.**

   Этот документ уже содержит ссылку на файл jQuery и функцию `$(document).ready()` (см. раздел «Больше концепций для собы-

тий jQuery» главы 5). Форма включает два раздела — один для ввода платежной информации, а второй — для сбора данных о доставке. (Проверьте страницу в браузере, прежде чем продолжить.)

В первом шаге (на самом деле в единственном шаге на данном этапе) нам предстоит сфокусироваться на текстовом поле.

2. **Щелкните мышью в пустой строке после функции `$(document).ready()` и введите: `$(':text:first').focus();`. Код должен выглядеть следующим образом:**

```
$(document).ready(function() {
$(':text:first').focus();
}); // конец ready()
```

Данный код выбирает первое текстовое поле и применяет к нему функцию `focus()`. В результате при загрузке страницы браузер помещает текстовый курсор в данное поле.

Сохраните файл и просмотрите его в браузере.

После загрузки страницы текстовый курсор будет мерцать в первом поле, это значит, что можно начинать заполнение формы.

Отключение элементов формы

Предыдущие действия были лишь разминкой. Сейчас нам предстоит выключить или включить два элемента формы в ответ на выбор, сделанный посетителем. Если вы просмотрите форму в браузере (или посмотрите на рис. 8.4), то увидите, что в конце раздела с платежной информацией есть три переключателя для выбора способа платежа: PayPal, Visa и MasterCard. Кроме того, есть два поля для ввода номера платежной карты и срока ее действия. Эти две настройки применимы только к платежным картам, но не к платежам с помощью системы PayPal, поэтому мы должны отключить данные настройки, если переключатель установлен в положение PayPal.

HTML-код для этого раздела страницы выглядит так (элементы формы выделены полужирным шрифтом):

```
1  <div><span class="label">Способ оплаты</span>
2  <input type="radio" name="payment" id="paypal" ↵
     value="paypal">
```

```
3     <label for="paypal">PayPal</label>
4  <input type="radio" name="payment" id="visa" ↵
      value="visa">
5     <label for="visa">Visa</label>
6  <input type="radio" name="payment" id="mastercard" ↵
      value="mastercard">
7     <label for="mastercard">MasterCard</label>
8  </div>
9  <div id="creditCard" class="indent">
10 <div>
11    <label for="cardNumber" class="label">Номер ↵
      карты</label>
12 <input type="text" name="cardNumber" ↵
      id="cardNumber">
13 </div>
14 <div>
15    <label for="expiration" class="label">Срок ↵
      действия</label>
16 <input type="text" name="expiration" ↵
      id="expiration">
17 </div>
18 </div>
```

3. **Вернитесь в текстовый редактор к файлу *08_01.html*.**

   Теперь вы добавите код, созданный только что. Для начала присвойте функцию событию щелчка для переключателя PayPal.

4. **В сценарий вверху страницы добавьте код, выделенный полужирным шрифтом:**

```
$(document).ready(function() {
$(':text:first').focus();
$('#paypal').click(function() {
}); // конец click
}); // конец ready()
```

   Переключатель для способа платежа PayPal имеет идентификатор `paypal` (см. строку 2 в HTML-коде), таким образом, для выбора это-

го поля достаточно написать: `$('#paypal')`. Остальная часть кода присваивает анонимную функцию событию щелчка (если это вам не ясно, обратитесь к разделу «Использование событий: подход jQuery» главы 5). Иначе говоря, установка переключателя в положение PayPal не только осуществляет выбор способа (это обычное поведение браузера), но и запускает функцию, которую вы создадите далее.

Теперь вы отключите поля для ввода номера платежной карты и срока ее действия, поскольку эти данные не применимы, если выбран вид платежа PayPal.

5. **Внутри анонимной функции, созданной в предыдущем шаге, добавьте новую строку кода (строка 4):**

```
1 $(document).ready(function() {
2   $(':text:first').focus();
3   $('#paypal').click(function() {
4   $('#creditCard input').prop('disabled', true);
5   }); // конец click
6 }); // конец ready()
```

Задачу отключения двух элементов формы можно легко решить с помощью одной строки кода. Оба поля расположены внутри элемента `div` с идентификатором `creditCard` (см. строку 9 HTML-кода выше). Поэтому селектор jQuery `$('#creditCard input')` означает «выбрать все элементы `input` формы внутри элемента с идентификатором `creditCard`». Этот гибкий подход обеспечивает выбор всех полей ввода, так что при добавлении еще одного поля ввода, например, CVV (три цифры на обратной стороне карты, которые могут требоваться для обеспечения безопасности транзакции), оно также будет выделено.

Чтобы отключить элементы формы, вам остается задать значение `true` для атрибута `disabled` (см. раздел «Отключение/включение элементов формы» данной главы). Однако это действие не затрагивает названия полей («Номер карты» и «Срок действия»). Хотя сами поля деактивированы, названия элементов формы по-прежнему выделены ярким полужирным шрифтом, давая пользователю ложный сигнал о том, что поля доступны для заполнения. Для наглядной демонстрации неактивного состояния элементов формы следует изменить цвет их названий на светло-серый. Вместе с этим вы поменяете фоновый цвет полей на серый, чтобы они выглядели неактивными.

6. Добавьте выделенный полужирным шрифтом код в ваш сценарий:

```
$(document).ready(function() {
$(':text:first').focus();
$('#paypal').click(function() {
$('#creditCard input').prop('disabled', ↵
true).css('backgroundColor','#CCC');
$('#creditCard label').css('color','#BBB');
}); // конец click
}); // конец ready()
```

ПРИМЕЧАНИЕ

Символ ↵ в конце строки 4 означает, что следующую строку на самом деле нужно напечатать как часть первой. Но так как длинная строка кода JavaScript не поместится на странице этой книги, пришлось разбить ее на две части. Однако из текста в рамке «Пробелы, табуляция и возврат каретки в языке JavaScript» в главе 2 вы помните, что язык JavaScript весьма гибок в отношении разрывов строк и пробелов, поэтому вы можете (для упрощения чтения кода) помещать одну инструкцию JavaScript на нескольких строках:

```
$('#creditCard input').prop('disabled',true)
.css('backgroundColor','#CCC');
```

Заметьте, что некоторые программисты создают отступ, как в случае с функцией `.css()`, чтобы выровнять ее с функцией `.prop()`.

Сначала используйте jQuery-функцию `css()` для изменения цвета фона текстовых полей (обратите внимание, что соответствующий код является частью строки 4, так что вы должны вписать его в ту же строку, что и функцию `prop()`). Затем примените функцию `css()` для настройки цвета шрифта элементов `label` внутри `div` (функция `css()` описана в разделе «Чтение и изменение свойств CSS» главы 4).

Если вы теперь просмотрите страницу в браузере, то увидите, что установка переключателя в положение PayPal приводит к тому, что поля и их названия «Номер карты» и «Срок действия» становятся тусклыми. Однако если установить переключатель в положение Visa или MasterCard, соответствующие элементы формы остаются неактивными! Вам нужно вновь обеспечить их включение при установке переключателя в любое из этих положений.

7. **После функции `click()` добавьте пустую строку (вы должны вписать новый код между строками 7 и 8 в пункте 6) и введите следующий код:**

```
$('#visa, #mastercard').click(function() {
$('#creditCard input').prop('disabled', ↵
false).css('backgroundColor','');
$('#creditCard label').css('color','');
}); // конец click
```

Селектор `$('#visa, #mastercard')` выбирает оба переключателя (см. строки 4 и 6 HTML-кода в начале руководства). Обратите внимание, что для удаления цветов фона и текста, добавленных при установке переключателя в положение PayPal, нужно просто передать пустую строку в качестве значения цвета: `$('#creditCard label').css('color','');`. Измененный цвет элемента удаляется, но сохраняется цвет по умолчанию, заданный таблицей стилей.

Вы почти закончили. Теперь вам предстоит полностью скрыть выбранную часть страницы.

Сокрытие элементов формы

Подобно многим формам заказов, форма, рассматриваемая в нашем примере, имеет отдельные поля для ввода платежной информации и данных о доставке. Часто эти сведения идентичны, и нет никакой нужды дважды вводить одну и ту же информацию.

Нередко в подобных формах вы видите пункт «Аналогично информации по платежу», возле которого можно установить флажок. Однако не лучше ли совсем убрать (скрыть) элемент формы, раз он оказывается ненужным? Язык JavaScript позволяет сделать это.

1. **Откройте файл *08_01.html* в текстовом редакторе.**

Продолжим работу над кодом, созданным в последних двух разделах данного руководства. Сначала добавим функцию событию установки флажка «Аналогично информации по платежу». HTML-код данного флажка выглядит так:

```
<input type="checkbox" name="hideShip" ↵
id="hideShip">
```

2. **Введите следующий фрагмент после кода, который вы добавили в пункте 4 ранее, но перед окончанием сценария (последняя строка кода: `}); // конец ready()`):**

```
$('#hideShip').click(function() {
});// конец click
```

Поскольку флажок имеет идентификатор `hideShip`, код выбирает его и добавляет функцию событию щелчка. В данном случае вместо сокрытия одного поля вам нужно спрятать целую группу. Поместите HTML-код, описывающий эти элементы формы, в элемент `div` с идентификатором `shipping`: теперь, чтобы спрятать поля, достаточно просто скрыть элемент `div`.

Однако вам нужно скрывать эти поля, только если установлен флажок. Если кто-то щелкнет кнопкой мыши дважды, тем самым сбросив флажок, то элемент `div` и его поля должны вновь стать видимыми. Поэтому сначала следует выяснить, установлен ли флажок.

3. **Добавьте код, выделенный полужирным шрифтом:**

```
$('#hideShip').click(function() {
if ($(this).prop('checked')) {
}
}); // конец click
```

Простая управляющая инструкция (см. раздел «Управляющие инструкции» главы 3) облегчает проверку состояния флажка для решения относительно отображения или сокрытия элементов формы. Ключевое слово `$(this)` ссылается на объект, по которому производится щелчок — в данном случае на флажок. Атрибут элемента `checked` позволяет узнать, установлен или сброшен флажок. Если он установлен, то атрибут возвращает значение `true`, если сброшен — `false`. Вам остается лишь добавить код, обусловливающий сокрытие и отображение элементов формы.

4. **Добавьте выделенный полужирным шрифтом код (строки 16—18) в свой сценарий. Завершенный сценарий должен выглядеть следующим образом:**

```
1  <script>
2  $(document).ready(function() {
3     $(':text:first').focus();
4     $('#paypal').click(function() {
```

```
5     $('#creditCard input').prop('disabled', true)
6     .css('backgroundColor','#CCC);
7     $('#creditCard label').css('color','#BBB);
8  }); // конец click
9  $('#visa, #mastercard').click(function() {
10    $('#creditCard input').prop('disabled', false)
11          .css('backgroundColor','');
12    $('#creditCard label').css('color','');
13    }); // конец click
14    $('#hideShip').click(function() {
15          if ($(this).prop('checked')) {
16    $('#shipping').slideUp('fast');
17 } else {
18    $('#shipping').slideDown('fast');
19          }
20    }); // конец click
21 }); // конец ready()
22 </script>
```

Селектор `$('#shipping')` относится к элементу `div`, содержащему элементы формы, тогда как функции `slideUp()` и `slideDown()` (описанные в разделе «Скользящие элементы» главы 6) скрывают и показывают элемент `div` посредством его перемещения, соответственно, вверх за пределы поля зрения и вниз в поле зрения. Вы можете использовать и другие эффекты библиотеки jQuery, вроде `fadeIn()` и `fadeOut()`, или даже создать собственную анимацию, используя функцию `animate()` (см. раздел «Анимация» главы 6).

Окончательная версия руководства *готовый_08_01.html* находится в папке *глава08*. Если ваш код не работает, сравните его с выполненным руководством и обратитесь к разделу «Отслеживание ошибок» главы 1 для решения проблем.

Проверка формы

Бывает обидно, когда вы, просматривая информацию, собранную с помощью форм, на вашем сайте, обнаруживаете, что посетитель не

ввел свое имя, адрес электронной почты или другие важные данные. Вот почему в зависимости от типа формы, которую вы создаете, необходимо делать обязательным ввод определенных сведений.

Например, форма для подписки на рассылку новостей окажется бесполезной, если потенциальный получатель не введет своего электронного адреса. Аналогичная ситуация — с почтовым адресом при заказе книги или любого другого товара.

Кроме того, при получении данных, введенных в веб-форму, вы, возможно, захотите быть уверенными, что информация представлена в корректном формате, например, количество товара выражено в числах, а для сайта использован верный формат URL-адреса. Процедура проверки правильности ввода информации называется *проверкой* или *верификацией формы*.

С помощью языка JavaScript вы можете идентифицировать любую ошибку, прежде чем посетитель отошлет некорректные сведения.

По сути, проверка формы требует проверки данных в ее элементах на предмет наличия в них нужной информации в правильном формате. Проверка обычно происходит во время запуска события отправки формы, происходящее при нажатии посетителем кнопки подтверждения или клавиши **Enter**, когда текстовый курсор находится в текстовом поле. Если все в порядке, форма отправляется на веб-сервер. Если же возникает проблема, сценарий прерывает процесс передачи и выводит сообщение об ошибке — обычно рядом с некорректно заполненным полем (рис. 8.5).

Удостовериться, что в поле введен текст, довольно легко. Как вы узнали в разделе «Получение и установка значений элементов форм» данной главы, можно просто воспользоваться свойством формы `value` (с помощью, например, jQuery-функции `val()`) и, если значением окажется пустая строка, значит, поле является пустым. Но задача усложняется, если необходимо проверить другие типы элементов формы: флажки, переключатели и раскрывающиеся списки. Кроме того, вам придется написать сложный сценарий JavaScript, если вы захотите удостовериться, что посетитель ввел информацию определенного типа, например, электронный адрес, почтовый индекс, числа, даты и т. д. К счастью, вам не придется писать код самому. Всемирная паутина предоставляет большой выбор сценариев для проверки форм, и одним из лучших является плагин для библиотеки jQuery.

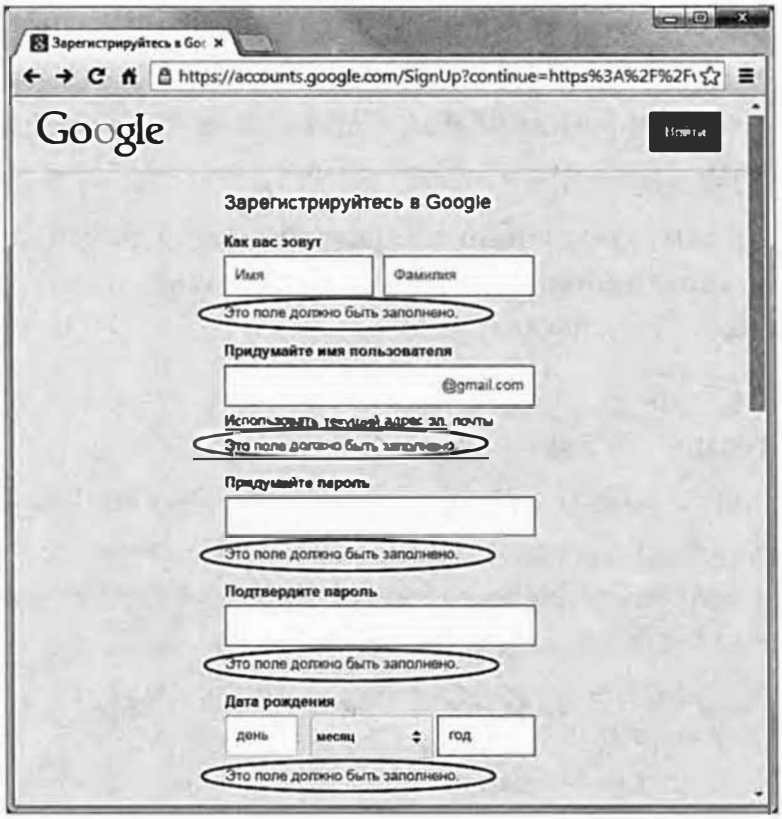

Рис. 8.5. Вы получите массу сообщений об ошибках (выделены на рисунке), если неправильно заполните элементы формы регистрации в системе Google

ПРИМЕЧАНИЕ

Версия HTML5 предусматривает множество функций, предназначенных для проверки форм, которые позволяют вам обойтись без JavaScript. К сожалению, несмотря на то, что многие последние браузеры поддерживают HTML-проверку форм, программа Internet Explorer 9 и более ранние версии ее не поддерживают, как и мобильные версии Safari для iOS и Android (на момент написания данной книги), кроме того, проблемы возникают и с настольной версией браузера Safari.

Плагин jQuery Validation

Validation (**jqueryvalidation.org**) — это мощный и вместе с тем простой в применении плагин библиотеки jQuery (автор Йорн Зеферер). Он может проверять форму на предмет заполнения всех нужных элементов и соответствия данных определенным требованиям. Например, поле ввода количества должно содержать число, а поле ввода адреса e-mail — электронный адрес. Если посетитель ввел данные в некорректной форме, плагин выдаст сообщение об ошибке с описанием проблемы.

Рассмотрим базовую схему процесса применения плагина Validation:

1. **Загрузите и присоедините файл *jquery.js* к странице, которая содержит форму, подлежащую проверке.**

   Прочтите раздел «Получение библиотеки jQuery» главы 4 для получения дополнительной информации по загрузке библиотеки jQuery. Плагин Validation использует именно библиотеку jQuery, чем и обусловлено присоединение файла *jquery.js*.

2. **Загрузите и присоедините плагин Validation.**

   Данный плагин можно найти по адресу: **jqueryvalidation.org**. Архив включает много дополнительного материала, в том числе демонстрации, тесты и др. Вам нужен только файл *jquery.validate.min.js* (плагин Validation вы также найдете в папке *jquery_validate* в каталоге *глава08*. Файл называется *jquery.validate.min.js*, см. руководство далее в главе). Это внешний файл JavaScript, так что просто следуйте инструкциям из раздела «Внешние файлы JavaScript» главы 1, чтобы прикрепить его к странице.

3. **Добавьте правила проверки.**

   Правила проверки — это просто инструкции, указывающие «сделать данный элемент формы обязательным для заполнения; убедиться, что данный элемент формы содержит адрес электронной почты и т. д.». Другими словами, на этом этапе вы определяете, какие элементы формы будут проверяться и каким образом. Существуют два способа добавления правил проверки: простой с использованием HTML (см. раздел «Простая проверка» далее) и более гибкий, но и несколько более сложный (см. раздел «Расширенная проверка»).

4. **Добавьте сообщения об ошибках.**

   Данный этап необязателен. Плагин Validation располагает встроенным набором сообщений об ошибках типа «Этот элемент формы обязателен для заполнения», «Введите допустимую дату», «Введите допустимый номер» и т. д. Подобные простые сообщения вполне уместны, но, тем не менее, вы можете настраивать их, чтобы сообщения предоставляли более ясные инструкции для заполнения каждого элемента формы, например, «Введите свое имя» или «Введите дату вашего рождения».

   Существуют два метода добавления сообщений об ошибках — простой, обсуждаемый в подразделе «Добавление сообщений об ошиб-

ках», и более гибкий, описываемый в подразделе «Расширенные сообщения об ошибках».

ПРИМЕЧАНИЕ

Вы также можете управлять стилем и расположением сообщений (см. раздел «Настройка стилей для сообщений об ошибках» данной главы).

5. Примените функцию `validate()` к форме.

Плагин включает функцию, которая творит все волшебство: `validate()`. Сначала с помощью библиотеки jQuery выберите форму, а затем примените к выделенной форме функцию. Например, ваша форма имеет идентификатор `signup`. Код HTML может выглядеть так:

```
<form action="process.php" method="post" ↵
name="signup" id="signup">
```

Простейший способ провести проверку:

```
$('#signup').validate();
```

Функция `validate()` принимает в качестве аргумента самую разную информацию, влияющую на работу плагина. Например, хотя вы можете определить правила проверки и сообщения об ошибках в HTML-коде формы (см. ниже), вы также можете задавать правила и сообщения при вызове функции `validate()` (дополнительно об этом методе вы узнаете в разделе «Расширенная проверка»).

Полный код JavaScript для самой простой проверки формы (включая уже описанные выше два этапа) можно представить в следующем виде:

```
<script src="js/jquery.min.js"></script>
<script src="js/jquery.validate.min.js"></script>
<script>
$(document).ready(function() {
$('#signup').validate();
}); // конец ready
</script>
```

ПРИМЕЧАНИЕ

Не забывайте всегда помещать ваш сценарий в пределах jQuery-функции `document.ready()`, чтобы обеспечить его выполнение после загрузки HTML-кода страницы (см. раздел «Больше концепций для событий jQuery» главы 5).

Простая проверка

Чтобы использовать плагин Validation, достаточно присоединить JavaScript-файл плагина, добавить нескольких атрибутов `class` и `title` к элементам формы, подлежащим проверке и применить к форме метод `validate()`. Метод `validate()` является простейшим способом проверки. Если же вам нужно управлять положением сообщений об ошибках, применить к элементу формы более одного правила или задать минимум/максимум знаков для текстового поля, то придется воспользоваться расширенным методом, описанным в разделе «Расширенная проверка».

Для добавления плагина проверки выполните шаги, описанные в предыдущих разделах (присоединение файла jQuery и плагина Validation и т. д.). Кроме того, вы можете внедрить правила и сообщения об ошибках в HTML-код элемента вашей формы.

Создание правил проверки

Простейшим способом проверки с помощью плагина Validation является присвоение элементу формы одного или нескольких имен класса, перечисленных в табл. 8.2. Плагин проверяет имена класса каждого элемента формы на предмет наличия условий проверки. Если они есть, к элементу формы будут применены соответствующие правила проверки.

Например, у вас есть текстовое поле для ввода имени. Базовый HTML-код может иметь следующий вид:

```
<input name="name" type="text">
```

Чтобы дать плагину понять, что данное поле обязательно для заполнения, другими словами, то, что форма не может быть отправлена, если это поле останется незаполненным, добавьте HTML-элементу класс `required`. Например, чтобы сделать текстовое поле обязательным для заполнения, добавьте атрибут класса в HTML-элемент:

```
<input name="name" type="text" class="required">
```

Табл. 8.2. Плагин Validation включает методы, отвечающие наиболее распространенным требованиям проверки

Правило проверки	Описание
`required`	Элемент формы обязателен для заполнения, пометки или выбора
`date`	Данные должны иметь формат ММ/ДД/ГГГГ. Например, дата 10/30/2014 считается корректной, а 10-30-2014 — нет
`url`	Веб-адрес должен быть записан в полной форме типа http://www.chia-vet.com. Частичный URL-адрес, например, www.chia-vet.com или chia-vet.com (http://chia-vet. com), является недействительным
`email`	Электронный адрес должен иметь формат bob@chia-vet.com. Этот класс не проверяет действительность введенного адреса, так что даже такой адрес, как nobody@noplace.com, успешно пройдет проверку
`number`	Данные должны быть числом типа 32 или 102.50 или даже —145.5555. Однако ввод символов, например, $45.00 или 100,000, недопустим
`digits`	Данные должны быть положительными целыми числами. Числа 1, 20, 12333 являются корректными, а 10.33 или —12 — нет
`creditcard`	Должен быть введен корректный номер кредитной карты

Добавление класса таким способом в действительности не имеет ничего общего с CSS, несмотря на то, что вы обычно присваиваете класс HTML-элементу для обеспечения способа его форматирования путем создания стиля класса CSS. В данном случае вы используете имя класса для снабжения плагина информацией, необходимой для определения того, какой тип проверки вы применяете к данному элементу формы.

ПРИМЕЧАНИЕ

Проверка с помощью языка JavaScript — это отличный способ предоставления дружественной обратной связи тем посетителям, которые ошибочно пропустили элемент формы или ввели неверный тип информации. Однако этот способ не является оптимальным для предотвращения злонамеренного ввода данных. Проверку с помощью JavaScript легко обойти, так что если вы хотите быть уверены, что не получите от посетителей «плохих» данных, вам нужно будет также произвести проверку на стороне сервера.

Требование от посетителя заполнить элемент формы является, вероятно, самой распространенной задачей проверки, но нередко возникает необходимость удостовериться в том, что введенные данные соответствуют заданному формату. Например, спрашивая, сколько виджетов

необходимо посетителю, вы ожидаете получить числовое значение. Чтобы элемент формы было обязательным для заполнения и содержал определенный тип значения, следует добавить класс `required`, а также один из классов, приведенных в табл. 8.2.

Например, существует поле для ввода даты рождения. Эта информация является не только обязательной, но и должна вводиться в специальном формате. HTML-код для подобного элемента формы может иметь такой вид:

```
<input name="dob" type="text" class="required date">
```

Заметьте, что имена классов — `required` и `date` — разделены пробелами.

Если вы исключите класс `required` и примените только тип проверки данных, например, `class="date"`, тогда элемент формы окажется необязательным для заполнения, но, если информация в него все-таки вносится, она должна отвечать заданному формату.

СОВЕТ

Когда вы определяете особый формат для данных элемента формы, позаботьтесь об инструкциях для посетителя, поясняющих ему правила ввода информации. Если, например, в элемент формы вводится дата, добавьте рядом с ним сообщение типа «Введите дату в формате ММ/ДД/ГГГГ, например, 01/25/2015».

Добавление сообщений об ошибках

Плагин Validation располагает набором универсальных сообщений об ошибках. Например, если обязательный для заполнения элемент формы не содержит данных, появляется сообщение «Этот элемент формы является обязательным для заполнения». Если в элемент формы необходимо ввести дату, появится сообщение «Введите правильную дату». Однако вы имеете возможность модифицировать эти сообщения и создавать собственные.

Простейший путь — добавить элементу формы атрибут `title` и задать текст сообщения в качестве значения атрибута. Предположим, с помощью класса `required` вы делаете элемент формы обязательным для заполнения:

```
<input name="name" type="text" class="required">
```

Для формирования текста сообщения об ошибке просто добавьте атрибут `title`:

```
<input name="name" type="text" class="required" ↵
title="Пожалуйста, введите ваше имя.">
```

Обычно веб-дизайнеры используют атрибут `title` для удобства пользователя, предоставляя конкретные инструкции, которые появляются при наведении указателя мыши на какой-либо элемент формы, или для программ, позволяющих озвучивать текст на экране. Но в случае с плагином Validation этот атрибут служит для создания сообщений, которые вы хотите вывести на экран. Плагин сканирует все проверяемые элементы формы в поиске атрибута `title`. Найдя его, плагин использует значение атрибута в качестве текста сообщения об ошибке.

При использовании нескольких методов проверки сообщение должно быть адекватным для всех ситуаций. Например, если у вас есть обязательный для заполнения элемент формы, который должен содержать дату, сообщение типа «Этот элемент формы обязателен для заполнения» бессмысленно для ситуации, когда введена дата в неверном формате. Вот пример сообщения, отвечающего как ситуации, когда элемент формы оставлен пустым, так и ошибочному вводу даты:

```
<input name="dob" type="text" class="required date" ↵
title="Пожалуйста, введите дату в формате 01/28/2014.">
```

Итак, задавание правил проверки и сообщений об ошибках посредством добавления имен классов и надписей просто и эффективно. Но иногда вам может понадобиться более сложная проверка. В данном случае плагин Validation предлагает дополнительный метод. Например, вы хотите предусмотреть разные сообщения об ошибках в зависимости от их типа: одно — для пустого элемента формы и другое — для неверного формата введенных данных. С помощью стандартного метода проверки, описанного ранее, решить такую задачу невозможно. К счастью, плагин Validation предусматривает более расширенный метод проверки, который позволяет применять более широкий диапазон правил проверки.

Например, расширенный метод необходим, чтобы задать минимальное количество знаков при вводе данных. Устанавливая пароль, вы можете ввести требование его минимального размера в шесть знаков.

Расширенная проверка

Плагин Validation предоставляет еще один способ проверки формы, не требующий изменения HTML-кода элементов формы. Кроме того, су-

ществует большое количество дополнительных настроек для управления работой плагина. Они задаются путем передачи функции `validate()` объектной константы (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), содержащей отдельные объекты для разных настроек. Например, для указания правила проверки передается объект, содержащий код для этого правила. Сначала введите открывающую фигурную скобку прямо после первой круглой скобки функции проверки и затем закрывающую — непосредственно перед последней круглой скобкой:

```
$('idOfForm').validate({
// сюда помещаются настройки
}); // конец validate();
```

Эти фигурные скобки представляют литерал объекта, который будет содержать настройки. Такое применение плагина Validation кажется несколько запутанным, поэтому лучше всего рассмотреть его на примере, который приведен на рис. 8.6.

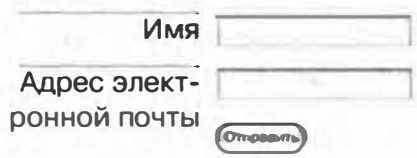

Рис. 8.6. Даже к такой простой форме вы можете применить расширенные настройки плагина Validation для дополнительного контроля

СОВЕТ

Допустимо применить оба метода к одной и той же форме. К элементам формы, для которых предусмотрено только одно правило проверки и одно сообщение об ошибке, можно применить стандартный метод, а расширенный использовать только при более сложной проверке. Например, руководство в конце данной главы использует оба метода для проверки одной формы.

HTML-код для формы на рис. 8.6 имеет следующий вид:

```
<form action="process.php" method="post" id="signup">
<div>
<label for="name">Имя</label>
<input name="name" type="text">
</div>
```

```
<div>
<label for="email">Адрес электронной почты</label>
<input name="email" type="text">
</div>
<div>
<input type="submit" name="submit" value="Отправить">
</div>
</form>
```

Рассматриваемая форма содержит два текстовых поля: для ввода имени и адреса электронной почты. В данном разделе будет описываться процесс расширенной проверки этих элементов формы на предмет заполнения первого поля и заполнения и правильного формата данных для второго поля.

ПРИМЕЧАНИЕ

Полный список настроек плагина Validation можно найти по адресу: **jqueryvalidation.org/validate**.

Расширенные правила

Расширенный способ задавания правил проверки подразумевает передачу объекта, содержащего имена элементов формы, и правила проверки, которые вы хотите применить к элементу формы. Базовая структура этого объекта выглядит так:

```
rules: {
fieldname : 'validationType'
}
```

Объект называется `rules`, и внутри него вы определяете элементы формы и типы проверки. Затем объект передается функции `validate()`. В случае с формой на рис. 8.6 для определения элемента формы в качестве обязательного для заполнения функция `validate()` применяется к форме, как описывалось ранее, а затем функции передается объект `rules`:

```
$('#signup').validate({
rules: {
```

```
name: 'required'
}
}); // конец validate()
```

В данном случае элемент формы называется `name`, а правило указывает на его обязательное заполнение. Для применения нескольких правил к этому элементу формы вы должны создать для него другой объект. Например, для расширения правил проверки для формы на рис. 8.6 вы можете добавить правило, требующее не только ввода электронного адреса, но и соблюдения корректного формата ввода:

```
1 $('#signup').validate({
2     rules: {
3           name: 'required',
4     email: {
5           required:true,
6           email:true
7     }
8           }
9 }); // конец validate()
```

ПРИМЕЧАНИЕ

Согласно правилам литералов объектов (объектных констант) JavaScript, вы должны завершать каждую пару имя/значение, за исключением последней, запятой. Например, в строке 3 после фрагмента `name: 'required'` должна быть запятая, поскольку далее следует другое правило (для поля электронного адреса). За информацией о работе литералов объектов обратитесь к разделу «Одновременное изменение нескольких свойств CSS» главы 4.

Строки 4—7, выделенные полужирным шрифтом, задают правила для поля, содержащего электронный адрес. Поле называется `email`, как указано в HTML-коде (см. HTML-код в начале данного раздела); фрагмент `required:true` делает данный элемент формы обязательным для заполнения, а код `email:true` требует наличия в поле электронного адреса.

Вы можете использовать любой из типов проверки, приведенных в табл. 8.2. Например, вы добавляете в форму элемент `birthdate`. Для гарантии ввода данных в него можно расширить список правил:

```
$('#signup').validate({
rules: {
name: 'required',
email: {
required:true,
email:true
},
birthdate: 'date'
}
}); // конец validate()
```

Если вы хотите сделать заполнение элемента формы обязательным, настройте код таким образом:

```
$('#signup').validate({
rules: {
name: 'required',
email: {
required:true,
email:true
},
birthdate: {
date:true,
required:true
}
}
}); // конец validate()
```

Как говорилось ранее, одной из наиболее полезных вещей, реализуемых с помощью расширенных правил проверки, является настройка минимального и максимального размеров записи. Например, в бланке рекламации вы можете установить лимит для замечаний, скажем, в 200 знаков, так что клиентам придется высказываться по существу, а не писать «Войну и мир». Существуют также правила, определяющие, что вводимые числовые значения должны принадлежать некоторому диапазону; например, если вы не собираетесь принимать данные от мумий или вампиров, то не станете принимать даты рождения ранее 1900 г.

- **minlength.** Поле должно содержать, *по крайней мере*, определенное число знаков. Например, правило, задающее ввод в поле минимум шести знаков имеет следующий вид:

  ```
  minlength:6
  ```

- **maxlength.** Поле должно содержать *не более* определенного числа знаков. Например, правило, задающее ввод в поле не более 100 знаков, записывается так:

  ```
  maxlength:100
  ```

- **rangelength.** Правило, сочетающее `minlength` и `maxlength`. Устанавливает минимальное и максимальное число знаков, разрешенных при вводе данных. Например, правило, задающее ввод в поле минимум шести знаков, но не более ста:

  ```
  rangelength:[6,100]
  ```

- **min.** Поле должно содержать число, равное или большее указанного числа. Например, следующее правило требует, чтобы поле содержало число, и чтобы это число было равным или большим 10.

  ```
  min:10
  ```

В данном примере, если посетитель введет 8, поле не пройдет проверку, поскольку 8 меньше 10. Аналогично при вводе слова «восемь» поле также не пройдет проверки и появится сообщение об ошибке.

- **max.** Задает максимальное число, которое может содержать поле. Например, для гарантии того, что поле содержит число, меньшее 1000, используется следующий код:

  ```
  max:1000
  ```

- **range.** Правило, сочетающее `min` и `max` и задающее диапазон для чисел, вводимых в поле. Например, чтобы обеспечить ввод чисел, не меньших 10, но и не больших 1000, используйте следующий код:

  ```
  range:[10,1000]
  ```

- **equalTo.** Требует, чтобы содержимое поля совпадало с содержимым другого поля. Например, при вводе пароля посетителя часто просят повторить его. Этим проверяется правильность ввода пароля в первый раз. Чтобы применить данный метод, вы должны сначала задать строку, содержащую правильный селектор jQuery. Например, первое поле пароля имеет идентификатор `password`. Если хотите обеспечить соответствие содержимого поля «Подтвердите пароль» содержимому первого поля, примените код:

  ```
  equalTo: '#password'
  ```

Расширенные правила проверки можно применять в сочетании друг с другом. Вот пример такого сочетания. Допустим, ваша форма включает два поля, одно из которых служит для ввода пароля, а второе — для его подтверждения. HTML-код для этих двух полей выглядит так:

```
<input name="password" type="password" id="password">
<input name="confirm_password" type="password" ↵
id="confirm_password">
```

ПРИМЕЧАНИЕ

Плагин jQuery Validation предусматривает второй плагин *additional-method.js*, предусматривающий дополнительные правила проверки, включая правила, определяющие минимальное количество слов, идентификационные номера автомобилей США, номера счетов банка Dutch Bank и другие менее конкретные, но потенциально полезные, правила. Для этих дополнительных правил не существует документации, поэтому вам придется обратиться к коду в файле *additional-method.js*. При обнаружении понравившихся правил вам следует сохранить копию файла, например, под именем *my-rules.js*, и удалить все ненужные правила. В результате этого размер вашего файла JavaScript будет гораздо меньше.

Оба поля обязательны для заполнения, а пароль должен содержать от 8 до 16 символов. Наконец, вам нужно гарантировать, что поле «Подтвердите пароль» совпадает с первым полем для ввода пароля. Допустим, форма имеет идентификатор `signup`, тогда вы можете верифицировать эти поля с помощью следующего кода:

```
$('#signup').validate({
 rules: {
 password: {
 required:true,
 rangelength:[8,16]
},
confirm_password: {
equalTo:'#password'
}
}
}); // конец validate()
```

Расширенные сообщения об ошибках

Как вы узнали ранее, сообщение об ошибке можно легко добавлять с помощью атрибута `title`, содержащего его текст. Однако данный способ не позволяет создавать отдельные сообщения для каждого типа ошибки проверки. Скажем, поле является обязательным для заполнения и должно содержать число. Вам нужно иметь два разных сообщения для каждого типа ошибки: «Это поле обязательно для заполнения» и «Введите число». Вы не можете сделать это с помощью атрибута `title`. Вместо этого вы должны передать функции `validate()` объект JavaScript, содержащий различные сообщения об ошибках, которые хотите отобразить.

Процесс схож с созданием расширенных правил, описанным в предыдущем разделе. Базовая структура объекта `messages` такова:

```
Messages: {
Fieldname: {
methodType: 'Сообщение об ошибке'
}
}
```

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Проверка с помощью сервера

Хотя проверка средствами JavaScript удобна для быстрой проверки введенной пользователем информации, иногда необходимо обратиться к серверу за проверкой корректности заполнения поля. Например, у вас есть форма регистрации, позволяющая посетителю создать имя пользователя для участия в форумах на вашем сайте. Два человека не могут иметь одинаковое имя пользователя, поэтому представляется целесообразным сообщить посетителю о том, что введенное имя уже используется, еще до того момента, когда он попытается отправить заполненную форму. В этом случае вам придется обратиться к серверу и узнать о доступности задаваемого имени.

Плагин Validation располагает методом `remote`, позволяющим выполнить такую проверку. Данный метод дает возможность передавать имя, введенное посетителем в поле, серверу, использующему серверный язык, например, PHP, Ruby, .NET, Java или Node.js. Сервер принимает эту информацию и совершает некоторое действие, например, проверяет доступность имени пользователя, а затем дает

ответ форме в виде значений `true` (проверка пройдена) или `false` (проверка не пройдена).

Предположим, у вас есть поле `"username"`, которое является обязательным для заполнения и не должно содержать имя, уже используемое на сайте. Для создания соответствующего правила для этого поля (с использованием расширенных правил, описанных ранее) вы можете добавить к объекту `rules` следующий код:

```
username : {
required: true,
remote: 'check_username.php'
}
```

Удаленный метод принимает строку, содержащую путь от текущей до серверной страницы. В нашем примере страница называется *check_username.php*. Когда плагин пытается проверить данные в поле, он посылает введенное имя пользователя (`username`) и данные посетителя на страницу *check_username.php*, которая определяет, можно ли использовать это имя. Если да, то PHP-страница возвращает слово `'true'`; если же имя уже задействовано, возвращается слово `'false'`, и поле не проходит проверку.

Все это волшебство осуществимо благодаря технологии Ajax, с которой вы познакомитесь в части 4. Рабочий пример данного метода можно найти на странице: **jquery.bassistance.de/validate/demo/captcha/**.

В предыдущем примере замените часть `fieldname` именем элемента формы, которое вы проверяете, а `methodType` — одним из методов проверки. Например, чтобы совместить методы проверки для полей ввода пароля и сообщения для каждого типа ошибок, добавьте код, выделенный полужирным шрифтом:

```
$('#signup').validate({
rules: {
password: {
required:true,
rangelength:[8,16]
},
confirm_password: {
equalTo:'#password'
}
```

```
}, // конец rules
messages: {
password: {
required: "Пожалуйста, введите пароль.",
rangelength: "Ваш пароль должен содержать ↵
от 8 до 16 символов."
},
confirm password: {
equalTo: "Введенные пароли не совпадают."
}
} // конец messages
}); // конец validate()
```

СОВЕТ

Как видите, использование расширенного метода может потребовать множества объектных констант, и масса фигурных скобок может затруднить понимание кода. Хорошей практикой при использовании расширенного метода проверки является отсутствие спешки и частое тестирование. Вместо того чтобы вводить все правила и сообщения разом, добавьте одно правило, а затем проверьте страницу. Если проверка не сработала, вы, вероятно, допустили ошибку в коде, поэтому, прежде чем продолжать добавление правил, исправьте ее. После того как вы закончили с правилами и убедились, что они работают, добавьте объектную константу для сообщения об ошибке. Опять же, не торопитесь, добавляйте сообщения по одному и часто проверяйте свою работу. Не забывайте использовать консоль браузера для обнаружения ошибок JavaScript.

Стилизация сообщений об ошибках

Когда плагин Validation проверяет форму и находит некорректное значение в элементе, происходит следующее: сначала элементу формы добавляется класс, а потом HTML-элемент `label`, содержащий сообщение об ошибке. Например, ваша страница имеет такой HTML-код для поля ввода адреса электронной почты:

```
<input name="email" type="text" class="required">
```

Если вы добавите плагин Validation на страницу с этой формой, и ваш посетитель попытается отправить ее, не заполнив поле для адреса элек-

тронной почты, плагин прервет процесс передачи и изменит HTML-код поля, добавив в него дополнительный HTML-элемент. Новый HTML-код будет иметь следующий вид:

```
<input name="email" type="text" class="required error">
<label for="email" generated="true" class="error"> ↵
Это поле обязательно для заполнения.</label>
```

Другими словами, плагин присвоил элементу формы имя класса `error`, а также вставил HTML-элемент `label` с классом `error`, содержащий текст сообщения об ошибке.

Для изменения вида сообщений об ошибках вам просто необходимо добавить нужный стиль в вашу таблицу стилей. Например, чтобы текст сообщения был выделен жирным и красным шрифтом, добавьте этот стиль в таблицу стилей:

```
label.error {
color: #F00;
font-weight: bold;
}
```

Поскольку плагин Validation также добавляет некорректно заполненному элементу формы класс ошибки, вы можете создать стили CSS и для их форматирования. Например, чтобы заключить неверно заполненный элемент формы в красную рамку, создайте следующий стиль:

```
input.error, select.error {
border: 1px red solid;
}
```

Проверка формы на практике

В этом руководстве вы примените к форме стандартные и дополнительные настройки проверки (рис. 8.7).

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

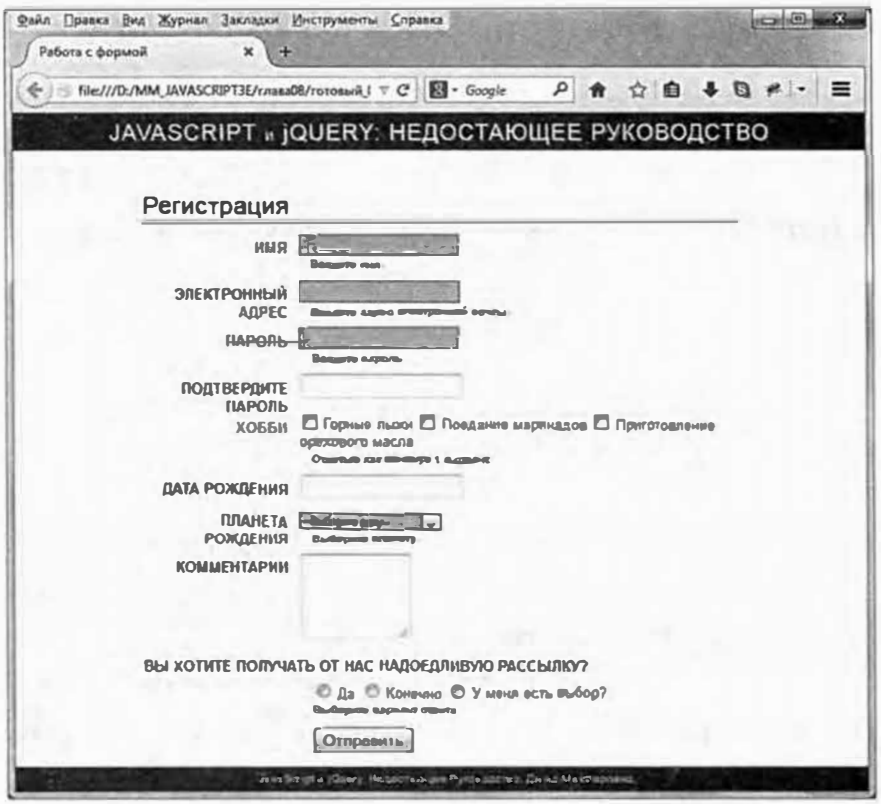

Рис. 8.7. Не допускайте некорректного заполнения форм! Плагин Validation поможет вам быть уверенными, что вы получаете нужную информацию

Простая проверка на практике

Начнем со стандартных методов проверки, описанных в разделе «Простая проверка». Затем изучим расширенные методы, которые обсуждаются в разделе «Расширенная проверка».

Как вы увидите, сочетания этих двух подходов для одной и той же формы дают прекрасные результаты.

1. **В текстовом редакторе откройте файл *08_02.html* в каталоге *глава08*.**

   Данный файл содержит форму с текстовыми полями, флажками, переключателями и раскрывающимися списками. Вы примените к этой форме процедуру проверки, но сначала необходимо прикрепить к веб-странице плагин Validation.

2. **В пустой строке непосредственно после элемента `script`, присоединяющего файл *jquery.js* к этой странице, введите:**

```
<script src="jquery_validate/jquery.validate. ↵
min.js"></script>
```

Плагин Validation находится в папке *jquery_validate* в каталоге *глава08*.

Эта страница уже имеет элемент `script` с функцией jQuery `ready()`. Вам нужно лишь добавить к форме функцию `validate()`.

3. **В пустой строке прямо под фрагментом `$(document).ready(function()` введите:**

```
$('#signup').validate();
```

Форма имеет идентификатор `signup`:

```
<form action="process.php" method="post" ↵
name="signup" id="signup">
```

Таким образом, код `$('#signup')` использует библиотеку jQuery для выбора этой формы, а функция `validate()` применяет плагин для ее проверки. Однако проверка невозможна, пока вы не зададите правила. Прежде всего, вы должны сделать поле ввода имени обязательным для заполнения и настроить сообщение об ошибке.

4. **Найдите HTML-код для поля ввода имени (`<input name="name" type="text" id="name">`) и добавьте атрибуты `class` и `title` так, чтобы тег выглядел следующим образом (изменения выделены полужирным шрифтом):**

```
<input name="name" type="text" id="name"
class="required" title="Введите свое имя.">
```

Часть `class="required"` кода сообщает плагину Validation о том, что поле является обязательным для заполнения, тогда как атрибут `title` определяет сообщение, которое увидит посетитель, если не заполнит это поле.

5. **Сохраните страницу, откройте ее в браузере и нажмите кнопку «Отправить».**

Поскольку имя в поле не введено, справа появляется сообщение (обведенное на рис. 8.8).

Поздравляем! Вы добавили правило проверки в форму с помощью базового метода (см. раздел «Простая проверка»). Теперь дополним проверку правилом для поля «Дата рождения».

ПРИМЕЧАНИЕ

Если вы не видите сообщения об ошибке, а вместо этого переходите на страницу, подтверждающую отправку формы, значит, проверка не проведена, и форма отослана в том виде, в котором составлена.

Проверьте шаги 1—4, чтобы убедиться, что вы не допустили опечаток.

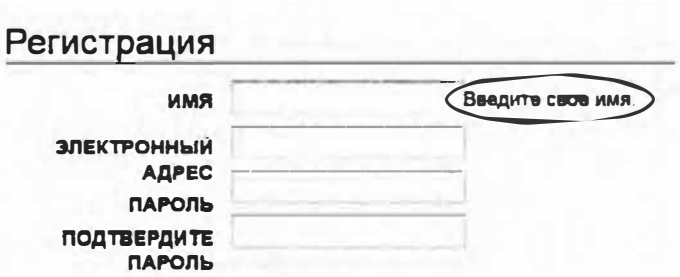

Рис. 8.8. Пока не беспокойтесь по поводу внешнего вида сообщения об ошибке. Вы научитесь форматировать его чуть позже

6. **Найдите HTML-код для поля «Дата рождения» (`<input name="dob" type="text" id="dob">`) и добавьте атрибуты `class` и `title` так, чтобы тег выглядел следующим образом:**

```
<input name="dob" type="text" id="dob" class="date" ↵
title="Введите дату своего рождения в формате:↵
01/19/2000">
```

Поскольку вы не добавили класс `required`, заполнение данного поля не является обязательным. Однако если посетитель ввел в него данные, атрибут `class="date"` сообщает плагину необходимость соблюдения формата даты. С помощью атрибута `title` назначается сообщение для случая, когда поле заполнено неправильно. Сохраните страницу и проверьте ее в браузере — введите в поле что-нибудь типа `kjsdf` и попытайтесь отправить данные формы.

ПРИМЕЧАНИЕ

Если вы хотите обязать посетителя заполнить поле «Дата рождения» и сделать это корректно, просто добавьте код `required` к атрибуту класса. При этом фрагменты `date` и `required` должны разделяться пробелом:

```
class="date required"
```

Вы можете применить тот же метод для проверки раскрывающегося списка (элемент `select`).

7. **Найдите HTML-код для открывающего тега `<select>` (`<select name="planet" id="planet">`) и добавьте атрибуты `class` и `title` так, чтобы тег имел следующий вид:**

```
<select name="planet" id="planet" class:"required" ↵
title="Выберите планету.">
```

Верифицировать раскрывающийся список можно подобно текстовым полям, добавляя атрибуты `class` и `title`.

А теперь пора попробовать расширенный метод проверки.

Расширенная проверка на практике

Как указывалось в разделе «Расширенная проверка», есть вещи, которые невозможно выполнить только стандартными методами проверки (назначить разные сообщения для разных проблем или задать определенное количество знаков для вводимой информации). В этих случаях необходимо использовать дополнительные возможности плагина Validation.

Для начала добавим два правила проверки и два разных сообщения об ошибках для поля ввода электронного адреса.

1. **В коде JavaScript в верхней части файла найдите строку `$('#signup').validate();` и отредактируйте ее следующим образом:**

   ```
   $('#signup').validate({

   }); // конец validate()
   ```

   Другими словами, добавьте открывающую и закрывающую фигурные скобки между круглыми скобками функции `validate()`, вставьте между ними пустую строку и напечатайте комментарий JavaScript для определения окончания функции `validate()`. Вы вскоре заполните сценарий разными скобками и станет нелегко вспомнить, какая скобка сочетается с какой. Поэтому введенный комментарий поможет сориентироваться, хотя, как и все комментарии, не является обязательным.

   Теперь создадим базовую структуру для добавления правил проверки.

2. **В пустой строке (между скобками), которую вы только что добавили, введите:**

   ```
   rules: {

   } //конец rules
   ```

Для облегчения чтения кода вам могут понадобиться два пробела перед `rules` и `}`. Отступ в этих строках подчеркнет их принадлежность функции `validate()`.

Данный код создает пустой объект, который заполняется именами элементов формы и методами проверки. Кроме того, комментарий JavaScript определяет окончание объекта. Далее вы добавите правила для поля ввода адреса электронной почты.

3. **Отредактируйте функцию `validate()` таким образом, чтобы она имела следующий вид (изменения выделены полужирным шрифтом):**

```
$('#signup').validate({
rules: {
email: {
required: true,
email: true
}
} // конец rules
}); // конец validate()
```

Здесь вы добавили еще одну объектную константу JavaScript. Первая часть `email` является именем элемента формы, подлежащего проверке, и совпадает с его именем в HTML-коде. Далее указаны два метода проверки — обязательность заполнения поля (то есть пользователь должен заполнить форму, если он собирается ее отправить) и соответствие вводимой информации формату электронного адреса. Хорошим девизом для любого программиста является: «Проверяй как можно чаще». Прежде чем двигаться дальше, удостоверьтесь, что сценарий работает.

4. **Сохраните страницу, откройте ее в браузере и нажмите кнопку «Отправить».**

Вы увидите сообщение об отсутствующей информации, предусмотренное плагином по умолчанию: «This field is required» («Это поле обязательно для заполнения»). Введите в поле пару символов. Сообщение изменится на «Please enter a valid email address» («Введите корректный адрес электронной почты») (это стандартное сообщение, отображаемое плагином, когда посетитель ошибается при вводе

электронного адреса в поле). Если вы не видите никаких сообщений, проверьте свой код и сравните его с шагом 3 выше.

Теперь вы добавите специальные сообщения об ошибках для данного поля.

5. **Вернитесь в текстовый редактор. Поставьте запятую после закрывающей фигурной скобки для объекта `rules` (но перед комментарием `// конец rules`) и напишите:**

```
messages: {

} // конец messages
```

Этот код представляет еще один объект JavaScript, называемый `messages`. Он будет содержать любые сообщения, которые вы пожелаете добавить для элементов формы. Комментарий в конце (`// конец messages`) не является обязательным. Теперь добавим сообщение для поля электронного адреса.

6. **Отредактируйте функцию `validate()` таким образом, чтобы она имела следующий вид (добавления выделены полужирным шрифтом):**

```
1  $('#signup').validate({
2  rules: {
3     email: {
4           required: true,
5           email: true
6     }
7  }, // конец rules
8  messages: {
9  email: {
10    required: "Введите адрес электронной почты.",
11    email: "Это некорректный адрес электронной почты."
12    }
13          } // конец messages
14 }); // конец validate(),
```

Сохраните страницу и просмотрите ее в браузере. Попытайтесь отправить форму, не заполнив поле электронного адреса. Вы должны получить сообщение «Введите адрес электронной почты». Теперь

введите в поле что-нибудь типа «привет» и вновь повторите попытку. На этот раз вы должны получить сообщение «Это некорректный адрес электронной почты».

Если же вы получили сообщение о том, что форма передана, значит, где-то в вашем коде есть ошибка. Наиболее часто причиной такой ситуации является пропущенная запятая после объекта `rules` (см. строку 7) или после объекта сообщения `email` (см. строку 10).

А теперь пора добавить правила проверки для двух полей с паролем.

7. **Отредактируйте объект `rules` таким образом, чтобы он имел следующий вид (изменения выделены полужирным шрифтом):**

```
1 rules: {
2  email: {
3     required: true,
4     email: true
5  },
6  password: {
7     required: true,
8     rangelength:[8,16]
9  },
10 confirm_password: {
11    equalTo:'#password'
12 }
13 }, //конец rules
```

Не пропустите запятую в строке 5, она необходима для отделения правил ввода электронного адреса от правил ввода пароля.

Первый набор правил применяется к первому полю с паролем. Оно задает обязательность заполнения данного элемента формы и требует для пароля не менее 8, но не более 16 знаков. Второе правило применяется к полю подтверждения пароля и требует, чтобы его содержимое совпадало с содержимым первого поля (детали о работе этих правил описаны в разделе «Расширенная правила» данной главы).

СОВЕТ

Рекомендуется сохранять и тестировать файл после каждого этапа выполнения данного руководства. Это поможет вам отследить, на каком этапе вы допустили ошибку.

Данные правила также должны сопровождаться сообщениями об ошибках.

8. Отредактируйте объект `messages` таким образом, чтобы он имел следующий вид (изменения выделены полужирным шрифтом):

```
1  messages: {
2     email: {
3     required: "Введите адрес электронной почты.",
4     email: "Это некорректный адрес электронной почты."
5     },
6  password: {
7  required: 'Введите пароль',
8  rangelength: 'Пароль должен содержать ↵
      от 8 до 16 символов.'
9  },
10 confirm_password: {
11    equalTo: 'Пароли не совпадают.'
12 }
13 } // конец messages
```

Не забудьте про запятую в строке 5.

На данном этапе вы уже должны чувствовать себя уверенно, добавляя правила и сообщения об ошибках. Далее вы займетесь проверкой флажков и переключателей.

Проверка состояний флажков и переключателей

Флажки и переключатели обычно сгруппированы, поэтому настройка правил проверки для нескольких из них в составе группы является непростым процессом поиска всех флажков или переключателей в группе. К счастью, плагин Validation позаботился и об этом, он облегчает проверку подобных элементов формы.

1. **Найдите HTML-код для первого флажка (`<input name="hobby" type="checkbox" id="heliskiing" value="heliskiing">`) и добавьте атрибуты `class` и `title` так, чтобы тег выглядел следующим образом (изменения выделены полужирным шрифтом):**

```
<input name="hobby" type="checkbox" id="heliskiing" ↵
value="heliskiing" class="required" ↵
title="Отметьте как минимум 1 вариант.">
```

   Здесь вы пользуетесь стандартным методом проверки, описанным в разделе «Простая проверка» ранее в главе. Вы могли бы применить и расширенный метод, задав правила и сообщения как часть функции `validate()`, но, если вам нужно только одно правило и одно сообщение, лучше обратиться к базовому методу как к более простому и менее подверженному возникновению ошибок.

   В данном случае все флажки имеют одинаковое имя, так что плагин Validation обрабатывает их как группу. Другими словами, это правило проверки применяется *ко всем трем* флажкам, хотя вы и добавили атрибуты `class` и `title` только для одного флажка. В сущности, вы обязали посетителя установить как минимум один флажок, прежде чем он сможет отправить форму.

   То же самое сделаем с переключателями в нижней части формы.

2. **Найдите HTML-код для первого переключателя (`<input type="radio" name="spam" id="yes" value="yes">`) и добавьте атрибуты `class` и `title` так, чтобы тег выглядел следующим образом (изменения выделены полужирным шрифтом):**

```
<input type="radio" name="spam" id="yes" value="yes"↵
class="required" title="Выберите вариант ответа">
```

   Группа связанных переключателей имеет единое имя (в нашем случае `spam`), так что, хотя вы и добавили атрибуты `class` и `title` только одному переключателю, они применяются ко всем трем. Поскольку элемент формы обязателен для заполнения, посетитель обязан установить переключатель в одно из положений, прежде чем он сможет отправить форму.

3. **Сохраните файл, проверьте его в браузере и нажмите кнопку «Отправить».**

   Кое-что вам может показаться не совсем логичным. Сообщения для флажков и переключателей появляются возле первого флажка или первого переключателя (выделены на рис. 8.9). Хуже того, они ото-

бражаются между флажком или переключателем и его названием (например, между элементом формы и названием «Горные лыжи»).

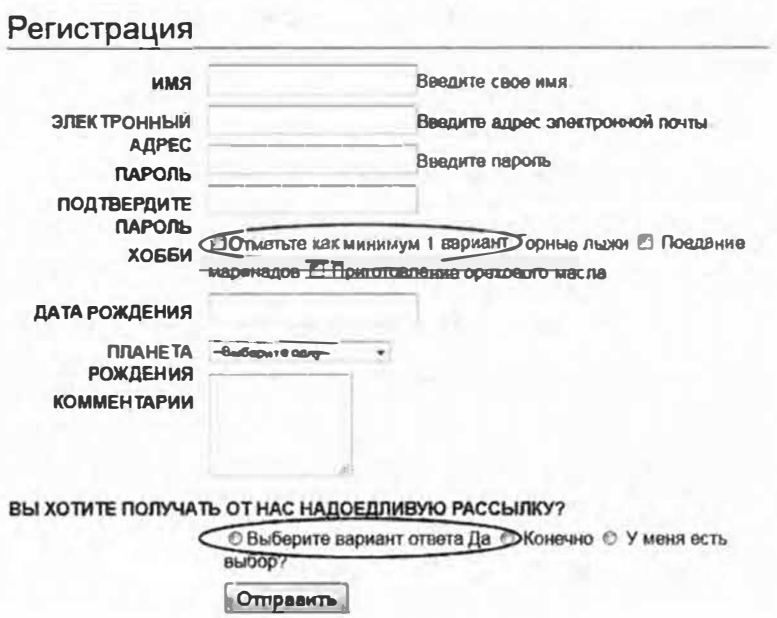

Рис. 8.9. Плагин Validation помещает сообщения об ошибках рядом с элементом формы, вызвавшим проблему. В случае с флажками и переключателями это выглядит ужасно. Для изменения положения сообщений вы должны предоставить функции `validate()` соответствующие инструкции

Плагин Validation размещает сообщение об ошибке непосредственно после элемента формы, к которому применено правило проверки. Это нормально, если сообщение расположено рядом с текстовым полем или раскрывающимся списком (как мы видели в предыдущих примерах). Но в данном случае сообщение должно находиться в другом месте, желательно после всех флажков или переключателей.

К счастью, плагин Validation предусматривает способ управления размещением сообщений об ошибках. Вы можете создавать собственные правила размещения сообщений путем передачи еще одной объектной константы функции `validate()`.

4. **Найдите сценарий проверки, созданный вами ранее, и введите запятую после закрывающей фигурной скобки для объекта `messages` (но перед комментарием `// конец messages`). Вставьте пустую строку после объекта `messages` и напишите:**

```
errorPlacement: function(error, element) {
if (element.is(":radio") || element.is(":checkbox")) {
error.appendTo(element.parent());
```

```
} else {
error.insertAfter(element);
}
} // конец errorPlacement
```

Плагин Validation принимает произвольный объект `errorPlacement`, представляющий собой анонимную функцию (см. раздел «Анонимные функции» главы 4), определяющую расположение сообщения об ошибке. Каждая ошибка обрабатывается данной функцией, так что если вам нужно изменить расположение некоторых сообщений, вам придется использовать управляющую инструкцию для определения элементов формы, чьи ошибки вы хотите переместить. Функция принимает и сообщение, и элемент формы, к которому относится ошибка, так что вы можете использовать управляющую инструкцию (см. раздел «Управляющие инструкции» главы 3) для выяснения того, является ли элемент формы флажком или переключателем. Если является, то сообщение об ошибке помещается в конец элемента, содержащего данный флажок или переключатель.

В HTML-коде этой страницы один элемент `div` охватывает группу флажков, а другой элемент `div` — группу переключателей. Таким образом, сообщение помещается перед закрывающим тегом `</div>` с помощью jQuery-функции `appendTo()` (см. раздел «Добавление содержимого на страницу» главы 4).

Вы завершили программирование для этой формы. Ниже представлен итоговый сценарий, включая функцию `$(document).ready()`:

```
1  $(document).ready(function() {
2     $('#signup').validate({
3     rules: {
4     email: {
5           required: true,
6           email: true
7     },
8     password: {
9           required: true,
10          rangelength:[8,16]
11    },
```

```
12    confirm_password: (equalTo:'#password'},
13    spam: "required"
14    }, //конец rules
15    messages: {
16          email: {
17    required: "Введите адрес электронной почты.",
18    email: "Это некорректный адрес электронной почты."
19    },
20    password: {
21          required: 'Введите пароль',
22          rangelength: 'Пароль должен содержать ↵
                от 8 до 16 символов.'
23    },
24    confirm_password: {
25          equalTo: 'Пароли не совпадают.'
26    }
27    }, // конец messages
28    errorPlacement: function(error, element) {
29          if ( element.is(":radio") || ↵
                element.is(":checkbox")) {
30          error.appendTo( element.parent());
31    } else {
32          error.insertAfter(element);
33    }
34    } // конец errorPlacement
35 }); // конец validate
36 }); // конец ready()
```

Форматирование сообщений об ошибках

Хотя сейчас страница содержит функционирующий метод проверки, сообщения об ошибках выглядят не лучшим образом. Они не просто разбросаны по странице, но и не выделяются так, как надо. Они бы выглядели намного лучше, если бы были напечатаны красным полужирным шрифтом и расположены под элементами формы, к которым они

имеют отношение. Все это вы можете сделать с помощью простой таблицы стилей.

1. **Вверху страницы *08_02.html* щелкните в пустой строке между открывающим и закрывающим тегами элемента `style`.**

   Данная страница содержит пустую таблицу стилей, в которую вы их добавите. В реальной ситуации вы, вероятно, примените внешнюю таблицу стилей — либо основную, используемую на других страницах вашего сайта, либо особую, предназначенную только для форм (например, *forms.css*). Однако в рамках данного руководства вы просто добавите новые стили на эту страницу.

2. **Добавьте следующее правило CSS в элемент `style`:**

```
#signup label.error {
font-size: 0.8em;
color: #F00;
font-weight: bold;
display: block;
margin-left: 215px;
}
```

   Селектор CSS `#signup label.error` выбирает все элементы `label` с классом `error`, которые находятся внутри другого элемента с идентификатором `signup`. В данном случае элемент `form` имеет идентификатор `signup`, и плагин Validation помещает сообщения об ошибках внутрь тега `<label>` и добавляет класс `error` (см. раздел «Настройка стилей для сообщений об ошибках» данной главы). Иными словами, это правило CSS применимо только к сообщениям внутри данной формы.

   Свойства CSS сами по себе довольно стандартны: сначала размер шрифта уменьшается до значения .8 em; цвет изменяется на красный, а текст выделяется полужирным шрифтом. Инструкция `display: block` приказывает браузеру обращаться с элементом `label` как с блочным элементом. То есть вместо того чтобы помещать сообщение об ошибке *рядом* с элементом формы, браузер воспринимает ошибку в качестве отдельного абзаца с разрывами строки вверху и внизу. Наконец, чтобы выровнять сообщение с элементами формы (имеющими отступ в 215 пикселов от левого края главной области), вам нужно добавить фрагмент кода `margin-left: 215px`.

Для еще большей наглядности вы можете использовать правила CSS для изменения вида некорректно заполненных элементов формы.

3. **Добавьте одно последнее правило в файл *form.css*:**

```
#signup input.error, #signup select.error {
background: #FFA9B8;
border: 1px solid red;
}
```

Это правило выделяет элемент формы, содержащий ошибку, окружая его красной рамкой и добавляя фоновый цвет.

С этим все. Сохраните файл CSS и просмотрите страницу *08_02.html* в браузере, чтобы увидеть, как CSS влияет на вид сообщений об ошибках (возможно, для этого браузер придется перезапустить).

В окончательном виде форма должна выглядеть, как на рис. 8.7. Полную версию руководства (*готовый_08_02.html*) вы можете найти в каталоге *глава08*.

Часть III
НАЧАЛО РАБОТЫ С JQUERY UI

Глава 9. Улучшение интерфейса

Глава 10. Стилизация форм

Глава 11. Настройка внешнего вида jQuery UI

Глава 12. Взаимодействия и эффекты jQuery UI

Глава 9
УЛУЧШЕНИЕ ИНТЕРФЕЙСА

Вы уже знаете, как с помощью библиотеки jQuery и небольшого объема программного кода можно усовершенствовать формы, изображения и ссылки. Вы научились добавлять на свои страницы анимацию и создавать такие простые элементы пользовательского интерфейса, как слайдеры (см. раздел «Всплывающее окно авторизации на практике» главы 6) и анимированные панели (см. раздел «Анимированная панель навигации на практике» главы 6). Тем не менее вы, вероятно, уже готовы к созданию более сложных элементов пользовательского интерфейса, например, всплывающих диалоговых окон, панелей с вкладками или всплывающих подсказок. Вы можете научиться программировать такие вещи с нуля, однако в разделе «Знакомство с плагинами jQuery» главы 7 вы узнали о существовании широкого спектра уже готовых к применению плагинов jQuery. jQuery UI — это плагин, позволяющий легко решить множество проблем, связанных с пользовательским интерфейсом.

Что такое jQuery UI?

jQuery UI (**jqueryui.com**) — это передовой плагин библиотеки jQuery и партнерский проект jQuery (рис. 9.1). jQuery UI предусматривает большой набор эффектов, способов взаимодействий с пользователем и элементов интерфейса (так называемых виджетов), которые упрощают процесс создания интерактивных веб-приложений. На самом деле, в главе 14 вы будете использовать jQuery UI и специальный программный код для создания простого (но полезного) веб-приложения.

Плагин jQuery UI состоит из множества различных частей, которые могут быть сгруппированы в три категории:

- **Виджеты.** Виджет — это фрагмент кода JavaScript, создающий один удобный элемент интерфейса. Например, виджет Dialog позволяет отображать всплывающие диалоговые окна, подобные окнам с оповещениями (см. раздел «Создание диалоговых окон с сообщениями» далее в главе), внешний вид и работу которых вы способны контро-

лировать. Можно использовать это диалоговое окно, чтобы показать посетителю, например, форму авторизации, или отобразить правила использования сайта. Вы можете использовать диалоговое окно для представления важного сообщения всякий раз, когда кто-то просматривает ваш сайт, или для вывода информации об изображении при наведении на него указателя мыши.

В качестве другого примера можно привести виджет DatePicker, который облегчает посетителям процесс выбора даты. Он открывает всплывающее окно с календарем. Пользователь может выбрать дату в этом календаре с помощью щелчка мыши. Вы можете использовать этот виджет как часть формы для бронирования номера в отеле («Дата прибытия») или как способ навигации по списку предстоящих событий.

Плагин jQuery UI предусматривает множество виджетов. О некоторых из них вы узнаете в этой и в следующей главах.

- **Взаимодействия.** jQuery UI включает некоторые очень полезные инструменты, позволяющие посетителям взаимодействовать с вашими веб-страницами. Например, вы можете сделать так, чтобы любой элемент на странице можно было перетаскивать. Представьте страницу интернет-магазина, на которой посетители могут буквально перетащить нужный предмет в корзину. Создайте онлайн-игру в шашки, предоставив игрокам возможность перемещать их путем перетаскивания. Вы также можете обеспечить способ изменения размера элементов страницы, например, вы открываете всплывающий диалоговый виджет с формой для создания сообщения в блоге. Посетитель, просматривающий диалоговое окно, может перетащить угол рамки, чтобы увеличить или уменьшить ее. Другими словами, вы можете заставить обычный элемент `div` вести себя подобно окну браузера, имеющему маркеры масштабирования. Плагин jQuery UI предусматривает несколько видов взаимодействия, о которых вы узнаете в главе 12.

- **Эффекты.** Библиотека jQuery предлагает различные виды анимации, например, появление, исчезновение (раздел «Постепенное появление и исчезновение элементов» главы 6), скольжение (раздел «Скользящие элементы» главы 6), а также функцию `animate()`. Однако плагин jQuery UI предусматривает множество дополнительных возможностей, включая изменение цвета, различных классов CSS и т. д. Вы узнаете об этих и других эффектах jQuery UI в разделе «Эффекты jQuery UI» главы 12.

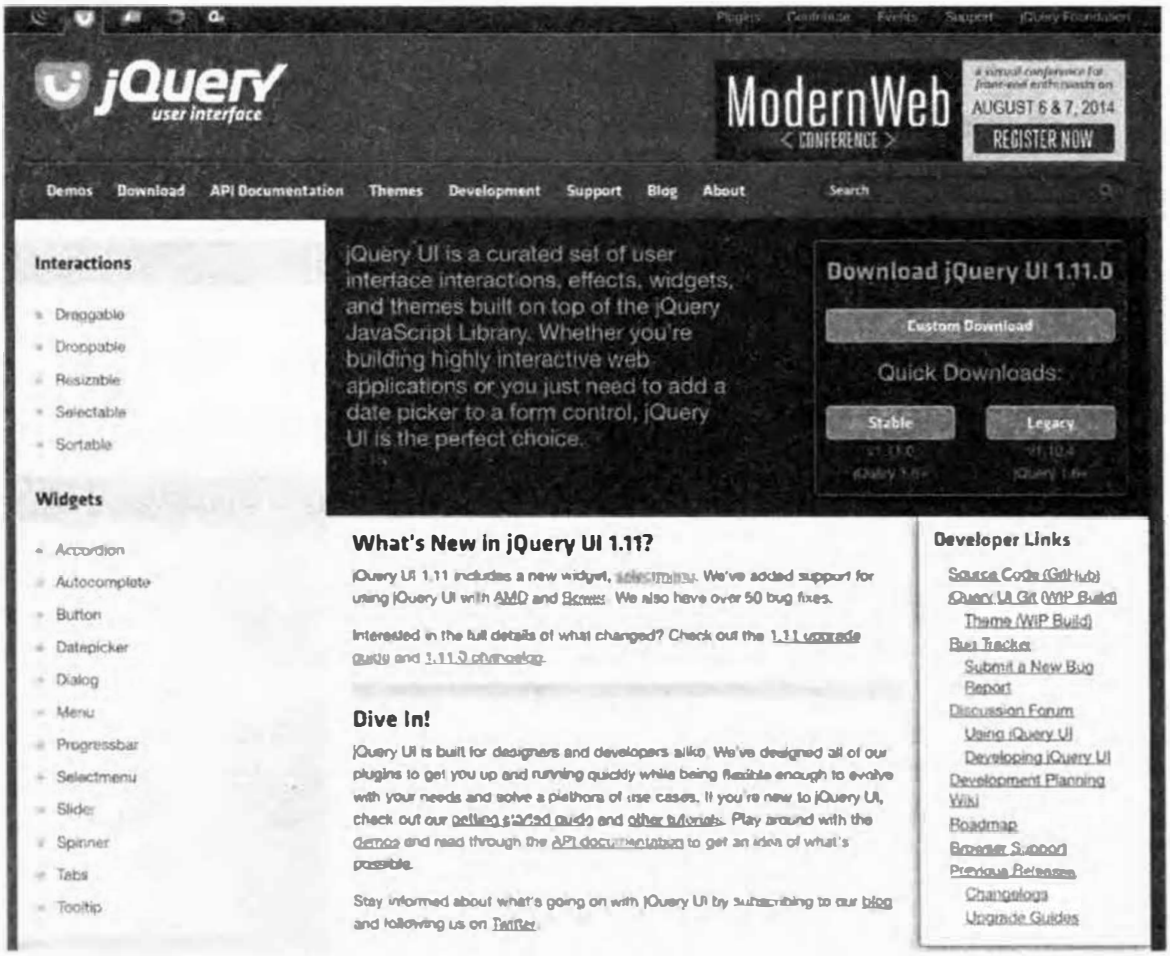

Рис. 9.1. На сайте jQuery UI вы можете загрузить плагин и создать собственные «темы» для настройки внешнего вида элементов его интерфейса. Вы можете просмотреть демонстрации различных виджетов, эффектов и взаимодействий, предусмотренных плагином, а также узнать о том, как он работает, обратившись к сайту с документацией, доступному по адресу **api.jqueryui.com**

Предназначение jQuery UI

Возможно, вы спрашиваете себя, почему из тысяч разнообразных плагинов вам следует выбрать именно jQuery UI. Даже без него вы можете найти множество инструментов для создания подсказок, вкладок и диалоговых окон. На самом деле, вы можете найти простые старые плагины jQuery, которые предусматривают все те же возможности, что и jQuery UI, и даже больше. Подробнее о них вы можете узнать из текста в рамке далее. Однако, несмотря на это, существует несколько причин, почему jQuery UI является отличным выбором:

- Плагин jQuery UI является частью jQuery Foundation (**jquery.org**) — некоммерческой организации, призванной содействовать развитию jQuery, jQuery UI и нескольких других проектов. Другими словами, jQuery и jQuery UI имеют много общего, и команды, ответственные

за эти два проекта, работают в тесном сотрудничестве, поэтому изменения в jQuery быстро повлекут изменения в jQuery UI.

- Плагин jQuery UI представляет собой полный пакет. При желании вы могли бы по частям собрать набор плагинов, которые обеспечивают все возможности jQuery UI. Однако в результате у вас были бы десятки различных плагинов от разных авторов, требующих десятки файлов CSS и JavaScript. Отслеживание всех этих различных плагинов является трудоемкой задачей. Плагин jQuery UI представляет собой один файл JavaScript с двумя файлами CSS. При внесении изменений в jQuery UI вы можете быстро и просто обновить эти файлы.
- Плагин jQuery UI обеспечивает единообразие внешнего вида. Все виджеты jQuery UI имеют единообразный вид. Панели с вкладками похожи на диалоговые окна и виджет для выбора даты, поэтому вам не придется тратить много времени на приведение различных плагинов в соответствие со стилем сайта. Кроме того, приложение ThemeRoller jQuery UI, о котором вы узнаете в главе 11, предоставляет онлайн-инструмент для настройки шрифтов, цветов, формы и дизайна jQuery UI. Это позволяет гораздо легче привести виджеты jQuery UI в соответствие с цветовой схемой, шрифтами и общим внешним видом вашего веб-сайта.
- Плагин jQuery UI — это хорошо поддерживаемый проект. Многие плагины jQuery являются плодом труда одного программиста, в лучшем случае двух или трех. Если программист потеряет интерес, получит новую работу или станет монахом, то этот плагин может больше никогда не обновиться, а существующие в нем ошибки так и не будут исправлены. Плагин jQuery UI создан большой командой разработчиков. Он постоянно обновляется, а обнаруженные ошибки быстро исправляются. Поскольку в проекте принимают участие множество людей, можно ожидать, что он просуществует в течение длительного времени. (Если вы обратитесь к списку разработчиков **github.com/jquery/jqueryui/ blob/master/AUTHORS.txt**, то увидите, что над данным проектом работает более 270 человек.)

ПАРА СЛОВ О ПЛАГИНАХ

Альтернативы для jQuery UI

jQuery UI — это не единственная библиотека, предназначенная для создания пользовательского интерфейса. Далее представлены другие популярные варианты.

- **Kendo UI** (**www.telerik.com/kendo-ui**) — это полный набор плагинов для создания веб- (и мобильных) приложений. Он включает в себя некоторые из функций jQuery UI, например, виджет для выбора даты и создания подсказок, однако он также предусматривает дополнительные инструменты визуализации данных для создания различных типов графиков и диаграмм. Как и jQuery, эта библиотека включает настройщик тем и предусматривает обширную документацию. К сожалению, в отличие от бесплатно распространяемой jQuery UI, библиотека Kendo UI стоит от 399 до 699 долларов США.
- **Wijmo UI** (**wijmo.com**) представляет собой набор продвинутых виджетов пользовательского интерфейса. Он основан на jQuery UI и jQuery Mobile и предусматривает более 40 различных виджетов, которые содержат диаграммы, сетки, таблицы и т. д. Это превосходный набор плагинов, который предоставляет все, что требуется веб-приложению (и даже больше), и хорошо работает на мобильных и настольных компьютерах. Тем не менее, поскольку стоимость данного набора составляет от 495 до 1195 долларов США, позволить себе его может, скорее, компания, имеющая на это средства.
- **jQWidgets** (**www.jqwidgets.com**) — это еще один набор плагинов с настройщиком тем и большим спектром виджетов, среди которых таблицы данных, сетки, слайдеры, панель выбора цветов и многое другое. Как и другие перечисленные здесь варианты, набор jQWidgets является коммерческим продуктом и стоит от 199 долларов США.

Использование плагина jQuery UI

Вы можете найти jQuery UI на сайте **jqueryui.com**. На главной странице присутствуют элементы управления для загрузки необходимых файлов (рис. 9.1). Пропустите ссылки **Quick Downloads** — эти файлы предназначены для программистов, которые собираются изучать или дорабатывать код JavaScript, лежащий в основе jQuery UI. (Если вы щелкнете по этой ссылке, вы найдете множество файлов, используемых при создании jQuery UI, многие из которых предназначены для автоматизации процесса создания этого плагина и бесполезны для использования jQuery UI на сайте.)

Вместо этого щелкните по большой кнопке **Custom Download** или по кнопке **Download** на панели навигации в верхней части страницы. После этого вы попадете на страницу **Download Builder** (рис. 9.2), где сможете выбрать нужные вам компоненты. Например, если вы сочтете индикатор прогресса, слайдер и счетчик бесполезными, то сможете

сбросить соответствующие флажки, чтобы исключить их из загрузки. Таким образом, вы можете уменьшить размер файла плагина jQuery UI, включив в него только действительно нужные вам компоненты.

После выбора интересующих вас компонентов вы увидите раздел под названием **Theme** в нижней части страницы. В этой области вы можете выбрать различные темы, которые сможете использовать с jQuery UI, и даже получить доступ к приложению ThemeRoller, который позволяет выбрать цветовую схему, шрифты и дизайн jQuery UI (просто щелкните по ссылке **design a custom theme**). Подробнее о темах и способах их создания вы узнаете в главе 11.

Рис. 9.2. Конструктор загрузки jQuery UI позволяет настроить плагин в соответствии с вашими нуждами. Вы можете выбрать только нужные виджеты, взаимодействия и эффекты, сбросив соответствующие флажки. Если хотите выбрать небольшое количество компонентов, сбросьте флажок **Toggle All** (Выбрать все) (выделен на рисунке), чтобы сбросить все флажки, а затем установите только те, которые вам нужны

Некоторые виджеты зависят от других компонентов плагина. К счастью, конструктор загрузки достаточно «умен», чтобы автоматически включить любые необходимые дополнительные элементы. Например, если вы сбросите все компоненты, а затем выберете виджет Accordion, то

конструктор загрузки автоматически установит флажки Core и Widget, поскольку они необходимы для работы этого виджета.

Для получения файлов jQuery UI просто щелкните по кнопке **Download** в нижней части страницы **Download Builder**. После этого будет загружен Zip-файл, содержащий папку с именем вида *jquery-ui-1.11.1.custom*. Внутри этой папки содержатся еще два каталога (рис. 9.3 слева). Вам нужен только каталог *images*, содержащий графические элементы, используемые плагином jQuery UI. Вы можете проигнорировать папку *external*: она содержит библиотеку jQuery, которую вы, вероятно, уже загрузили и добавили на свой сайт.

Чтобы использовать плагин jQuery UI, вам нужен соответствующий JavaScript-файл, содержащий программный код, необходимый для создания впечатляющих виджетов, эффектов и взаимодействий jQuery UI. Кроме того, вам понадобится файл CSS, который применяет стили к виджетам и эффектам. На рис. 9.3 показано множество файлов CSS и JavaScript, так что вам предстоит решить, какие из них использовать.

Как вы знаете из раздела «Загрузка файла jQuery» главы 4, файлы jQuery, имеющие в названии фрагмент *min*, например *jquery-ui.min.js*, являются «минимизированными», то есть для максимального уменьшения размера файла из него были удалены ненужные пробелы (и возвраты каретки), а также произведена другая оптимизация. На своем сайте лучше всего использовать минимизированные файлы, поскольку они быстро загружаются. Тем не менее процесс минимизации делает их нечитаемыми, так что вы не сможете внести в них какие-либо изменения. Вам вообще не понадобится читать JavaScript-файл jQuery UI, если вам не требуется разобраться в его программном коде, поэтому на своем сайте вам следует использовать файл *jquery-ui.min.js*.

Выбор CSS-файлов представляет собой немного более сложную задачу, поскольку существуют шесть различных файлов, которые можно загрузить! На самом деле, вам нужен только файл *jquery-ui.min.css*, поскольку он содержит весь необходимый код CSS для обеспечения работы плагина jQuery UI. Правое изображение на рис. 9.3 демонстрирует хороший способ организации файлов jQuery и jQuery UI.

ПРИМЕЧАНИЕ

Почему существует так много файлов CSS? В дополнение к файлу *jquery-ui.min.css* вы найдете файлы *jquery-ui.theme.min.css* и *ui.*

structure.min.css. *Структурный* файл содержит код CSS, отвечающий за «структуру» CSS-виджетов, например, информацию о размещении элементов на странице, в то время как *тематический* файл содержит только данные о цветах, шрифтах, размерах шрифтов, отступах и других визуальных аспектах. Другими словами, для обеспечения эффектов файла *jquery-ui.min.css* вам потребовалось бы прикрепить и тематический, и структурный файлы. Не беспокойтесь об этом.

Добавление jQuery UI на веб-страницу

jQuery UI — это просто плагин jQuery, поэтому к нему применимы те же основные правила, о которых вы прочитали в разделе «Знакомство с плагинами jQuery» главы 7. Вам нужно прикрепить файл CSS, jQuery, JavaScript-файл jQuery UI, немного изменить HTML-код, а затем вызвать функцию плагина. Далее подробно описаны все необходимые действия:

1. **Загрузите плагин jQuery UI, следуя описанной в предыдущем разделе процедуре.**

   После загрузки файлов вам нужно переместить файлы и папки, необходимые для работы плагина jQuery UI, на свой сайт, как показано на рис. 9.3. Поместите файл *jquery-ui.min.css* в папку *images*, содержащуюся в каталоге с CSS-файлами вашего сайта, а файл *jquery-ui.min.js* — в папку с файлами JavaScript. Всегда используйте минимизированные файлы, содержащие в своем названии фрагмент *min*. Их небольшой размер обеспечивает более быструю загрузку. Для использования плагина jQuery UI вам необходимы только файлы *jquery-ui.min.js* и *jquery-ui.min.css*, а также папка с изображениями. Поместите папку *images* в ту же папку, в которой содержится файл CSS (см. рис. 9.3 справа).

2. **Прикрепите CSS-файл плагина jQuery UI к веб-странице следующим образом:**

```
<link href="css/jquery-ui.min.css" ↵
rel="stylesheet">
```

   Также рекомендуется прикрепить тематическую таблицу стилей перед таблицей стилей вашего сайта:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
```

Таким образом, если хотите внести небольшие изменения в тему, вы должны включить в таблицу стилей вашего сайта правила CSS, которые переопределяют правила в тематической таблице стилей. Как правило, вам не захочется изменять код CSS в таблице стилей jQuery UI, поскольку в этом случае вам, вероятно, потребуется заменить его новой таблицей стилей после выхода новой версии jQuery UI. (Подробнее о темизации и CSS, используемых в jQuery UI, вы узнаете в главе 11.)

3. **Прикрепите JavaScript-файлы jQuery и jQuery UI:**

```
<script src="js/jquery-1.11.0.min.js"></script>

<script src="js/jquery-ui.min.js"></script>
```

Файл jQuery UI не будет работать без библиотеки jQuery, поэтому вы должны убедиться в том, что файл jQuery загружается перед jQuery UI.

После добавления файлов на сайт и их прикрепления к веб-страницам вы можете приступить к использованию jQuery UI. Поскольку каждый эффект, взаимодействие и виджет имеет свои особенности, не существует определенного набора инструкций. Далее в этой главе будут описаны некоторые из самых полезных виджетов плагина jQuery UI.

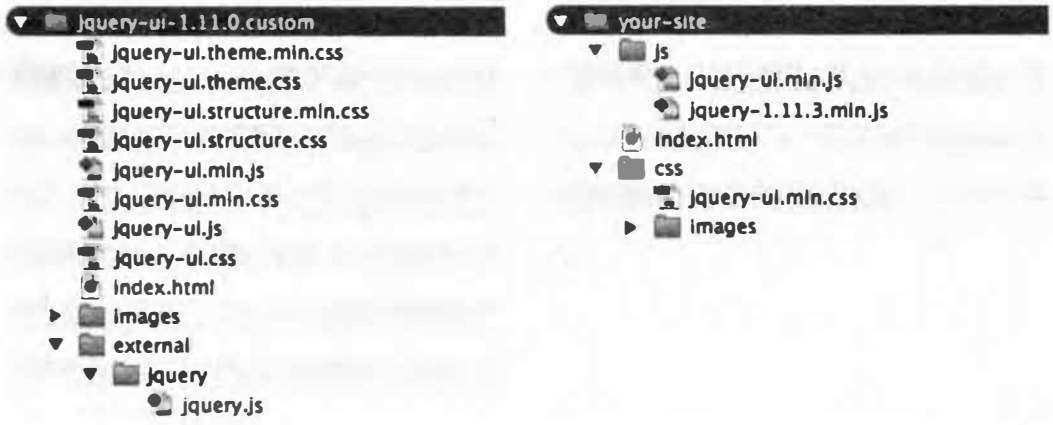

Рис. 9.3. Загруженная версия jQuery UI включает множество файлов, большая часть которых вам не нужна (слева). Файлы, в названиях которых нет фрагмента *min*, являются несжатыми файлами CSS и JavaScript. Вы можете использовать их для исследования кода CSS и JavaScript

Создание диалоговых окон с сообщениями

Окно с оповещением, предусмотренное веб-браузером, выглядит довольно нелепо (обратитесь к рис. 1.3), оно не соответствует внешне-

му виду веб-страницы, и вы не можете изменить цвет или шрифт текста. Существуют также сообщения с предупреждениями, которые не должны видеть ваши посетители. Например, браузер Chrome добавляет в окно с оповещением флажок с фразой: «Запретить этой странице создавать дополнительные диалоговые окна». К счастью, плагин jQuery UI предусматривает диалоговый виджет, который позволяет вам создавать свои собственные диалоговые окна, подобные изображенному на рис. 9.4. В диалоговые окна jQuery UI вы можете добавлять текст, формы и изображения, форматировать их в соответствии с дизайном вашего сайта и даже программировать их на выполнение определенных действий в ответ на действия пользователя.

Принцип использования диалогового виджета (как и большинства плагинов jQuery) удивительно прост:

1. **Выполните приведенную ранее инструкцию, чтобы прикрепить файлы CSS и JavaScript.**

   Вам нужно с чего-то начать!

2. **Добавьте элемент `div` с содержимым, которое будет отображаться в диалоговом окне, и атрибут `title`, содержащий слова заголовка диалога.**

   Например, далее приведен HTML-код для диалогового окна, показанного на рис. 9.4:

   ```
   <div id="hello" title="Привет, Мир!">
   <p>Это диалоговое окно на самом деле является элементом div, размещенным на странице с помощью абсолютного позиционирования.</p>
   <p>Попробуйте перетащить диалоговое окно по экрану. Вы можете это сделать!</p>
   </div>
   ```

   Поскольку вам нужно дать плагину jQuery UI инструкцию превратить этот элемент `div` в диалоговое окно, вам необходим способ его идентификации. Хорошим способом является добавление идентификатора, например, `id="hello"`.

ПРИМЕЧАНИЕ

Диалоговое окно не обязательно должно представлять собой элемент `div`. Вы можете использовать любой блочный элемент, например, `article` или `p`.

3. **Добавьте на страницу jQuery-функцию `$(document).ready()`:**

```
$(document).ready(function() {
});  // окончание ready
```

4. **С помощью jQuery выберите элемент `div` и вызовите функцию `dialog()`:**

```
$(document).ready(function() {
$('#hello').dialog();
});  // окончание ready
```

В данном примере вы используете фрагмент `$('#hello')`, поскольку этот идентификатор вы присвоили элементу `div` в шаге 2. Однако вы можете использовать любой способ выбора элементов, предусмотренных jQuery (см. раздел «Выбор элементов страницы: подход jQuery» главы 4), чтобы выбрать HTML-код и превратить его в диалоговое окно.

Эти шаги заставят диалоговое окно появиться после загрузки страницы. Этот способ подходит, если вам требуется создать для посетителей сайта срочное уведомление, например, «Сайт будет закрыт на техническое обслуживание с 3:00 до 4:00 утра» или вам требуется вывести на экран рекламное объявление, прежде чем отобразить контент вашего сайта. В следующем разделе объясняется, как оставить диалоговое окно скрытым во время загрузки страницы и отображать его только в ответ на определенное событие. Однако сначала вы узнаете о том, как создать диалоговое окно.

Создание диалогового окна на практике

Теперь, когда вы знаете, как работает виджет Dialog, вы можете добавить простое диалоговое окно, которое появляется при загрузке страницы.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *09_01.html* в папке *глава09*.**

Этот документ уже содержит ссылку на файл jQuery и функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5), однако вам предстоит добавить ссылки на файлы CSS и JavaScript плагина jQuery UI.

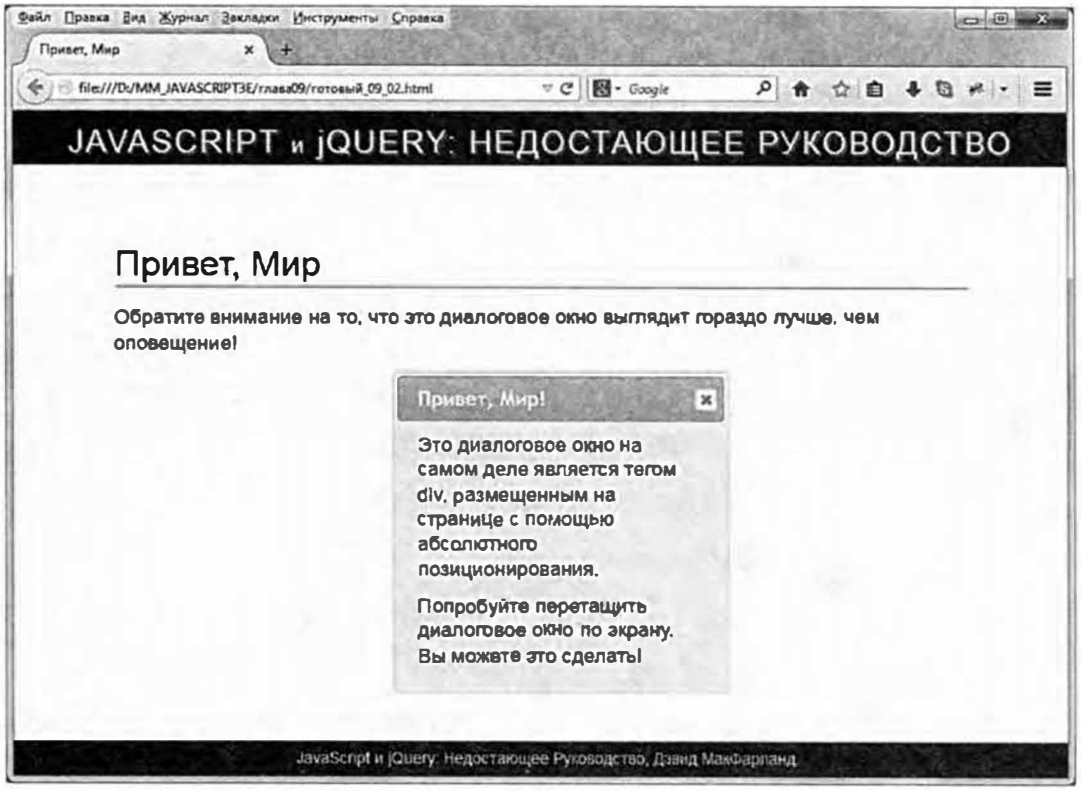

Рис. 9.4. Виджет Dialog плагина jQuery UI позволяет очень легко добавлять диалоговые окна, которые соответствуют внешнему виду вашего сайта

2. **Добавьте строки, выделенные полужирным шрифтом, в раздел заголовка страницы:**

```
<link href="../_css/ui-lightness/jquery-ui.min.css" ↵
rel="stylesheet">
<link href="../_css/site.css" rel="stylesheet">
<script src="../_js/jquery.min.js"></script>
<script src="../_js/jquery-ui.min.js"></script>
```

Обратите внимание на то, что таблица стилей jQuery UI прикрепляется перед файлом *site.css*, а файл jQuery UI прикрепляется после файла jQuery. Теперь пришло время добавить HTML-код, составляющий диалоговое окно.

3. **Найдите пустую строку непосредственно под HTML-комментарием `<!-- Добавьте тег <div> диалогового окна здесь -->` и введите:**

```
<div id="hello" title="Привет, Мир!">
<p>Диалоговое окно jQuery UI</p>
</div>
```

Далее вам предстоит превратить этот элемент `div` в диалоговое окно.

4. **В пустой строке внутри функции `$(document).ready()` добавьте выделенный полужирным шрифтом код:**

```
$(document).ready(function() {
$('#hello').dialog();
}); // окончание ready
```

Этот код выбирает только что добавленный элемент `div` и превращает его в диалоговое окно. Это действительно так просто.

5. **Сохраните страницу и просмотрите ее в браузере.**

Появится диалоговое окно. Вы можете перемещать его по экрану, перетаскивая за оранжевую полосу с заголовком, и даже увеличить или уменьшить, перетащив любой из его четырех углов. Снова наведите указатель мыши на область заголовка и обратите внимание на то, как он принимает вид указателя, состоящего из четырех стрелок. Это не является обычным поведением вашего браузера, это запрограммировано прямо в jQuery UI.

Вы найдете готовую версию этого урока в файле *готовый_09_01.html*.

Настройка свойств диалогового окна

Вы можете настроить различные свойства диалогового окна, например, высоту, ширину способ появления и исчезновения с экрана, передав функции `dialog()` список имен и значений в объекте. Как вы знаете из раздела «Одновременное изменение нескольких свойств CSS» главы 4, литерал объекта представляет собой группу пар имя/значение, заключенных в фигурные скобки:

```
{
name : 'Dave',
awesomeAuthor : true
}
```

Вы передаете объект с понятными для плагина параметрами, когда вызываете функцию `dialog()`. Например, вы не хотите, чтобы посетители могли перетаскивать или изменять размер диалогового окна. В этом случае вы просто передаете литерал объекта с двумя параметрами:

```
$('#hello').dialog({
draggable : false,
resizable : false
});
```

Виджет Dialog предотвращает перетаскивание, когда для свойства `draggable` задано значение `false`, а также предотвращает изменение размеров диалогового окна, когда для свойства `resizable` задано значение `false`. Далее перечислены некоторые из самых полезных свойств.

- **`draggable`**. Задайте для этого свойства значение `false`, чтобы зафиксировать диалоговое окно на экране и предотвратить его перетаскивание посетителями. (Если вы хотите, чтобы посетители имели возможность перемещать диалоговое окно, вам не нужно ничего делать, поскольку это нормальное поведение виджета Dialog.)
- **`resizable`**. Задайте для этого свойства значение `false`, чтобы сохранить размер диалогового окна по умолчанию и предотвратить его изменение посетителями. (Опять же, если вы хотите дать посетителям возможность изменять размер диалогового окна, вам ничего не нужно делать; это нормальное поведение виджета.)
- **`height`** и **`width`**. Обычно плагин jQuery UI задает для диалогового окна такой размер, который позволяет отобразить его содержимое. Однако вы можете управлять этим поведением, указав точное значение высоты и ширины в пикселах. Например, если вы хотите, чтобы ширина диалогового окна составляла 600 пикселов, а высота — 400 пикселов, можете добавить следующий код в литерал объекта, передаваемый функции `dialog()`:

  ```
  width: 600,
  height: 400
  ```

Разрешается использовать только значения в пикселах (процентные значения или единицы измерения em использовать нельзя), однако вам нужно опустить часть px, которая обычно сопровождает значение в пикселах в коде CSS. Если вы установите такие значения ширины и высо-

ты, при которых содержимое диалогового окна в нем не поместится, то плагин jQuery UI добавит внутри окна полосу прокрутки. Посетители смогут просмотреть все содержимое. (Подобное поведение встречается во многих всплывающих окнах «Правила и условия», которые необходимо прокручивать в течение нескольких дней, чтобы прочитать весь текст, написанный мелким шрифтом.)

Вам необязательно устанавливать оба свойства. Возможно, вам нужно убедиться в том, что диалоговое окно имеет определенную ширину, а его высота не важна. В этом случае достаточно указать свойство `width`.

- **Минимальная ширина и высота**. Вы можете задать для диалогового окна минимальные значения ширины и высоты, указав значения свойств `minWidth` и `minHeight`. Например, если вы хотите, чтобы ширина диалогового окна составляла, по крайней мере, 600 пикселов, а высота — 400 пикселов, то вы можете задать эти свойства следующим образом:

  ```
  minWidth: 600,
  minHeight: 400
  ```

  Когда вы устанавливаете минимальную высоту и ширину, плагин jQuery UI может задать для диалогового окна большие, но не меньшие значения. Другими словами, если содержимое окна в нем не помещается, то плагин jQuery UI увеличит его для отображения всего контента.

- **Максимальная ширина и высота**. Вы также можете задать максимальные значения высоты и ширины диалогового окна, используя свойства `maxWidth` и `maxHeight`. Например, если вы хотите, чтобы ширина диалогового окна не превышала 600 пикселов, а высота — 400 пикселов, то можете использовать эти свойства следующим образом:

  ```
  maxWidth: 600,
  maxHeight: 400
  ```

  При небольшом объеме содержимого плагин jQuery UI создаст диалоговое окно, размеры которого окажутся меньше указанных значений, но никогда не превысят их. Если содержимое не будет помещаться в диалоговом окне, то плагин jQuery UI добавит полосу прокрутки, чтобы посетители могли прокрутить контент и ознакомиться с ним.

- **modal**. Модальное диалоговое окно используется для того, чтобы обратить внимание посетителя на сообщение, и предотвращает какие-либо действия, пока тот его не закроет. Когда модальное диалоговое окно открывается, посетитель не может щелкнуть кнопкой мыши в другом месте страницы. На самом деле, на странице появляется темное прозрачное наложение, затрудняющее чтение. Используйте модальные диалоговые окна, если хотите, чтобы посетитель сначала прочитал сообщение и, возможно, принял важное решение, например, ответил на вопрос: «Вы уверены, что хотите удалить все эпизоды сериала "Доктор Кто" из своей видеотеки?» Чтобы диалоговое окно стало модальным, задайте для свойства `modal` значение `true`:

  ```
  modal: true
  ```

ПРИМЕЧАНИЕ

Для получения полного списка вариантов, а также более подробной информации об использовании виджета Dialog посетите страницу **api.jqueryui.com/dialog/**.

- **show** и **hide**. Диалоговое окно обычно появляется на экране, когда оно открыто, и исчезает, когда закрывается. Однако какой в этом интерес? Вы можете анимировать открытие и закрытие диалогового окна, используя свойства `show` и `hide`. Эти свойства могут принимать различные значения. Если вы зададите для них значение `true`, то диалоговое окно будет быстро отображаться и исчезать:

  ```
  show: true,
  hide: true
  ```

  Вы также можете предоставить численное значение, определяющее количество миллисекунд, в течение которых диалоговое окно станет появляться или исчезать. Например, вам нужно, чтобы диалоговое окно появлялось очень быстро, скажем за 250 миллисекунд, и исчезало в течение 2 секунд. В этом случае вам следовало бы установить эти свойства следующим образом:

  ```
  show: 250,
  hide: 2000
  ```

  Вы не ограничены только эффектами появления и исчезновения. Вы можете передать в качестве значения этих свойств имя любого эффекта jQuery (см. раздел «Эффекты jQuery» главы 6). Просто за-

ключите имя эффекта в кавычки, например, `'slideDown'`. Чтобы заставить диалоговое окно скользить, используйте следующий код:

```
show: 'slideDown',
hide: 'slideUp'
```

Вы также можете использовать такие эффекты jQuery UI (см. раздел «Эффекты jQuery UI» главы 12), как `'scale'` или `'explode'`. Если вам этого недостаточно, вы также можете передать еще один литерал объекта, содержащий имя эффекта, продолжительность, задержку и значение параметра `easing` (см. раздел «Управление скоростью анимации» главы 12). Например, вы хотите, чтобы при закрытии диалогового окна после задержки в 250 миллисекунд оно взрывалось в течение 1 секунды, используя функцию `'easeInQuad'` (пожалуйста, не пытайтесь повторить это дома). В этом случае вы можете передать вместе с объектом следующую строку:

```
hide: { effect: 'explode', delay: 250, duration: ↵
1000, easing: 'easeInQuad' }
```

- **`position`**. Обычно диалоговые окна появляются в центре интерфейса браузера. Однако это необязательно. Вы можете настроить позиционирование диалогового окна с помощью свойства `position`. Как и свойства `show` и `hide`, оно может принимать несколько различных типов значений. В случае с простыми координатами X,Y передайте массив (см. раздел «Массивы» главы 2), содержащий два числа. Первое число определяет расстояние от левого края окна браузера до левого края диалогового окна в пикселах, а второе — расстояние от верхнего края окна браузера до верхнего края диалогового окна в пикселах. Например, вам нужно, чтобы диалоговое окно располагалось на расстоянии 100 пикселов от левого края экрана и очень близко к верхней границе экрана (скажем, на расстоянии в 10 пикселов от верхнего края). В этом случае вы бы указали для свойства `position` следующие значения:

```
position: [100,10]
```

Для указания положения диалогового окна вы можете также использовать ключевые слова `center`, `left`, `top`, `right`, `bottom`. Например, для позиционирования окна в правом нижнем углу экрана используйте свойство `position` следующим образом:

```
position: 'right bottom'
```

Первое ключевое слово определяет положение по горизонтали — `left`, `center` или `right`, а второе ключевое слово определяет положение по вертикали — `top`, `center` или `bottom`. Разделите два ключевых слова пробелом.

Наконец, вы можете передать свойство `position` объекту jQuery UI position. Эта полезная утилита обсуждается в тексте в рамке далее в главе.

Передача параметров виджету Dialog на практике

При вызове функции `dialog()` вы можете комбинировать любые несколько или все перечисленные выше свойства. Вы можете увидеть, как работают свойства, продолжив работу над кодом диалогового окна из предыдущего руководства. В этом уроке вы сделаете диалоговое окно модальным, чтобы посетитель не мог ничего предпринять, пока окно не будет закрыто. Кроме того, вы предотвратите перемещение или изменение размера диалогового окна и, наконец, обеспечите интересный эффект его исчезновения:

1. **В текстовом редакторе вернитесь к файлу *09_01.html*, над которым вы работали в предыдущем руководстве.**

   Вы начнете с передачи пустого литерала объекта функции `dialog()`.

2. **Установите текстовый курсор между открывающей и закрывающей круглыми скобками функции `dialog()`. Введите `{`, дважды нажмите клавишу Enter, а затем введите `}`. Код должен выглядеть следующим образом:**

```
$(document).ready(function() {
$('#hello').dialog({
});
}); // окончание ready
```

   Теперь вы можете использовать пары свойство/значение. Сначала вы сделаете диалоговое окно модальным.

3. **Внутри литерала объекта введите `modal: true`:**

```
$(document).ready(function() {
$('#hello').dialog({
```

```
modal: true
});
}); // окончание ready
```

Сохраните файл и откройте его в веб-браузере. Остальная часть экрана вокруг диалогового окна будет затемнена прозрачным полосатым фоном. Чтобы снова четко увидеть страницу, вы должны закрыть диалоговое окно.

4. **В строке, которую вы только что ввели, добавьте запятую после значения `true`. Нажмите клавишу Enter и добавьте строки, выделенные полужирным шрифтом:**

```
$(document).ready(function() {
$('#hello').dialog({
modal: true,
resizable: false,
draggable: false
});
}); // окончание ready
```

Помните о том, что вы должны разделять пары имя/значение запятыми. Каждая строка с парой имя/значение должна заканчиваться запятой, за исключением последней пары имя/значение в объекте. Эти две новые строки предотвращают перемещение или изменение размера диалогового окна. Наконец, пришло время обеспечить интересное закрытие диалогового окна.

5. **Добавьте запятую после значения `false` в последней введенной вами строке, нажмите клавишу Enter и введите `hide: 'explode'`:**

```
$(document).ready(function() {
$('#hello').dialog({
modal: true,
resizable: false,
draggable: false,
hide: 'explode'
});
}); // окончание ready
```

Этот код заставляет плагин jQuery UI применить эффект `'explode'` при закрытии диалогового окна. Сохраните страницу и просмотрите ее в браузере. Посмотрите, что происходит при закрытии диалогового окна. Вы можете поэкспериментировать с другими эффектами jQuery UI, заменив фрагмент `'explode'` на `'bounce'`, `'blinds'` или `'drop'`.

Вы найдете готовую версию этого урока в файле *готовый_09_01_2.html* в папке *глава09*.

Открытие диалоговых окон с помощью событий

Диалоговые окна jQuery UI просты в использовании и являются отличной заменой скучным окнам с предупреждениями, предусмотренным браузерами. Тем не менее вам, вероятно, не захочется видеть диалоговое окно при каждом посещении веб-страницы. Диалоговые окна гораздо полезнее, когда они открываются в ответ на действие посетителя. Например, если посетитель щелкает по кнопке «Подписаться на рассылку», вместо того, чтобы перенаправлять его на другую веб-страницу, браузер может открыть диалоговое окно с регистрационной формой на текущей странице.

Вы способны открывать диалоговые окна в ответ на любое из тех событий, о которых узнали в главе 5, например, на щелчок кнопкой мыши, нажатие клавиши или изменение размера окна (но это было бы странно). Кроме того, вы всегда можете открыть диалоговое окно, написав собственный программный код. Например, вы создали викторину, проводимую на время. Если оно истечет до того, как посетитель ее закончит, вы можете открыть диалоговое окно с надписью «Время вышло!»

Чтобы открыть диалоговое окно, вы должны сделать несколько вещей. Во-первых, предотвратить открытие окна при его создании, поскольку обычно оно появляется сразу (как вы видели в предыдущем уроке). Во-вторых, вам нужен способ вызова диалогового окна. Чаще всего вы будете использовать обработчик события (см. раздел «Использование событий: способ jQuery» главы 5).

Чтобы скрыть диалоговое окно в момент его создания, необходимо передать функции `dialog()` свойство `autoOpen` со значением `false`. Например, вы добавили на страницу элемент `div` с идентификатором `login`. Чтобы превратить этот элемент в диалоговое окно jQuery UI и сразу скрыть его, добавьте следующий код JavaScript:

```
$('#login').dialog({
autoOpen: false
});
```

При загрузке страницы плагин jQuery UI превратит элемент `div` в объект диалогового окна и скроет его на странице. Чтобы сделать его видимым, вы должны передать функции `dialog()` аргумент `open`. Например, у вас есть кнопка «Вход на сайт», и вы хотите, чтобы в ответ на нажатие этой кнопки открывалось диалоговое окно. Предположим, ссылка имеет идентификатор `loginLink`. Вы можете выбрать эту ссылку, добавить обработчик события `click` и применить функцию, которая откроет диалоговое окно:

```
$('#loginLink').click(function(evt) {
evt.preventDefault();
$('#login').dialog('open');
}); // окончание click
```

Вторая строка кода — `evt.preventDefault();` — необходима для того, чтобы предотвратить переход браузера по ссылке и загрузку новой веб-страницы (описание метода `preventDefault()` вы можете найти в разделе «Отмена обычного поведения событий» главы 5).

Кроме того, поскольку посетитель всегда может закрыть диалоговое окно, щелкнув по кнопке **Close** (Закрыть) в его правом верхнем углу, вы также можете запрограммировать закрытие диалогового окна.

Например, если вы создадите диалоговое окно с формой, то, вероятно, захотите сделать так, чтобы оно закрылось после того, когда посетитель отправит форму, вместо того, чтобы заставлять посетителя самостоятельно закрывать диалоговое окно после отправки формы.

ПРИМЕЧАНИЕ

Вы увидите примеры с открытием и закрытием диалоговых окон в следующем руководстве.

Теперь предположим, что диалоговое окно, обсуждаемое в данном разделе — элемент `div` с идентификатором `login` — содержит форму. Вам нужно, чтобы это диалоговое окно закрывалось, когда посетитель отправляет данные формы. Вот как вы можете это сделать:

```
$('#login form').submit(function() {
$('#login').dialog('close');
}); // окончание submit
```

Чтобы закрыть диалоговое окно, выберите элемент `div`, вызовите функцию `dialog()` и передайте `'close'` в качестве аргумента. Вот и все.

ПРИМЕЧАНИЕ

Приведенный выше пример с веб-формой является только частью решения. При отправке формы вам также придется выполнить некоторые другие действия. Поскольку отправка формы приводит к уходу с текущей веб-страницы и приводит на страницу, отвечающую за передачу формы, вам нужно остановить эту передачу и обработать ее с помощью одного кода JavaScript, используя технологию Ajax, о которой вы узнаете в главе 13.

Добавление кнопок в диалоговое окно

Диалоговые окна подходят не только для отображения сообщений для ваших посетителей. Они также удобны для приема вводимых посетителем данных. Например, вы создали веб-приложение, которое позволяет людям создавать списки дел (вы займетесь этим в главе 14). Если кто-то добавил элемент списка дел, а потом захотел удалить его, он мог бы нажать для этого соответствующую кнопку. Чтобы предотвратить нежелательное удаление элемента списка, вы могли бы добавить всплывающее диалоговое окно с просьбой подтвердить свое намерение (рис. 9.5).

Виджет Dialog jQuery UI позволяет вам добавлять кнопки в любое диалоговое окно. Кроме того, затем вы можете запускать различные программы в зависимости от нажатой посетителем кнопки. Например, если посетитель нажимает кнопку **Удалить**, из списка удаляется один пункт, однако если он нажимает **Отменить**, то этот элемент остается на месте.

Чтобы добавить кнопки, вы передаете функции `dialog()` свойство `buttons`, значением которого является литерал объекта, содержащий имя и действие каждой кнопки. Например, вам нужно добавить в диалоговое окно кнопки **Подтвердить** и **Отменить**. Вы можете сделать это следующим образом:

```
$('#dialog').dialog({
buttons : {
"Подтвердить" : function() {
// код, выполняемый при нажатии кнопки "Подтвердить"
},
"Отменить" : function() {
// код, выполняемый при нажатии кнопки "Отменить"
}
} // окончание buttons
}); // окончание dialog
```

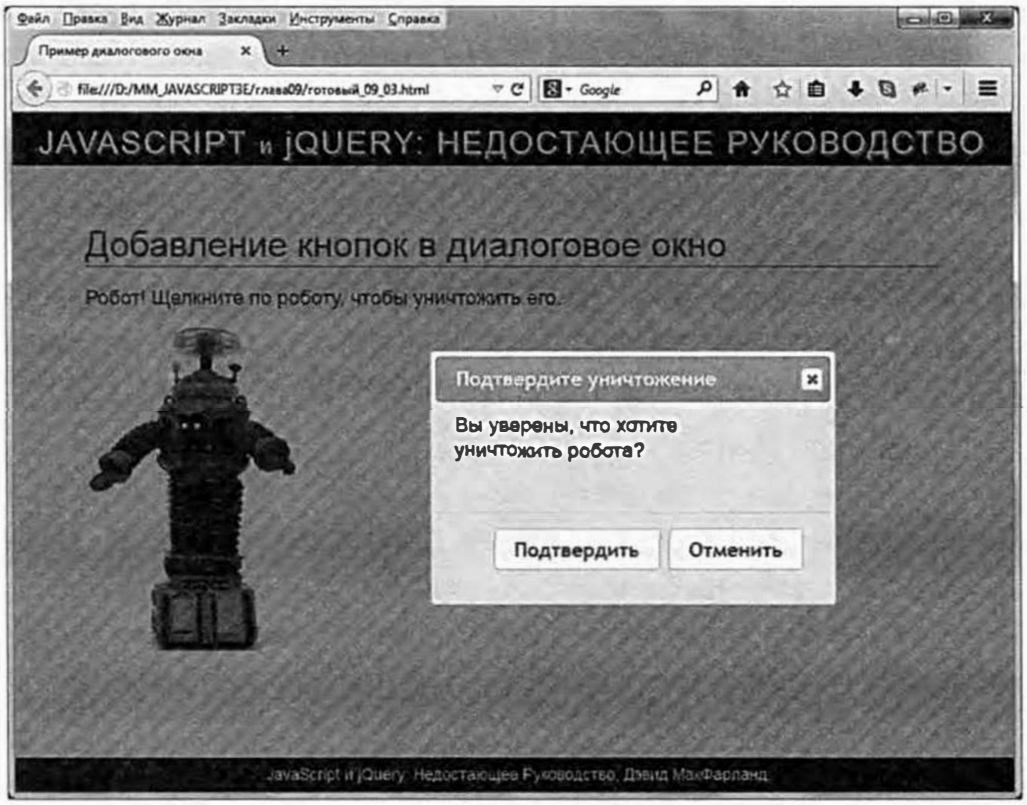

Рис. 9.5. Хотите сделать диалоговое окно интерактивным? Добавьте в него кнопку! Вы можете запрограммировать любое действие, совершаемое в ответ на нажатие этой кнопки посетителем. Например, щелчок по представленной здесь кнопке **Подтвердить** заставит изображение робота взорваться!

Это очень объемный код, содержащий много символов { и }. Однако имейте в виду то, что каждая кнопка, которую вы добавляете, является просто элементом внутри литерала объекта с именем и значением функции. Например, далее приведен код только для кнопки **Подтвердить**:

```
"Подтвердить" : function() {
```

```
// код, выполняемый при нажатии кнопки "Подтвердить"
},
```

Первая часть объекта — имя — это то, что будет отображено в качестве текста на кнопке. В данном примере это «Подтвердить». Вторая часть объекта — это функция (см. раздел «Анонимные функции» главы 4), содержащая код, который вы хотите выполнить при нажатии этой кнопки. Этот код может выполнять любое действие, начиная с удаления элемента с веб-страницы, и заканчивая запуском игры, основанной на JavaScript. В большинстве случаев вы также захотите, чтобы диалоговое окно закрывалось после нажатия кнопки. Для этого вы можете использовать аргумент `'close'`. Например, чтобы закрыть окно с кнопкой **Подтвердить** после нажатия кнопки и выполнения соответствующего кода, вы добавляете в конец анонимной функции фрагмент `$(this). dialog('close');`:

```
"Подтвердить" : function() {
// код, выполняемый при нажатии кнопки "Подтвердить"
$(this). dialog('close');
},
```

Вы читали о ключевом слове `$(this)` в разделе «Ключевые слова `this` и `$(this)`» главы 4. В контексте объекта диалогового окна `$(this)` относится к элементу `div`, который представляет собой диалоговое окно. Вы можете применить к этому элементу `div` функцию `dialog()`, чтобы закрыть его. Пришло время попрактиковаться в добавлении кнопок в диалоговое окно.

Добавление кнопок в диалоговое окно на практике

В этом руководстве вы обеспечите посетителю возможность удаления фотографии с веб-страницы с помощью щелчка по ней кнопкой мыши. Для предотвращения случайного удаления фотографии вы добавите диалоговое окно с просьбой подтвердить действие:

1. **В текстовом редакторе откройте файл *09_03.html*.**

   В данном примере файлы CSS и JavaScript плагина jQuery UI уже добавлены на страницу, так что вы начнете непосредственно с добавления диалогового окна.

2. **Найдите пустую строку под HTML-комментарием `<!-- добавьте диалоговое окно сюда -->` и введите:**

```
<div id="confirm" title="Подтвердите уничтожение">
<p>Вы уверены, что хотите уничтожить робота?</p> </div>
```

Далее вам предстоит превратить этот элемент `div` в диалоговое окно.

3. **Установите текстовый курсор в пустой строке после функции `$(document).ready()` и введите:**

```
$('#confirm').dialog({
});
```

Данный фрагмент кода превращает элемент `div` в диалоговое окно. Символы `{` и `}` обозначают пустой литерал объекта (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), который будет содержать варианты, предлагаемые диалоговым окном. Сначала вы сделаете это окно модальным, чтобы посетители не могли ничего сделать до его закрытия.

4. **Внутри литерала объекта введите код `modal: true`:**

```
$('#confirm').dialog({
modal: true
});
```

Вам также нужно сделать так, чтобы диалоговое окно было изначально скрыто. Оно будет показано, только когда посетитель щелкнет по изображению робота.

5. **Добавьте запятую после значения `true` в последней добавленной вами строке, нажмите клавишу Enter и введите код, выделенный полужирным шрифтом:**

```
$('#confirm').dialog({
modal: true,
autoOpen: false
});
```

Вы должны отделять пары имя/значение запятыми, поэтому не забудьте добавить запятую после значения `true`. Далее вы добавите в диалоговое окно кнопки, однако сначала вы добавите код, необходимый для открытия диалогового окна при щелчке по изображению

робота. Это изображение имеет идентификатор `robot`, так что вы можете легко выделить его и добавить к нему событие `click`.

6. **После функции `dialog()` добавьте три новые строки кода, чтобы ваша программа выглядела следующим образом:**

```
$(document).ready(function() {
$('#confirm').dialog({
modal: true,
autoOpen: false
});
$('#robot').click(function() {
}); // окончание click
}); // окончание ready
```

Этот код выбирает фотографию и добавляет к ней обработчик события (см. раздел «Использование событий: способ jQuery» главы 5). Теперь, чтобы открыть диалоговое окно, вы выбираете соответствующий элемент `div`, вызываете функцию `dialog()` и передаете ей значение `'open'`.

7. **Добавьте следующую строку кода в функцию `click()`:**

```
$(document).ready(function() {
$('#confirm').dialog({
modal: true,
autoOpen: false
});
$('#robot').click(function() {
$('#confirm').dialog('open');
}); // окончание click
}); // окончание ready
```

Теперь диалоговое окно открывается при щелчке по изображению робота.

8. **Сохраните файл и просмотрите его в браузере. Щелкните по изображению робота, чтобы увидеть свое диалоговое окно.**

В данный момент в диалоговом окне нет кнопок. В следующих шагах вы создадите их фрагмент за фрагментом, чтобы разобраться в принципе их работы.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Точное позиционирование с помощью плагина jQuery UI

Виджеты для создания диалоговых окон и всплывающих подсказок позволяют контролировать их положение с помощью свойства `position`. Свойство `position` является объектом (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), который определяет положение окна по отношению к другому элементу на странице. Сначала способ его написания может показаться несколько странным, однако, когда приобретете некоторый опыт, вы увидите, что он довольно прост. У этого свойства существует несколько параметров, наиболее важные из которых — `my` и `at`.

Например, если вы хотите поместить диалоговое окно в нижнем правом углу окна, вы можете написать следующий код:

```
$('#dialog').dialog({
position: {
my: 'right bottom',
at: 'right bottom'
}
}); // окончание dialog
```

Свойство `my` относится к диалоговому окну, а свойство `at` к окну браузера. Таким образом, свойство `my` определяет, какая часть диалогового окна будет находиться в том или ином положении в окне браузера (`at`). В данном примере код означает: «Помести нижний угол диалогового окна (`my`) в (`at`) нижнем правом углу окна браузера».

Вы используете синтаксис позиционирования CSS (тот же, что и в случае со свойством `background-position`). Первое ключевое слово определяет положение по горизонтали: `left`, `right` или `center`. Второе — положение по вертикали: `top`, `center` или `bottom`.

Диалоговые окна позиционируются относительно окна браузера. С другой стороны, всплывающая подсказка (см. раздел «Предоставление информации с помощью всплывающих подсказок» далее в этой главе) позиционируется относительно элемента-триггера, то

есть элемента, на который вы наводите указатель мыши, чтобы открыть подсказку. Тем не менее вы можете добавить третье свойство `of`, чтобы позиционировать диалоговое окно или подсказку относительно другого элемента страницы. Это удобно для создания презентации типа «тур по интерфейсу», которая позволяет познакомить посетителя с разными элементами веб-страницы. Например, вы можете вывести диалоговое окно рядом с полем «Вход», чтобы показать посетителям, где они могут авторизоваться.

Свойство `of` принимает селектор (см. раздел «Выбор элементов страницы: способ jQuery» главы 4) или элемент jQuery. Допустим, панель авторизации имеет идентификатор `login`, и вы хотите, чтобы диалоговое окно отображалось непосредственно под ним. В этом случае вы можете использовать следующий код:

```
$('#dialog').dialog({
position: {
my: 'center top',
at: 'center bottom',
of: '#login'
}
}); // окончание dialog
```

Вы даже можете предоставить значение смещения для достижения еще большей степени контроля над расположением диалогового окна или подсказки. Например, если вы хотите, чтобы диалоговое окно перекрывало окно авторизации на 10 пикселов, вы можете написать следующий код:

```
$('#dialog').dialog({
position: {
my: 'center top-10',
at: 'center bottom',
of: '#login'
}
}); // окончание dialog
```

Вы можете прибавлять или вычитать, используя численное (для значений в пикселах) или процентное значение (например, `my: 'center top+25%'`). Убедитесь в том, что вы не добавляете никаких пробелов между ключевым словом (например, `top`), символом операции (`+` или `-`) и значением (например, `25%`), в противном случае объект position не будет работать.

Посетите сайт **api.jqueryui.com/position/**, чтобы подробнее узнать об утилите `position` плагина jQuery UI.

9. **Вернитесь в текстовый редактор к файлу *09_03.html*. В функции `dialog()` в последней строке переданных функции параметров введите запятую после значения `false`, нажмите клавишу Enter, а затем добавьте выделенный полужирным шрифтом код:**

```
$(document).ready(function() {
$('#confirm').dialog({
modal: true,
autoOpen: false,
buttons: {
}
});
$('#robot').click(function() {
$('#confirm').dialog('open');
}); // окончание click
}); // окончание ready
```

Этот код передает диалоговому окну еще один параметр — `buttons`. Его значением является еще один литерал объекта, который состоит из элементов-кнопок. Сначала вы добавите кнопку подтверждения.

10. **В объекте `buttons` добавьте код, выделенный полужирным шрифтом:**

```
buttons : {
"Подтвердить" : function() {
}
}
```

Слово «Подтвердить» будет меткой для первой кнопки. Когда посетитель щелкает по этой кнопке, срабатывает анонимная функция. Сохраните файл и просмотрите его в веб-браузере: щелкните по изображению робота, и вы увидите диалоговое окно с кнопкой «Подтвердить». К сожалению, после щелчка по ней ничего не происходит. Вернитесь в текстовый редактор.

11. **В анонимной функции для кнопки «Подтвердить» добавьте одну строку кода:**

```
buttons : {
"Подтвердить" : function() {
$('#robot').effect('explode');
}
}
```

Этот фрагмент кода выбирает изображение робота, а затем применяет к нему эффект jQuery UI `explode`, который вы видели в действии ранее в этой главе. (Подробнее об этом и других эффектах плагина jQuery UI вы можете узнать в разделе «Эффекты jQuery UI» главы 12.)

Пришло время заняться другой кнопкой.

12. **Введите запятую после закрывающей фигурной скобки в коде для кнопки «Подтвердить», нажмите клавишу Enter и введите код, выделенный полужирным шрифтом:**

```
buttons : {
"Подтвердить": function() {
$('#robot').effect('explode');
},
"Отменить": function() {
}
}
```

Если вы просмотрите страницу сейчас, то увидите в диалоговом окне две кнопки. Кнопка «Отменить» ничего не делает. Так и должно быть, поскольку она предназначена для отмены других действий. Тем не менее вы заметите, что щелчок по ней не приводит даже к закрытию диалогового окна, хотя она должна делать, по крайней мере, это.

13. **Внутри анонимной функции для кнопки «Отменить» введите код `$(this). dialog('close');`:**

```
buttons : {
"Подтвердить": function() {
$('#robot').effect('explode');
```

```
},
"Отменить": function() {
$(this).dialog('close');
}
}
```

Поскольку кнопки создаются внутри функции `dialog()`, которая применяется к элементу `div` диалогового окна, ключевое слово `$(this)` относится к самому диалоговому окну. Таким образом, в данном случае фрагмент `$(this).dialog('close');` равнозначен фрагменту `$('#confirm').dialog('close');`.

Сохраните страницу и просмотрите ее в браузере. Щелкните по изображению робота, чтобы открыть диалоговое окно, а затем щелкните по кнопке «Отменить» — диалоговое окно закроется! Снова щелкните по изображению робота, и на этот раз нажмите кнопку «Подтвердить»: изображение робота взорвется и исчезнет. К сожалению, диалоговое окно при этом не закроется. Однако это легко исправить.

14. **Добавьте фрагмент `$(this).dialog('close');` в качестве последней строки анонимной функции для кнопки «Подтвердить». Ваш окончательный код должен выглядеть следующим образом:**

```
$(document).ready(function() {
$('#confirm').dialog({
modal: true,
autoOpen: false,
buttons : {
"Подтвердить" : function() {
$('#robot').effect('explode'); $(this) ↵
.dialog('close');
},
"Отменить" : function() {
$(this).dialog('close');
}
}
```

```
});
$('#robot').click(function() {
$('#confirm').dialog('open');
}); // окончание click
}); // окончание ready
```

Сохраните и просмотрите страницу в браузере. Готовая страница (с открытым диалоговым окном) должна выглядеть, как на рис. 9.5. Пощелкайте по обеим кнопкам и посмотрите, что произойдет. Вы найдете готовую версию этого урока в файле *готовый_09_03.html* в папке *главы09*.

Предоставление информации с помощью всплывающих подсказок

Иногда вам нужно предоставить посетителям дополнительную информацию. Например, у вас есть строка со значками социальных сетей, щелчок по которым перенаправляет посетителя на такие сайты, как Twitter, Facebook, Reddit, Instagram и т. д. Тот, кто не знаком с каким-либо сайтом, не узнает значок и не поймет, куда он его приведет. Чтобы помочь таким посетителям, вы можете добавить всплывающие подсказки, отображающиеся при наведении на них указателя мыши, например: «Моя страница в Facebook».

Плагин jQuery UI предусматривает простой способ добавления такого типа подсказок к любому элементу страницы (рис. 9.6). Вы даже можете создать более сложные подсказки, включающие целые фрагменты HTML-кода, в том числе текст, изображения и ссылки. Далее перечислены шаги, с помощью которых вы можете добавить на страницу всплывающие подсказки:

1. **Следуйте инструкции, приведенной в начале данной главы, чтобы прикрепить к своей странице файлы jQuery и jQuery UI.**

   Вы не продвинетесь далеко без необходимых файлов CSS и JavaScript.

2. **Добавьте атрибут `title` к любому элементу, который должен иметь подсказку:**

```
<a href="https://twitter.com/eksmo_live" ↵
title="Следуйте за нами в Twitter">
```

```
<img src="twitter.png">
</a>
```

Некоторые браузеры отображают подсказки сразу после добавления к элементу атрибута `title`. Однако, как и в случае с окном оповещения, вы не можете изменить стиль подсказки, предусмотренной браузером. Кроме того, не все браузеры отображают подсказки, поэтому использование плагина jQuery UI обеспечивает максимальный контроль и эффективность.

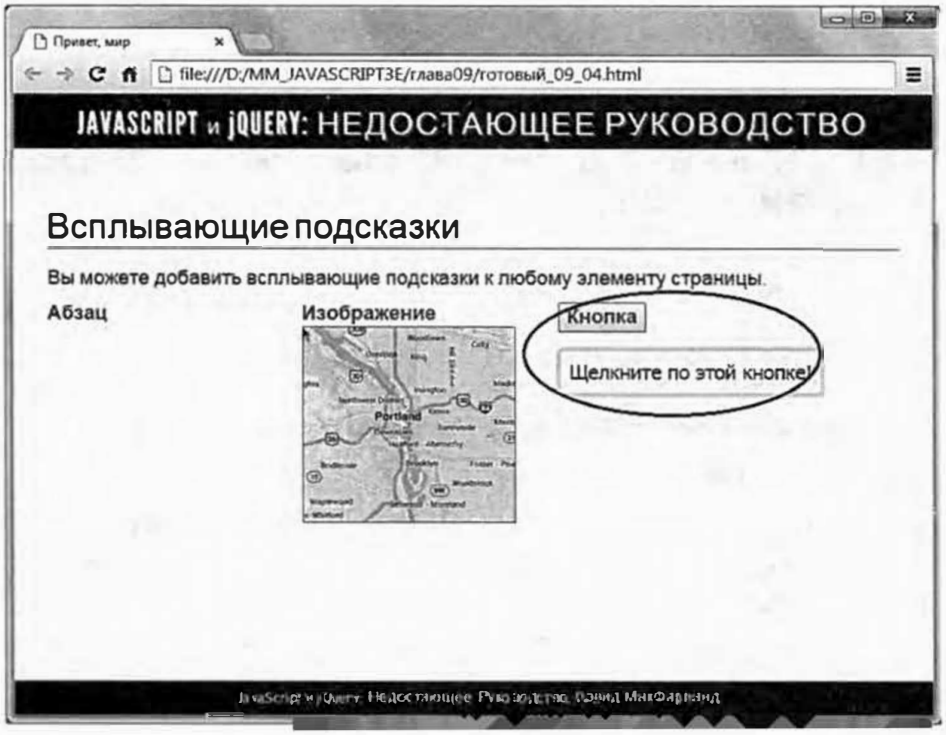

Рис. 9.6. Предоставьте своим посетителям дополнительную информацию с помощью подсказки. Вы можете сделать так, чтобы при наведении указателя мыши на элемент появлялся небольшой текстовый блок (выделен на рисунке). Всплывающие подсказки являются отличным способом объяснить, к чему приведет щелчок по значку, отобразить название изображения или даже фотографии и другой HTML-контент в ответ на наведение указателя мыши на элемент веб-страницы

3. **Выберите элементы и примените функцию `tooltip()`:**

```
$(document).ready(function() {
$('[title]').tooltip();
}); // окончание ready
```

В данном коде вы используете простой селектор атрибута (см. раздел «Специальные селекторы» главы 4), чтобы найти все элементы

с атрибутом `title`, а затем примените к ним функцию `tooltip()`. Вот и все. Плагин jQuery UI позаботится обо всем остальном и создаст привлекательные всплывающие подсказки для каждого элемента с атрибутом `title`.

При желании вы можете добавить конкретики. Например, если вы не хотите создавать подсказки для всех элементов с атрибутом `title`, то можете добавить класс, например, `tooltip`, к каждому тегу, а затем использовать селектор класса, чтобы добавить подсказки только к этим элементам:

```
$('.tooltip').tooltip();
```

Быстрое добавление всплывающих подсказок на практике

Плагин jQueryUI предусматривает самый простой способ добавления всплывающих подсказок к элементам веб-страницы (и ваши посетители будут за это вам благодарны):

1. **В текстовом редакторе откройте файл *09_04.html*.**

   Файлы CSS и JavaScript плагина jQuery UI уже добавлены на эту веб-страницу. Вы можете начать с добавления к HTML-элементам атрибутов `title`, которые станут текстом всплывающих подсказок.

2. **Найдите абзац `<p>Абзац</p>` и добавьте в качестве заголовка текст «Я абзац.»:**

   ```
   <p title="Я абзац.">Абзац</p>
   ```

   Вы можете добавить заголовок к любому элементу в теле веб-страницы. (Вы также можете проявить больше творчества при выборе названий.)

3. **Найдите тег изображения `<img src="images/map.png">` и добавьте в качестве заголовка текст «Я карта!»:**

   ```
   <img src="images/map.png" title="Я карта!">
   ```

   Еще один элемент, и работа с HTML-кодом будет закончена.

4. **Найдите абзац `<button>Кнопка</button>` и добавьте в качестве заголовка текст «Щелкните по этой кнопке!»:**

   ```
   <button title=" Щелкните по этой кнопке!"> ↵
   Кнопка</button>
   ```

Теперь приступим к коду jQuery.

5. **В функции `$(document).ready()` добавьте фрагмент `$('[title]').tooltip();`, чтобы код выглядел следующим образом:**

```
$(document).ready(function() {
$('[title]').tooltip();
}); // окончание ready
```

На этом все. Фрагмент `$('[title]')` выбирает элементы с атрибутом `title`, а фрагмент `.tooltip()` добавляет к ним всплывающие подсказки. Сохраните страницу и просмотрите ее в браузере. Наведите указатель мыши на абзац, изображение и кнопку, чтобы увидеть результаты (9.6). Итоговая версия кода находится в файле *готовый_09_04.html* в папке *глава09*.

Параметры всплывающих подсказок

Как и диалоговое окно, подсказка просто появляется на экране при его открытии. Однако у вас есть несколько способов изменить принцип работы подсказки. Как и виджет Dialog (см. раздел «Создание диалоговых окон с сообщениями» ранее в этой главе), всплывающие подсказки могут принимать объект (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), содержащий свойства, определяющие поведение подсказки. Далее перечислены самые полезные из них:

- **`show`**. Свойство `show` позволяет анимировать процесс появления всплывающей подсказки. Оно работает подобно свойству `show` для диалоговых окон, описанному в разделе «Настройка свойств диалогового окна» ранее в этой главе. Другими словами, для создания интересной визуальной составляющей вы можете сделать так, чтобы подсказка медленно проявлялась, скользила или возникала с использованием одного из множества эффектов jQuery UI (см. раздел «Эффекты jQuery UI» главы 12).
- **`hide`**. Работает подобно свойству `show`, но контролирует процесс исчезновения подсказки.
- **`track`**. Укажите для этого свойства значение `true`, и подсказка будет следовать за указателем мыши по экрану (при условии, что указатель остается наведенным на HTML-элемент, который ее вызывает):

```
track: true
```

Движущаяся подсказка определенно будет заметной, однако она может отвлекать внимание, а текст в ней может быть трудночитаемым.

- **tooltipClass**. Если вы хотите добавить имя класса для ваших подсказок, вы можете предоставить значение этого свойства:

  ```
  tooltipClass: 'tooltip'
  ```

  Это значение позволяет дополнить обычные тематические стили jQuery UI для всплывающих подсказок пользовательскими стилями с помощью созданного вами класса.

- **position**. Это свойство принимает объект position (см. врезку ранее в этой главе) jQuery UI и определяет местоположение подсказки относительно целевого элемента (элемент страницы, на который посетитель наводит указатель мыши, чтобы увидеть подсказку).

ПРИМЕЧАНИЕ

Будьте осторожны со свойством `position`. Всплывающие подсказки исчезают при наведении на них указателя мыши. Если вы расположите подсказку над HTML-элементом, связанным с ней, то можете столкнуться с ситуацией, при которой подсказка так и не появится. Это связано с тем, что наведение указателя мыши на элемент, открывающий подсказку, означает наведение указателя на саму подсказку, что приводит к ее закрытию!

Вы можете использовать эти параметры, передавая их в литерале объекта функции `tooltip()`. Например, вы хотите, чтобы подсказки медленно проявлялись, взрывались при исчезновении и следовали за указателем мыши. В этом случае вы вызываете функцию `tooltip()` так:

```
$('[title]').tooltip({
show: true,
hide: 'explode',
track: true
});
```

Использование в подсказке HTML-контента

Как видите, процесс добавления всплывающей подсказки с помощью плагина jQuery UI прост. Но что, если вам нужно, чтобы подсказка со-

держала больше контента, например, фотографию или несколько абзацев текста? Вы не можете просто поместить HTML-код непосредственно в атрибут элемента `title`, поскольку это не является допустимым и испортит вашу страницу. Для более объемных фрагментов контента виджет Tooltip предоставляет возможность указать другой источник содержимого. Свойство `content` позволяет указать содержимое, которое вы хотели бы добавить в подсказку.

Вы можете использовать это свойство несколькими способами. Самый простой заключается в передаче строки с HTML-кодом, который должен отображаться во всплывающей подсказке. Например, у вас есть ссылка, указывающая на страницу с информацией о вас:

```
<a href="coolest_person_on_earth.html" id="me">Обо мне</a>
```

Если на вашем сайте есть ваша фотография, то вы можете отобразить ее в подсказке при наведении указателя мыши на ссылку:

```
$('#me').tooltip({
content: '<img src="me.png" alt="Посмотрите на ↵
меня!">' });
```

Фрагмент `$('#me')` выбирает ссылку, а часть `.tooltip()` добавляет к этой ссылке подсказку. Свойство `content` определяет HTML-контент, который должен появиться внутри этой подсказки, в данном примере это элемент `img`.

Другой способ добавления HTML-кода в подсказку заключается в том, чтобы добавить HTML-код на страницу, а затем использовать свойство `content`, чтобы поместить его во всплывающей подсказке. Как это работает? Если вы добавляете на страницу HTML-код, он отображается в веб-браузере, а вам нужно, чтобы он отображался только во всплывающей подсказке. Для этого можно было бы использовать jQuery-метод `hide()`, чтобы скрыть HTML-код из поля зрения, а затем использовать функцию `tooltip()`, чтобы взять этот скрытый HTML-код и отобразить его в виде всплывающей подсказки.

Однако есть другой, более простой способ, к которому начали прибегать многие программисты JavaScript: вы можете создать шаблон, поместив HTML-код между элементами `<script>`. Поскольку браузер рассматривает содержимое элементов `script` в качестве кода JavaScript, он не будет отображать HTML-код. Однако вы можете использовать jQuery

для получения доступа и использования этого HTML-кода во всплывающей подсказке! Например, вы хотите поместить в подсказку элемент h2 и неупорядоченный список. Вы можете создать шаблон с этим HTML-кодом между тегами элемента script следующим образом:

```
<script id="tooltipTemplate" type="text/template">
<h2>Обо мне</h2>
<ul>
<li>Я потрясающий.</li>
<li>Я чищу зубы 3 раза в день</li>
<li><img src="me.png" alt="Посмотрите на меня!"></li>
</ul>
</script>
```

Поскольку вы будете использовать его для выбора шаблона, идентификатор очень важен, однако это необязательно должен быть tooltipTemplate. Используйте любой идентификатор, какой хотите. Точно так же часть type="text/template" не является обязательной, однако многие программисты, которые используют способ с шаблоном, добавляют атрибут type, чтобы уточнить назначение тегов элемента script. Также обычной практикой является помещение этих шаблонов в нижней части страницы перед закрывающим тегом </body>.

После создания шаблона вы можете выбрать его и передать содержащийся в нем HTML-код виджету Tooltip:

```
$('#me').tooltip({
content: $('#tooltipTemplate').html();
});
```

Как вы знаете из раздела «Добавление содержимого на страницу» главы 4, метод html() библиотеки jQuery может извлечь HTML-код из выборки. В данном примере вы выбираете тег <script>, который определяет HTML-код для шаблона, а затем помещаете в него HTML-код. Плагин jQuery UI затем использует этот HTML-код в качестве содержимого для всплывающей подсказки.

Добавление в подсказку HTML-кода на практике

Создание форматированной с помощью HTML-кода подсказки немного сложнее по сравнению с добавлением обычной текстовой, однако

благодаря использованию элемента `script`, это по-прежнему довольно легко.

1. **В текстовом редакторе откройте файл *09_05.html*.**

   Данная веб-страница уже связана с файлами CSS и JavaScript плагина jQuery UI. Следующим шагом является добавление HTML-кода, который должен отобразиться в подсказке.

2. **В нижней части страницы после комментария `<!-- поместите шаблон здесь -->` добавьте следующий HTML-код:**

```
<script id="contactInfo" type="text/template">
<p> Наш телефон 555-555-5555</p>
<p><img src="images/map.png" title="Я карта!"></p>
</script>
```

   Вы используете технику, описанную в предыдущем разделе, чтобы скрыть HTML-код в элементах `script`. Браузер не будет отображать этот HTML-код, однако вы все равно можете получить к нему доступ и использовать его с jQuery.

3. **Внутри функции `$(document).ready()` добавьте следующий код:**

```
$('#contact').tooltip({
content: $('#contactInfo').html();
}); // окончание tooltip
```

   Этот код должен показаться вам знакомым: он выбирает элемент на странице (в данном случае элемент `p` с идентификатором `contact`). Затем создается подсказка с указанным содержимым. Фрагмент `$('#contactInfo')` выбирает элемент `script`, который вы добавили в предыдущем шаге, а часть `.html()` извлекает HTML-код.

   Сохраните файл и откройте его в веб-браузере. Наведите указатель мыши на слово «Контакты», и вы увидите всплывающую подсказку, изображенную на рис. 9.7. Итоговая версия этого кода находится в файле *готовый_09_05.html* в папке *глава09*.

Добавление панелей с вкладками

Иногда процесс создания веб-страницы может показаться борьбой за внимание посетителя. Вам может потребоваться отобразить так много информации, что страница становится длинной, переполненной и труд-

но читаемой. Одним из решений в данном случае является использование панели с вкладками, позволяющие разделить контент на отдельные подстраницы, между которыми посетитель может переключаться, щелкая по той или иной вкладке. На таких сайтах электронной коммерции, как Best Buy (рис. 9.8), такая техника используется постоянно. Когда информация разделена на вкладки, посетители могут найти все, что им нужно знать, например, технические характеристики продукта, отзывы и варианты покупки, не будучи перегруженными слишком большим объемом информации.

Рис. 9.7. Всплывающие подсказки не ограничиваются только тем, что вы можете поместить в атрибуте элемента `title`. Вы можете поместить в подсказку любой HTML-код, включая фото, текст и видео. Однако имейте в виду, что при наведении указателя мыши на подсказку она исчезнет, поэтому не стоит включать в нее то, с чем должен взаимодействовать пользователь, например, ссылки или элементы формы. Для такого типа интерактивности вам лучше использовать диалоговое окно (см. раздел «Создание диалоговых окон с сообщениями» ранее в этой главе)

Панели с вкладками jQuery UI (как и другие виджеты) легко реализовать. Ключ заключается в том, как вы структурируете свой HTML-код. Плагин jQuery UI подразумевает некоторые особенности, касающиеся кода HTML для панели с вкладками, однако, по сути, вам необходимо включить три компонента:

- **Контейнер `div`.** Вся коллекция вкладок и панелей должна быть заключена в элементе-контейнере. Это необязательно должен быть элемент `div`, однако чаще всего используется именно он. Вы выбираете этот элемент с помощью jQuery, и затем он сообщает плагину jQuery UI, где найти панели с вкладками. Предоставьте элементу идентификатор, чтобы иметь возможность выбрать его.

Рис. 9.8. Панели с вкладками часто используются на сайтах, на которых требуется представить большой объем информации. На сайтах электронной коммерции такие панели применяются на страницах с описанием товаров, чтобы не перегружать потенциальных покупателей большим объемом данных, но предоставить им доступ к нужной информации

СОВЕТ

Если вы планируете создать на одной странице более одного набора панелей с вкладками, то можете использовать одно и то же имя класса для каждого контейнера `div`, например, `<div class="tabbedPanels">`. Затем вы можете выбрать все эти элементы `div` сразу с помощью фрагмента `$('.tabbedPanels')` и создать все наборы панелей с вкладками на странице, используя следующую строку кода:

```
$('.tabbedPanels').tabs();
```

- **Вкладки.** Для вкладок используйте неупорядоченный или упорядоченный список. Каждая вкладка представлена одним элементом `li`. Внутри этого элемента `li` вы также должны включить элемент `a` со значением `href`, которое указывает на идентификатор соответствующей панели. Например, вы включили три вкладки, указывающие на три панели. HTML-код для этих вкладок может выглядеть следующим образом:

```
<ul>
<li><a href="#details">Описание товара</a></li>
<li><a href="#reviews">Отзывы</a></li>
<li><a href="#order">Заказать</a></li>
</ul>
```

- **Панели.** Каждая панель представляет собой отдельный блочный HTML-элемент. Чаще всего это элемент `div`, однако вы можете использовать `article`, `section` или любой другой блочный элемент. Включите идентификатор для этого элемента `div`, который соответствует # ссылки в панели. Например:

```
<div id="details">
<!-- HTML-код для панели помещается здесь -->
</div>
```

Идентификатор имеет большое значение. Плагин jQuery UI использует его, чтобы связать вкладку с соответствующей панелью. Кроме того, это полезно для браузеров без JavaScript, поскольку этот именной якорь позволяет перейти от ссылки в неупорядоченном списке к соответствующему элементу `div` на странице. Внутри элемента панели вы можете поместить любой HTML-код, например, изображения, текст, списки, видео и т. д. HTML-код панели будет отображаться только при щелчке по вкладке.

Далее приведена простая структура HTML-кода полной панели с вкладками:

```
<div id="tabbedPanel">
<ul>
<li><a href="#details">Описание товара</a></li>
<li><a href="#reviews">Отзывы</a></li>
<li><a href="#order">Заказать</a></li>
</ul>
<div id="details">
<!--HTML-код для панели 1 размещается здесь -->
</div>
<div id="reviews">
<!-- HTML-код для панели 2 размещается здесь -->
```

```
</div>
<div id="order">
<!-- HTML-код для панели 3 размещается здесь -->
</div>
</div>
```

Вы можете выбрать в качестве идентификатора любое имя. Вам необязательно использовать в качестве идентификатора элемента-контейнера `tabbedPanel`, кроме того, вы можете назвать панели как угодно вместо `#details`, `#reviews` и `#order`.

Просто помните, что выбранный идентификатор панели должен совпадать со ссылкой в соответствующей вкладке — в элементе `li`.

ПРИМЕЧАНИЕ

Навигация по панелям с вкладками jQuery UI может осуществляться с помощью клавиатуры. Посетители могут нажать клавишу →, чтобы открыть следующую панель справа, и клавишу ←, чтобы открыть панель слева.

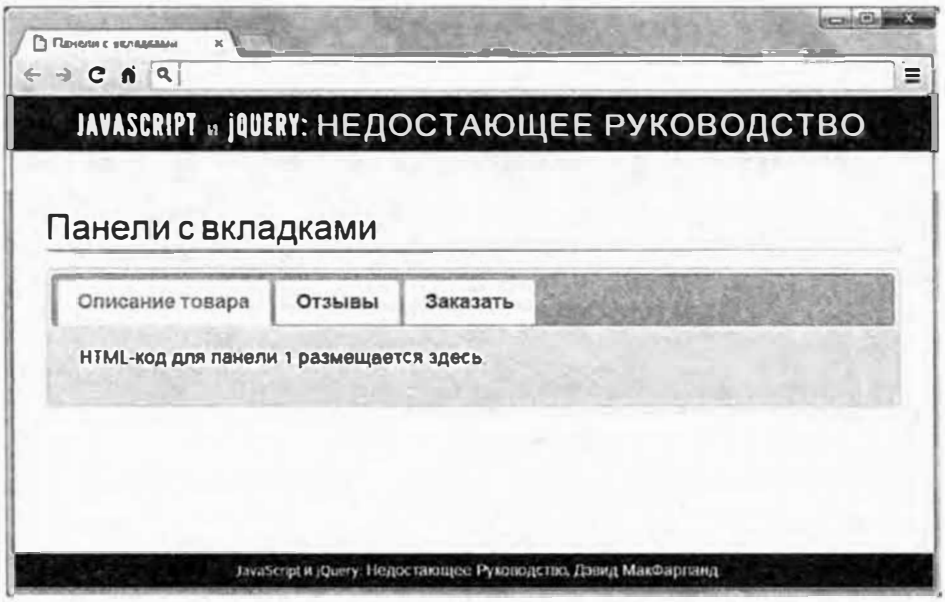

Рис. 9.9. Создать панель с вкладками с помощью плагина jQuery UI действительно очень просто: все, что вам потребуется — это немного кода HTML и JavaScript. Если вам не нравится внешний вид панели с вкладками, вы можете изменить его с помощью приложения jQuery UI ThemeRoller (см. раздел «Знакомство с приложением ThemeRoller» главы 11)

Чтобы превратить этот HTML-код в панели с вкладками, вы выбираете контейнер, а затем вызываете функцию `tabs()` следующим образом:

```
$('#tabbedPanels').tabs();
```

Это приведет к созданию простой панели с вкладками, изображенной на рис. 9.9.

Параметры панели с вкладками

Как и другие виджеты jQuery UI, виджет Tabs предусматривает множество параметров для настройки принципа его работы. Чтобы изменить параметры панели с вкладками, просто передайте литерал объекта функции `tabs()`, содержащей имя свойства и значение, которое вы хотите для него указать. Далее перечислены некоторые из этих параметров:

- **`show` и `hide`**. Эти два свойства контролируют то, как панели появляются и исчезают с экрана. Они принимают те же значения, что и одноименные свойства виджета Dialog. Например, чтобы панель скользила при открытии и закрытии, вам нужно передать функции `tabs()` следующие два свойства:

  ```
  show: 'slideDown',
  hide: 'slideUp'
  ```

- **`active`**. Обычно при загрузке страницы, содержащей панели с вкладками, выбирается первая вкладка и отображается первая панель. Однако вы можете отобразить сначала вторую или последнюю панель. Для этого вы можете использовать свойство `active`:

  ```
  active: 1
  ```

  Как и в случае с массивами JavaScript (см. раздел «Массивы» главы 2), порядковые номера панелей начинаются с 0, поэтому передача функции `active` значения `1` приводит к открытию второй панели. Если вместо числа вы передаете этому свойству значение `false` — `active: false`, вы скроете все панели. Панель откроется только тогда, когда посетитель щелкнет по одной из вкладок. Учтите, что этот параметр работает только при указании значения `true` для функции `collapsible`.

- **`collapsible`**. Укажите для свойства `collapsible` значение `true`, если вы хотите, чтобы пользователи могли скрыть все панели. Обыч-

но по крайней мере одна панель видна постоянно, однако, когда для этого свойства указано значение `true`, при щелчке по вкладке уже открытой панели сама панель полностью закрывается, и вы видите только вкладки. Вы можете использовать этот параметр, когда пространство на экране очень ограничено, однако, как правило, так не делается, и большинству посетителей вашего сайта такое положение вещей может показаться незнакомым. Тем не менее вы должны указать для этого свойства значение `true` при присвоении значения `false` свойству `active` (см. выше), чтобы скрыть все панели при загрузке страницы.

- **`event`**. При щелчке по вкладке открывается связанная с ней панель. По крайней мере, так происходит при нормальной работе. Если вы хотите, чтобы панель вызывалась другим событием, установите свойство `event` с именем нужного события (см. раздел «События мыши» главы 5). Например, чтобы панель появлялась при наведении на вкладку указателя мыши, вы можете использовать следующий код:

  ```
  event: 'mouseover'
  ```

  Тем не менее с данным параметром следует быть осторожным. Люди привыкли к определенным правилам во Всемирной паутине, наиболее распространенное из которых подразумевает то, что если вы щелкните по элементу веб-страницы, то что-то произойдет. Если людям приходится дважды щелкнуть по вкладке (событие `dblclick`), чтобы открыть панель, то они могут так и не понять, как получить доступ к ее содержимому.

- **`heightStyle`**. Свойство `heightStyle` определяет высоту каждой панели и может принимать одно из трех возможных значений: `content`, `auto` и `fill`. Обычно используется значение `content`, при котором высота панели определяется ее содержимым. Если в одной панели содержится много абзацев текста, а в другой — только одна фраза, то при переключении между панелями изменяется общая высота виджета. При существовании по-настоящему большой разницы в объеме содержимого ваших панелей этот визуальный эффект «йо-йо» может показаться вашим посетителям раздражающим.

Значение `auto` задает одну и ту же высоту для каждой панели, основываясь на панели с наибольшим объемом контента. Оно предотвращает изменение высоты группы панелей, но также означает, что в нижней части панели с небольшим объемом содержимого будет много пустого

места. Наконец значение `fill` заставляет группу панелей заполнить доступную область родительского элемента. Обычно оно создает много пустого места на каждой панели и приводит к трате ценного пространства экрана, поэтому его использования лучше избегать.

Добавление панелей с вкладками на практике

В этом руководстве вы познакомитесь с процессом добавления на страницу панели с вкладками. Самая сложная часть заключается в создании HTML-кода. Код JavaScript прост:

1. **В текстовом редакторе откройте файл *09_06.html*.**

   Файлы CSS и JavaScript плагина jQuery UI уже присоединены к этой веб-странице. Вам не придется набирать весь HTML-код, необходимый для создания панелей с вкладками. (Нужные HTML-компоненты описаны в разделе «Добавление панелей с вкладками» ранее в этой главе). Тем не менее в данном фрагменте кода кое-чего не хватает. В частности, идентификатор для контейнера панели с вкладками и ссылок, которые указывают на вкладки в панелях.

2. **В HTML-коде страницы найдите элемент `<h1>Панель с вкладками</h1>`. В тег `<div>`, следующим после заголовка, добавьте код `id="tabContainer"`:**

   ```
   <h1>Панель с вкладками</h1>

   <div id="tabContainer">
   ```

   Вкладки jQuery UI применяются к элементу, который содержит вкладки и панели, обычно к элементу `div`. Добавление идентификатора дает вам возможность выбрать элемент `div` с помощью jQuery и применить функцию `tabs()`. Теперь давайте добавим ссылки.

3. **Найдите элемент неупорядоченного списка (`ul`) под открывающим тегом `<div>`, который вы только что отредактировали. Для каждого тега `<a>` добавьте ссылку `#`:**

   ```
   <li><a href="#panel1">Вкладка 1</a></li>

   <li><a href="#panel2">Вкладка 2</a></li>

   <li><a href="#panel3">Вкладка 3</a></li>
   ```

   Это так называемые ссылки привязки — ссылки, которые указывают на другие разделы страницы с соответствующим идентификатором.

Другими словами, первый элемент `li` связывает элемент `div` с первой панелью, второй элемент `li` — со второй панелью и т. д. Чтобы эти ссылки работали, необходимо добавить соответствующие идентификаторы к элементам `div` панели.

4. **Найдите элемент `div` под элементом `ul`, который вы только что отредактировали (для облегчения поиска воспользуйтесь HTML-комментарием `<!-- Панель 1 -->` над элементом `<div>`). Добавьте в этот тег `<div>` фрагмент `id="panel1"`:**

```
<!-- Панель 1 -->
<div id="panel1">
```

Вам также нужно будет это сделать и для остальных панелей.

5. **Повторите действие в шаге 4 для оставшихся двух элементов `div` панели (их также можно найти по HTML-комментариям).**

Убедитесь в том, что вы присвоили каждому элементу `div` идентификатор, который соответствует ссылке, добавленной в шаге 3. Например, тег `<div>` второй панели должен иметь вид `<div id="panel2">`. Теперь давайте превратим эти элементы в панели с вкладками.

6. **Внутри функции `$(document).ready(function()` выберите контейнер и вызовите функцию `tabs()`:**

```
$(document).ready(function() {
$('#tabContainer').tabs();
}); // окончание ready
```

Сохраните файл и просмотрите его в браузере. Страница должна выглядеть так, как показано на рис. 9.10. Если это не так, проверьте консоль браузера (см. раздел «Отслеживание ошибок» главы 1) на предмет наличия ошибок JavaScript. Если вы ни одной не видите, проверьте корректность имен идентификаторов для элементов `div` контейнера и панелей.

Далее вы добавите некоторые эффекты переходов, сделав так, чтобы панели медленно проявлялись при открытии и постепенно исчезали при закрытии.

7. **Добавьте следующий литерал объекта (выделен полужирным шрифтом) в функцию `tabs()`:**

```
$(document).ready(function() {
```

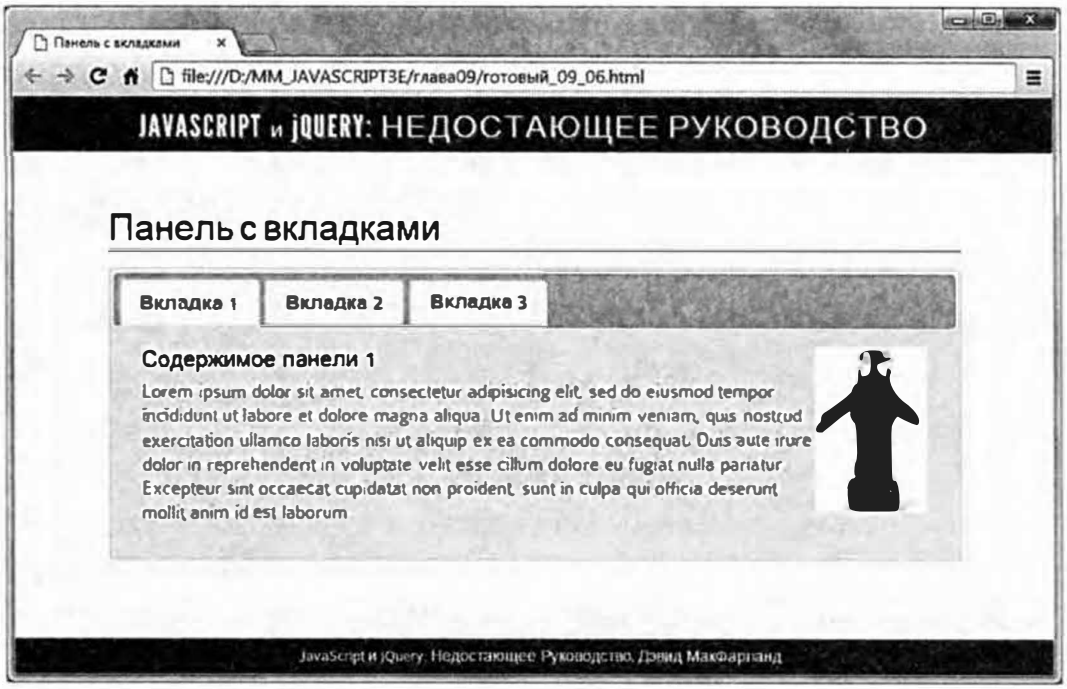

Рис. 9.10. Окончательный вид веб-страницы, содержащей панель с тремя вкладками

```
$('#tabContainer').tabs({
show: 'fadeIn',
hide: 'fadeOut'
});
}); // окончание ready
```

Этот фрагмент кода применяет эффект перехода, наблюдаемый при переключении между вкладками. Вы можете попробовать использовать другие эффекты, например, `'slideDown'` и `'slideUp'`. Сохраните страницу и просмотрите ее в браузере.

Панели с вкладками работают хорошо, однако существует одна проблема: страница всегда открывается либо на первой вкладке, либо на той, которую вы указали с помощью свойства `active` (см. раздел «Параметры панели с вкладками» ранее в этой главе). Но что, если вы захотите отправить по электронной почте ссылку на страницу, и вам нужно, чтобы при переходе по этой ссылке открывалась конкретная вкладка? Например, представьте, что вы сотрудник службы по работе с клиентами, и клиент спрашивает о технических характеристиках продукта, который продает ваша компания. В панели с вкладками есть страница с техническими характеристиками, однако проблема в том, что при загрузке страницы всегда открывается вкладка «Описание товара».

Что, если бы вы могли просто отправить ссылку, например, *mycompany.com/productA.html#specs*, при переходе по которой открывалась бы вкладка с техническими характеристиками. Вы можете это сделать, если добавите магию JavaScript. Секрет заключается в выборе фрагмента `#specs` из URL-адреса и его использовании в качестве элемента-триггера для панели.

8. **После функции `tabs()` добавьте новую строку и введите `var hash = location.hash;`.**

   Окно браузера предусматривает то, что называется объектом *location*. Этот объект содержит много информации об URL-адресе текущей страницы, в том числе имя хоста (`location.hostname`), полный URL-адрес (`location.href`) и другие свойства (посетите страницу **developer.mozilla.org/en-US/docs/Web/API/Location** для ознакомления с их полным списком). Свойство `location.hash` возвращает только часть URL-адреса, включающую фрагмент `#`.

   Например, вы посетили страницу *http://mycompany.com/productA.html#specs*. Свойство `location.hash` для этого URL-адреса имеет значение `#specs`. Вы будете использовать свойство `hash` для загрузки панели с соответствующим значением `hash`.

9. **После только что введенного кода добавьте управляющую инструкцию, чтобы итоговый код выглядел следующим образом:**

```
$(document).ready(function() {
$('#tabContainer').tabs({
show: 'fadeIn',
hide: 'fadeOut'
});
var hash = location.hash;
if (hash) {
$('#tabContainer').tabs('load', hash)
}
}); // окончание ready
```

   Этот фрагмент кода сначала проверяет наличие значения `hash`, например, если посетитель посещает страницу *tabs.html*, то никакого значения `hash` не существует, так что вы можете пропустить остав-

шуюся часть кода и просто позволить первой панели загрузиться в обычном режиме. Однако при наличии значения `hash`, например, `#panel1`, выполняется следующий фрагмент кода. Он просто снова выбирает контейнер (`$('#tabContainer')`), вызывает `tabs()` и передает два аргумента. Первый из них, `load` — это команда, встроенная в программу jQuery UI, приказывающая функции `tabs()` загрузить панель. Второй аргумент — `hash` — это панель, которая должна быть загружена, в данном примере это `#panel1`, `#panel2` или `#panel3`.

10. Сохраните страницу и просмотрите ее в браузере.

Должна открыться первая панель. В адресной строке добавьте фрагмент `#panel3` в конце URL-адреса (после *tabs.html*). Перезагрузите страницу.

Должна отобразиться третья панель. (Если этого не происходит, попробуйте скопировать URL-адрес из адресной строки, открыть новую вкладку или окно браузера и вставить адрес.) Итоговая версия данного кода представлена в файле *готовый_09_06.html* в папке *глава09*.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Пользовательские события jQuery UI

В разделе «Что такое события» главы 5 вы узнали о таких событиях браузера, как `click`, `mouseover`, `focus` и `resize`. Эти события встроены в браузеры и срабатывают, когда посетители совершают на странице определенные действия, например, щелкают по ссылке или отправляют форму. События полезны, потому что они позволяют писать программы, которые отвечают на действия, совершаемые на странице.

Программы виджетов jQuery UI предусматривают собственные события, которые несколько отличаются от привычных для вас событий браузера. События виджетов представляют собой просто моменты создания, исполнения или завершения компонента виджета.

Например, виджет Tabs предусматривает событие `beforeActivate`, которое позволяет выполнять код непосредственно перед тем, как плагин jQuery UI отобразит скрытую панель. Это может пригодиться, если вам нужно выполнить другие действия при каждом щелчке по вкладке. Например, вы можете применять пользовательское событие `beforeActivate` для обновления URL-адреса в адресной строке

браузера при каждом щелчке по вкладке, чтобы URL-адрес добавлял фрагмент # соответствующей панели к имени файла, например, `tabs.html#panel3`. При использовании в сочетании с управляющей инструкцией, подобной добавленной в шаге 9 предыдущего руководства, вы можете предоставить пользователям возможность добавить в закладки пользовательский URL-адрес, который бы отображал нужную панель при каждой загрузке закладки.

Чтобы сделать это, вы добавляете свойство `beforeActivate` к функции `tabs()` с функцией в качестве значения следующим образом:

```
$('#tabContainer').tabs({
beforeActivate: function(evt) {
location.hash=$(evt.currentTarget). attr('href'); }
});
```

В этом фрагменте кода происходит очень многое. Если в двух словах, то вы находите атрибут `href` вкладки, по которой щелкнул посетитель (`#panel`, `#panel2` или `#panel3`), и сохраняете его в свойстве `hash` данного объекта `location` (см. шаг 8 предыдущего руководства). Вы можете найти рабочую копию этого кода в файле *готовый_09_06_02.html* в папке *глава09*.

Как и виджет Tabs, каждый виджет jQuery UI предусматривает множество способов выполнения различных действий при создании, изменении или уничтожении виджета. Программисты называют их *крючками*, поскольку они позволяют вам прикреплять свой собственный программный код к уже существующему в jQuery UI. Это сложная, но интересная для исследования тема.

Лучший способ узнать об этих пользовательских событиях — это посетить страницу API для каждого виджета (API расшифровывается как Application programming interface — Интерфейс программирования приложений и представляет собой все свойства и функции, к которым программист может получить доступ). В верхней части каждой страницы есть окно QuickNav, где перечислены все пользовательские события, поддерживаемые виджетом. Например, на странице API виджета Dialog (**api.jqueryui.com/dialog/**) перечислены 11 различных событий, которые вы можете использовать в качестве «крючков»!

Вкладки с удаленным содержимым

Плагин jQuery UI позволяет использовать в качестве содержимого панели с вкладками контент других веб-страниц. Другими словами, вместо создания неупорядоченного списка ссылок, связанных с элемен-

тами div в рамках страницы, вы можете создать неупорядоченный список ссылок, указывающих на другие страницы (или на контент, который генерируется веб-сервером). Вы можете использовать этот подход, если содержимое каждой вкладки постоянно меняется (цены на акции, отзывы, сообщения форума). Создавая ссылку на динамически создаваемый контент, например, на информацию, извлекаемую из часто обновляемых баз данных с помощью такой технологии, как PHP, .NET или Ruby On Rails, вы можете быть уверенными в актуальности содержимого панели.

Чтобы загрузить содержимое панели с других страниц или запросов веб-сервера, просто создайте элемент-контейнер div, неупорядоченный список, содержащий ссылки на другие страницы, и вызовите функцию tabs(). Например, вам нужно, чтобы каждая панель включала содержимое, взятое из разных страниц. Эти страницы имеют имена *panel1.html*, *panel2.html* и *panel3.html*. В этом случае вам нужно добавить на страницу следующий HTML-код:

```
<div id="tabContainer">
<ul>
<li><a href="panel1.html">Вкладка 1</a></li>
<li><a href="panel2.html">Вкладка 2</a></li>
<li><a href="panle3.html">Вкладка 3</a></li>
</ul>
</div>
```

Если вы ссылаетесь на динамические данные, вы можете указывать не на веб-страницу, а на контент, генерируемый серверным языком, например, PHP:

```
<div id="tabContainer">
<ul>
<li><a href="reviews.php?id=1298">Последние отзывы ↵
</a></li>
<li><a href="forum.php?id=1298">Обсуждения на форуме ↵
</a></li>
</ul>
</div>
```

Когда вы ссылаетесь на внешние веб-страницы, вам не нужно включать элементы div панели, как в шаге 3 предыдущего руководства. Пла-

гин jQuery UI автоматически создаст эти элементы при создании панели с вкладками. Для создания панели с вкладками просто выберите элемент-контейнер `div` и вызовите функцию `tabs()`:

```
$('#tabContainer').tabs();
```

При выполнении этого кода плагин jQuery UI будет извлекать HTML-код из веб-страницы, которая связана с первой видимой панелью. Например, в случае с вышеприведенным кодом при загрузке страницы jQuery UI загрузит HTML-код из файла *panel1.html* и отобразит его в панели под вкладками. Когда пользователь щелкнет по второй вкладке, плагин jQuery UI запросит HTML-код у второй ссылки и создаст новую панель, в которой этот HTML-код будет помещен.

В большинстве случаев вы можете даже сослаться на такой внешний веб-сайт, как Google, Wikipedia или на сервер, отличный от того, на котором расположена страница, содержащая панель с вкладками. Тем не менее сайты могут блокировать этот процесс, что может предотвратить загрузку такого содержимого, как веб-шрифты, изображения и видео.

У этого подхода существует еще одна проблема: в панели станет отображаться *все* содержимое связанной страницы. Таким образом, если вы создаете ссылку на *целый* HTML-файл с элементами `header` и `title`, а также со ссылками на файлы CSS и JavaScript, то все это также будет загружено. В конечном итоге у вас получится страница в странице. Если вы хотите поместить в панель только фрагмент HTML-контента, то для этого вы можете использовать два подхода.

Самый простой способ заключается в создании фрагментов страницы — HTML-файлов, содержащих *только* те части HTML-кода, которые вы хотите поместить в панели. Вы можете легко создать фрагмент, если загружаете данные с сервера динамически — ваш серверный сценарий должен предоставить только тот контент, который должен отобразиться в панели (то есть за исключением таких фрагментов, как элемент `head`, который требуется для отображения полной веб-страницы).

В качестве альтернативы вы можете позволить панели загрузить всю веб-страницу и извлечь только то содержимое, которое вы хотите отобразить в панели с помощью *пользовательского события* (см. врезку выше). Вот как это работает: панели с вкладками jQuery UI предусматривают событие `load`. Это событие позволяет запускать функцию сразу после того, как плагин jQuery UI загрузит контент из удаленного источника. Вы можете использовать jQuery для нахождения только необходимого

вам содержимого, его извлечения и размещения внутри панели. Этот метод несколько сложен, но он не требует большого объема кода.

Во-первых, вы должны убедиться в существовании способа выбора конкретного содержимого из удаленной страницы. Самый простой способ это сделать заключается в том, чтобы обернуть содержимое, которое должно отобразиться в панели, в тег `<div>` с идентификатором, например, `<div id="panelContent">`. Теперь у вас есть возможность выбрать только это содержимое, исключив ненужный HTML-код, вроде элементов `head` или `title`.

Теперь вам необходимо передать параметр `load` функции `tabs()`. Параметр `load` представляет собой пользовательское событие, и вы предоставляете анонимную функцию (см. раздел «Работа с каждым элементом выборки» главы 4), которая говорит плагину jQuery UI, что делать после получения содержимого с удаленной страницы и его помещения в панель. К моменту срабатывания события `load` плагин jQuery UI уже создаст новую вкладку и вставит в нее HTML-код удаленной страницы. В данный момент новая панель содержит весь дополнительный HTML-код, который вам не нужен. Тем не менее вы можете использовать jQuery для быстрого удаления этого нового HTML-кода и его замены сокращенной версией. Это происходит так быстро, что замена содержимого даже не отображается в браузере.

Вот пример:

```
$('#tabContainer').tabs({
load: function(evt,ui) {
var newHTML =
ui.panel.find('#panelContent').html();
ui.panel.html(newHTML);
}
})
```

Строки 2–5 представляют собой параметр `load`. Плагин jQuery UI предоставляет два фрагмента информации его пользовательским событиям (это `evt` и `ui` в строке 2). Первый фрагмент `evt` — это обычный объект события jQuery. Вы можете использовать любое из свойств и методов событий, описанных в разделе «Объект события» главы 5. В данном случае вас интересует второй аргумент, переданный анонимной функции события `load`: `ui`. Объект `ui` представляет собой обновлен-

ный элемент пользовательского интерфейса. Для панели с вкладками существуют объекты `ui.panel` и `ui.tab`. Объект `ui.panel` представляет вновь созданную панель (элемент `div`, создаваемый плагином jQuery UI при загрузке внешней веб-страницы).

В строке 3 вы создаете новую переменную — `newHTML`, которая будет содержать итоговый контент панели (без такого дополнительного HTML-кода, как элемент `head`). Когда плагин jQuery UI создает новую панель, он загружает весь HTML-контент запрашиваемой страницы, так что вы можете заглянуть в эту панель и получить только нужный HTML-код. Фрагмент `ui.panel.find('#panelContent')` получает HTML-код новой панели и ищет в нем элемент с идентификатором `panelContent` (метод jQuery `find()` описан в разделе «Обход дерева DOM» главы 15). Затем в игру вступает метод `html()` (см. раздел «Выбор элементов страницы: способ jQuery» главы 4) и извлекает HTML-код только из этой конкретной части страницы. Другими словами, вы взяли нужный вам фрагмент HTML-кода и сохранили его в переменной `newHTML`.

ПРИМЕЧАНИЕ

Вы можете найти пример использования отдельных веб-страниц в качестве содержимого панели в файле *удаленные_вкладки.html*, который находится в папке *глава09*. Для исключения ненужного HTML-кода в данном файле также применяется пользовательское событие `load`.

В строке 4 вы просто заменяете HTML-код панели (старый HTML-код с элементами `head`, `title` и другими нежелательными элементами) новым HTML-кодом (который нужен вам для новой панели). Этот код выполняется так быстро, что браузер даже не отображает всей веб-страницы внутри панели, и посетитель увидит только сокращенную версию HTML-кода.

Экономия пространства с помощью аккордеонов

Аккордеоны jQuery UI, как и панели с вкладками, являются еще одним элементом пользовательского интерфейса, позволяющим экономить место. Пример изображен на рис. 9.11.

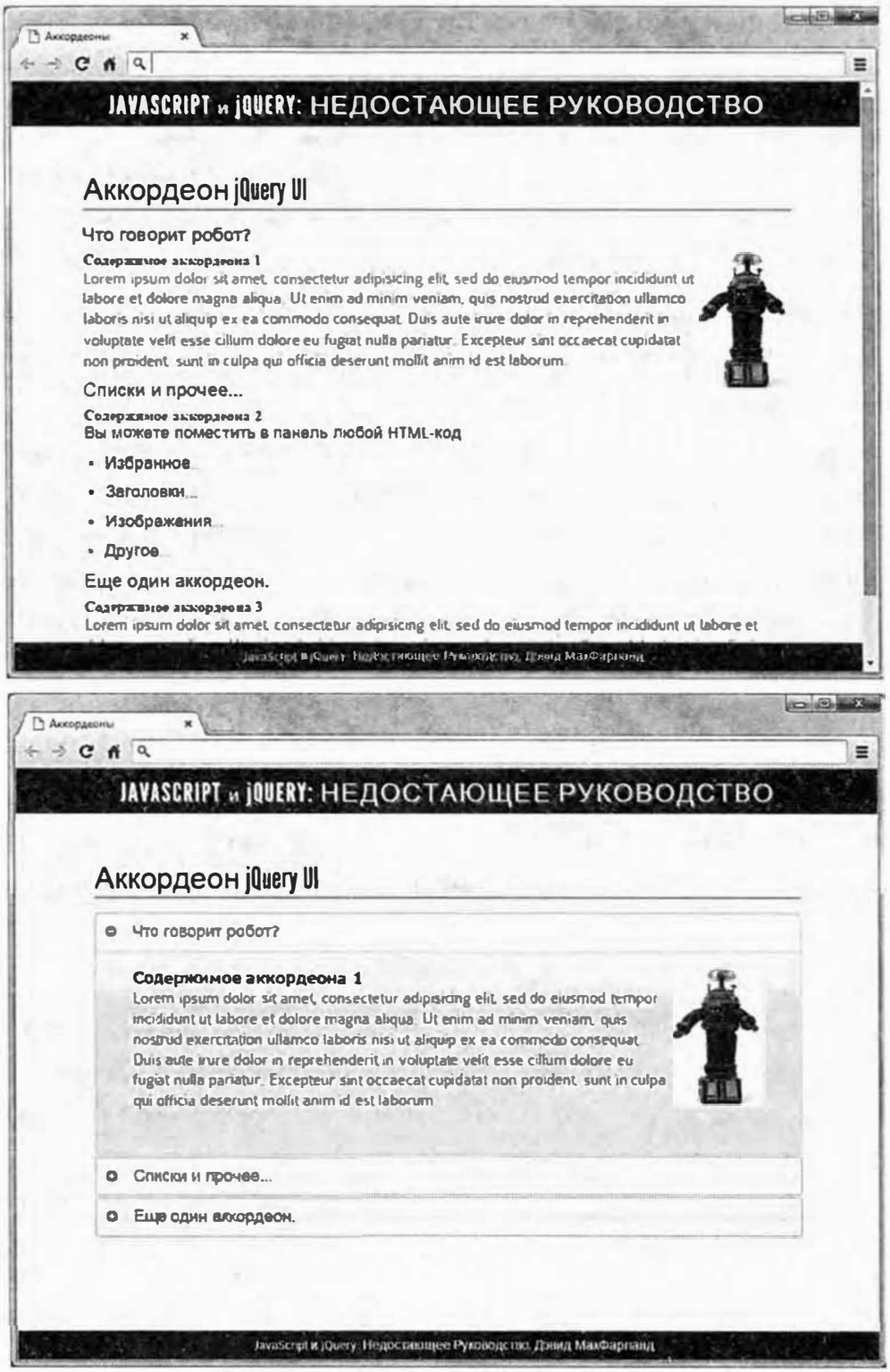

Рис. 9.11. Плагин jQuery UI позволяет превратить простую коллекцию заголовков и элементов `div` (вверху) в интерактивную презентацию, состоящую из складывающихся панелей (внизу)

Однако вместо вкладок в верхней части каждая панель аккордеона имеет интерактивный заголовок, который открывает панель прямо под ним и закрывают панель, которая открыта в данный момент. Другими

словами, в любой момент может быть открыта только одна панель аккордеона.

Как правило, аккордеоны работают так же, как вкладки. Большинство параметров совпадает и функционируют подобно вкладкам, однако структура HTML-кода имеет много отличий. В случае с аккордеонами вам нужны всего три компонента:

- **Элемент `div`, содержащий аккордеон.** Вам нужно выбрать элемент `div` с помощью jQuery, так что добавьте ему идентификатор или класс и примените к нему виджет Accordion.
- **Заголовок, содержащий текст.** Этот заголовок является интерактивным элементом управления, который открывает и закрывает панели аккордеона (например, синий заголовок «Что говорит робот?» на рис. 9.11). Не имеет значения, заголовок какого уровня вы используете — `h2`, `h3` или любой другой. Просто используйте тот же самый уровень заголовка для каждой группы аккордеона. (Технически вы даже не обязаны использовать элемент заголовка — в данном случае подойдет любой блочный элемент, однако заголовки используются чаще всего.)
- **Блочный элемент, следующий сразу после заголовка.** Обычно это элемент `div` с содержимым, которое вы хотите показать и скрыть. Этот элемент `div` должен идти сразу после заголовка.

Заголовок и следующий за ним элемент `div` представляют собой одну часть аккордеона. Чтобы добавить больше элементов аккордеона, добавьте больше пар заголовок/элемент `div`. Например, базовая структура HTML-кода аккордеона, состоящего из трех панелей, выглядит следующим образом:

```
<div id="accordion">
<h3>Триггер для первой панели аккордеона</h3>
<div>
<!—Содержимое первого аккордеона -->
</div>
<h3>Триггер для второй панели аккордеона</h3>
<div>
<!-- Содержимое второго аккордеона -->
</div>
```

```
<h3>Триггер для третьей панели аккордеона</h3>
<div>
<!-- Содержимое третьего аккордеона -->
</div>
</div>
```

После создания HTML-кода и прикрепления файлов CSS и JavaScript библиотеки jQuery и плагина jQuery UI применение виджета Accordion будет сводиться к выбору элемента-контейнера и вызову функции функции `accordion()`:

```
$('#accordion').accordion();
```

Как обычно, плагин jQuery UI выполнит всю тяжелую работу и превратит вашу простую HTML-структуру в аккордеон, изображенный на рис. 9.11. Как и панель с вкладками, аккордеон jQuery UI может принимать объект с множеством параметров, многие из которых работают также как и аналогичные параметры панели с вкладками:

- **`active`**. Параметр `active` работает подобно аналогичному параметру панели с вкладками (см. раздел «Параметры панели с вкладками» ранее в данной главе). Укажите числовое значение, соответствующее панели аккордеона, которая должна открыться при загрузке страницы. Например, чтобы открыть вторую панель, вы можете использовать следующий код:

  ```
  active: 1
  ```

  Как и в случае с массивами, отсчет панелей аккордеона начинается с `0`. Если вы укажете для этого параметра значение `false`, а для свойства `collapsible` — значение `true`, то сможете загрузить страницу с закрытыми аккордеонами.

- **`collapsible`**. Укажите для данного свойства значение `true`, а для свойства `active` — значение `false`, и при загрузке страницы все панели аккордеона будут закрыты. Кроме того, если вы укажете для данного свойства значение `true`, то заголовки аккордеона станут работать подобно переключателю: щелкните по заголовку, и если панель под этим заголовком открыта, то она закроется. Если панель под этим заголовком закрыта в момент щелчка, то она откроется.

- **`animate`**. Обычно при открытии и закрытии панелей аккордеона используется эффект скольжения. Вы можете отменить это поведение

и заставить панели мгновенно открываться и закрываться, указав для данного свойства значение `false`:

```
animate: false
```

Вы также можете предоставить несколько других значений для разных типов поведения. Численное значение сообщает jQuery UI длительность анимации (в миллисекундах). Например, чтобы взбесить посетителей своего сайта, вы может заставить панели аккордеона открываться и закрываться мучительно медленно — в течение 5 секунд:

```
animate: 5000
```

Вы также можете предоставить строку, соответствующую имени функции `easing`. Как говорится в разделе «Управление скоростью анимации» главы 12, функция `easing` контролирует воспроизведение анимации: вы можете сделать так, чтобы воспроизведение анимации начиналось очень медленно, а под конец ускорялось. Например, чтобы использовать функцию `easeInElastic`, вы можете написать свойство `animate` так:

```
animate: 'easeInElastic'
```

Тем не менее любое значение параметра `easing`, кроме того, которое используется по умолчанию, производит ужасный эффект.

- **`event`**. Указывает событие, которое приводит к открытию панели аккордеона. Работает так же, как параметр `event` панели с вкладками, описанный в разделе «Параметры панели с вкладками» ранее в данной главе.
- **`heightStyle`**. Аналогичен одноименному параметру панели с вкладками (см. раздел «Параметры панели с вкладками» ранее в данной главе).
- **`icons`**. Как вы можете видеть на рис. 9.11, плагин jQuery UI прикрепляет маленькие значки слева от заголовков аккордеона. Небольшая стрелка, направленная вниз, отображается в заголовках, относящихся к открытым панелям аккордеона. В заголовках, относящихся к закрытым панелям, отображается стрелка, направленная вправо. Плагин jQuery UI включает большой выбор значков в качестве части каждой темы (полный список вы можете найти на сайте **api.jqueryui.com/theming/icons/**). Вы можете заменить значки, передав параметр `icons` и значение, включающее литерал объекта:

```
icons : {
header: "ui-icon-plus",
activeHeader: "ui-icon-minus"
}
```

Литерал объекта определяет два значка, которые должен использовать плагин jQuery UI. Свойство `header` устанавливает значок, используемый в заголовках, относящихся к закрытым панелям аккордеона, а свойство `activeHeader` определяет значок, используемый для заголовков открытых панелей. В приведенном выше примере символы + и – используются для обозначения различных состояний заголовков.

Как и в случае с другими плагинами jQuery UI, вы можете объединить несколько параметров виджета Accordion для управления его внешним видом и принципом работы. Например, вы хотите изменить событие-триггер так, чтобы при наведении указателя мыши на заголовок открывалась панель аккордеона, а также вы хотите изменить значки, используемые в заголовках. Это можно сделать, вызвав функции функцию `accordion()`:

```
$('#accordion').accordion({
event: 'mouseover',
icons : {
header: 'ui-icon-circle-plus',
activeHeader: 'ui-icon-circle-minus'
}
});
```

Создание аккордеона jQuery UI на практике

Аккордеоны очень похожи на панели с вкладками. Большая часть необходимой работы связана с форматированием HTML-кода. Однако HTML-код для аккордеонов является даже более простым, чем HTML-код для панелей с вкладками. В данном руководстве вы добавите аккордеон в уже созданный HTML-файл:

1. **Сначала посмотрите на HTML-код, чтобы увидеть, что в нем содержится. В веб-браузере откройте файл *09_07.html*.**

   Страница содержит заголовок «Аккордеон jQuery UI» и простой набор заголовков, текста и изображений (см. верхнее изображение на рис. 9.11).

2. **Откройте страницу *09_07.html* в текстовом редакторе и посмотрите на HTML-код, следующий за элементом `h1`.**

   Обратите внимание на элемент `div`, — это контейнер, который содержит элементы аккордеона. Внутри этого элемента `div` вы увидите элемент `h3`, за которым следует еще один элемент `div`. Элемент `h3` представляет собой метку аккордеона, а элемент `div` — панель аккордеона. В HTML-коде для этой страницы присутствуют три пары элементов `h3/div`, поэтому после добавления программного кода у вас будет три панели аккордеона. Сначала вам необходимо дать библиотеке jQuery «крючок» для выбора контейнера аккордеона.

3. **Найдите элемент `div` под элементом `h1` и добавьте идентификатор:**

```
<div id="accordion">
```

   Это весь HTML-код, который вам необходим для добавления аккордеонов на страницу. Теперь пришло время добавить программный код.

4. **Внутри функции `$(document).ready()` выберите контейнер и вызовите функцию `accordion()`:**

```
$(document).ready(function() {
$('#accordion).accordion();
}); // окончание ready
```

   Сохраните файл и просмотрите его в браузере. Страница должна соответствовать нижнему изображению на рис. 9.11. Теперь понимаете, почему это называется «мини-руководством»? Благодаря плагину jQuery UI все очень просто. Однако вы еще не закончили. Вам нужно, чтобы при загрузке страницы все панели аккордеона были закрыты.

5. **Добавьте следующий литерал объекта (выделен полужирным шрифтом) в функцию `tabs()`:**

```
$(document).ready(function() {
$('#accordion').accordion({
active: false,
collapsible: true
});
}); // окончание ready
```

   Просмотрите страницу в браузере, и вы увидите, что изначально все панели аккордеона закрыты. Щелкните по одному из заголовков,

чтобы открыть панель, а затем щелкните по этому же заголовку, чтобы закрыть ее. Панели аккордеона закрываются при щелчке по их заголовку, только если для параметра `collapsible` указано значение `true`. Давайте выберем несколько других значков из большой коллекции jQuery UI **(api.jqueryui.com/theming/icons/).**

6. **Добавьте свойство `icons` в объект с параметрами аккордеона. Значение этого параметра будет являться еще одним литералом объекта с двумя свойствами:**

```
$('#accordion').accordion({
active: false,
collapsible: true,
icons : {
header: 'ui-icon-circle-plus',
activeHeader: 'ui-icon-circle-minus'
}
});
```

Не забудьте добавить запятую после значения `true` в строке 3. Как вы можете видеть, код JavaScript позволяет вкладывать литералы объекта в другие литералы объекта. Иногда такой код может быть несколько трудным для восприятия, однако имейте в виду, что вы можете обращаться с литералом объекта, как и с любым другим объектом — переменной, числом или строкой, и использовать его в качестве единственного значения для свойства переменной или объекта.

7. **Сохраните страницу и просмотрите ее в браузере.**

Страница должна соответствовать нижнему изображению на рис. 9.11. Если это не так, перепроверьте код и сравните его с кодом в шаге 4. Итоговый вариант этого руководства можно найти в файле *готовый_09_07.html* в папке *глава09*.

Добавление меню на страницу

Плагин jQuery UI также включает в себя виджет Selectmenu, который позволяет очень легко превратить неупорядоченный вложенный список ссылок в меню с вылетающими подменю. Этот виджет предназначен для создания вертикальной строки меню, где пункты расположены друг над другом, а подменю отображаются справа (рис. 9.12). Одна-

ко если вы остановитесь на одном наборе подменю, то сможете создать с помощью виджета Selectmenu горизонтальное меню верхнего уровня.

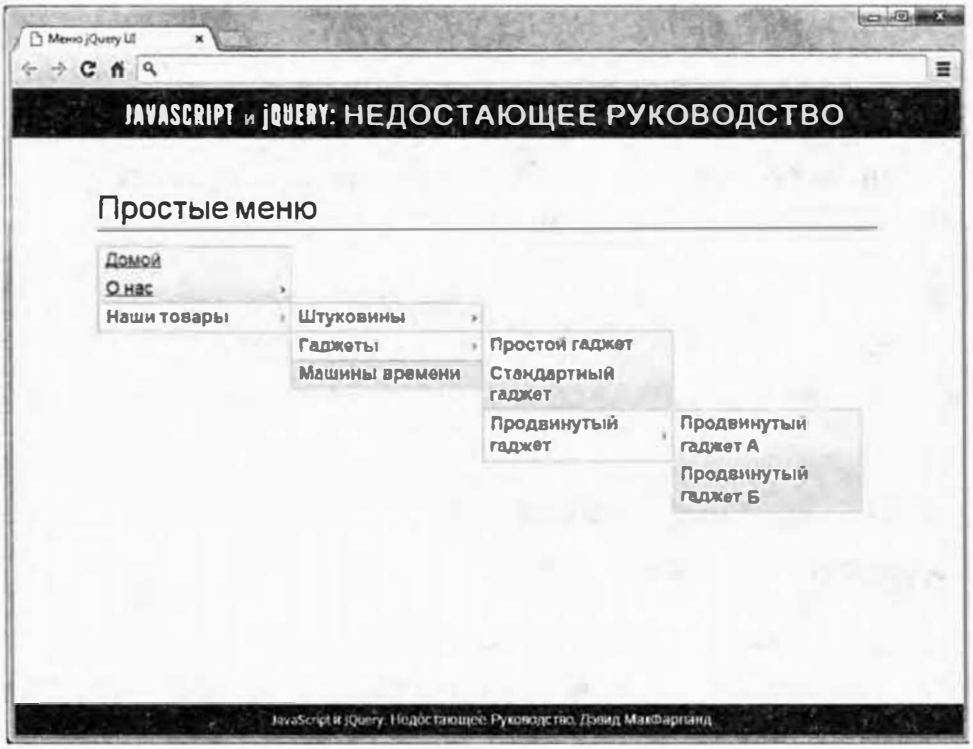

Рис. 9.12. Виджет меню jQuery UI позволяет легко создавать многоуровневое меню, чей внешний вид соответствует остальным виджетам jQuery UI. Он хорошо подходит для добавления меню в веб-приложение, чтобы оно больше походило на настольную программу

Как и в случае со всеми виджетами jQuery UI, меню очень просты в использовании. Трудная часть заключается в структурировании HTML-кода, но даже это не сложно. Вы уже видели такую структуру, когда изучали плагин SmartMenus в разделе «Создание отзывчивого меню навигации» главы 7. Вы начнете с простого неупорядоченного списка ссылок — это кнопки меню верхнего уровня, которые вы видите при загрузке страницы. Если хотите добавить в выпадающее меню к одной из этих кнопок, просто вложите другой неупорядоченный список в тег `<li>` кнопки верхнего уровня. Допустим, далее приведен HTML-код для меню с тремя меню верхнего уровня и подменю, которое отображается при наведении указателя мыши на последний пункт меню:

```
<ul id="mainMenu">
<li><a href="about.html">О нас</a></li>
<li><a href="contact.html">Контакты</a></li>
<li><a href="products.html">Продукты</a>
```

```
<ul>
<li><a href="a.html">Продукт A</a></li>
<li><a href="b.html">Продукт B</a></li>
<li><a href="c.html">Продукт C</a></li>
</ul>
</li>
</ul>
```

Когда HTML-код преобразуется в меню, посетители могут навести указатель мыши на пункт меню «Продукты», чтобы отобразить подменю с тремя вариантами продуктов. Как и в случае с другими виджетами jQuery UI, для элемента, содержащего виджет, следует предоставить идентификатор. В данном случае это первый элемент `ul`, поскольку он является контейнером для всех пунктов меню и подменю.

СОВЕТ

Если вы хотите визуально организовать пункты вашего меню, вы можете добавить разделитель для разделения кнопок подменю. Просто вставьте элемент `li`, содержащий вместо ссылки тире:

```
<li>-</li>
```

Плагин jQuery UI создаст в меню линию (вместо добавления кнопки).

Использовать виджет для создания меню легко:

1. **Прикрепите файлы CSS jQuery UI, JavaScript jQuery и JavaScript jQuery UI.**

   Это основные шаги для использования любого виджета jQuery UI, как вы знаете из раздела «Добавление jQuery UI на веб-страницу» данной главы.

2. **Вставьте неупорядоченный список ссылок и дополнительные неупорядоченные списки для подменю, как описано выше.**

   Следует убедиться, что пункты меню верхнего уровня ссылаются на страницы вашего сайта.

3. **Добавьте код CSS, чтобы ограничить размер кнопок меню и подменю.**

   CSS-код, предусмотренный для меню jQuery UI, не ограничивает ширину кнопок меню, поэтому ваше главное навигационное меню может содержать необычайно широкие кнопки. Чтобы указать раз-

мер кнопок основного меню, создайте стиль `.ui-menu` со свойством `width`:

```
.ui-menu {
width: 10em;
}
```

Класс `ui-menu` применяется автоматически плагином jQuery UI, когда он создает виджет меню. Этот класс применяется к каждому элементу `ul` в меню (в главном меню и в любом подменю).

Вы можете использовать любые единицы измерения — `em`, `px`, `%`, однако будьте осторожны при использовании процентных значений. Подменю принимает свое процентное значение от своего родительского элемента, поэтому каждое следующее подменю будет становиться все тоньше. Чтобы обойти эту проблему, укажите 100% в качестве значения ширины подменю (которое представляет собой элемент `ul` внутри элемента `ul`), чтобы она соответствовала ширине родительского меню. Например:

```
.ui-menu {
width: 25%;
}
.ui-menu .ui-menu {
width: 100%;
}
```

Добавьте этот код CSS в пользовательский CSS-файл вашего сайта, а не в CSS-файл плагина jQuery UI. Если вы решите изменить тему или обновить плагин jQuery UI до более новой версии, то любые изменения, внесенные в CSS-файл jQuery UI, будут потеряны в процессе обновления. Вы узнаете больше о стилизации виджетов jQuery UI в главе 11.

4. **Выберите внешний элемент `ul` и примените функцию `menu()`:**

```
$('#mainMenu').menu();
```

ПРИМЕЧАНИЕ

Вы найдете пример виджета меню в файле *готовый_09_08.html* в папке *глава09*.

Как и в случае с другими виджетами jQuery UI, виджет меню предусматривает несколько параметров, позволяющих контролировать его поведение и внешний вид. Для этого передайте функции `menu()` литерал объекта:

- **`icons`**. Как вы можете видеть на рис. 9.12, плагин jQuery UI отображает небольшие значки справа от любой кнопки меню, которая прикреплена к подменю.

  Этот значок показывает посетителям, что под этой кнопкой есть еще одно меню. Плагин jQuery UI предусматривает большой выбор значков в каждой теме (на сайте **api.jqueryui.com/theming/icons/** вы можете познакомиться с полным списком). Вы можете заменить значки, передав параметр `icons` и значение, которое включает литерал объекта:

```
icons : {
submenu: "ui-icon-circle-triangle-e"
}
```

  К сожалению, можно указать только один значок, поэтому значок, используемый для кнопок в главном меню, также используется на кнопках подменю, которые имеют собственные подменю.

- **`position`**. Параметр `position` определяет положение подменю относительно их родительского элемента. Обычно подменю размещаются непосредственно справа от своей родительской кнопки, однако вы можете изменить это с помощью объекта jQuery UI `position`, описанного в тексте в рамке «Точное позиционирование с помощью плагина jQuery UI» ранее в данной главе. Например, если вы хотите поместить подменю непосредственно под кнопкой, которая его открывает, вы можете использовать данный параметр следующим образом:

```
position : {
my: "center top",
at: "center bottom"
}
```

  Этот код помещает центр и верх подменю `my` в центре и внизу кнопки родительского меню (`at`) . Этот метод полезен для создания выпадающих меню в горизонтальной панели навигации, как описано далее.

Создание горизонтальной панели навигации

Меню jQuery UI не предназначены для создания классической навигационной панели, подобной тем, что вы видите в верхней части большинства веб-сайтов. Если вам нужны такие меню, то лучше воспользоваться плагином SmartMenus, обсуждаемом в разделе «Создание отзывчивого меню навигации» главы 7. Тем не менее с помощью виджета меню вы можете создать горизонтальное меню с одним выпадающим меню, применив небольшой фрагмент кода CSS, как показано на верхнем изображении на рис. 9.13.

Многоуровневые выпадающие меню работают не очень хорошо, поскольку виджет меню jQuery UI помещает все подменю в одном и том же положении относительно родительского элемента. В случае с вертикальным меню (рис. 9.12) виджет работает нормально: подменю выпадают справа от родительской кнопки. Однако в случае с горизонтальным меню первое подменю обычно отображается под родительской кнопкой (рис. 9.13 сверху), а под-подменю отображаются справа от родительского элемента. К сожалению, при использовании плагина jQuery UI, если вы поместите одно подменю под его родительским элементом, то все подменю будут размещены таким же образом, так что в конечном итоге у вас получится путаница, как показано внизу на рис. 9.13.

1. **Чтобы создать горизонтальное меню, начните с неупорядоченного списка ссылок, но добавьте только один уровень вложенных неупорядоченных списков (как в примере HTML-кода в разделе «Добавление меню на страницу» ранее в этой главе).**

   Теперь вам нужно добавить CSS-код, чтобы обеспечить горизонтальное положение кнопок главой панели навигации. Вы должны поместить этот CSS-код в основную таблицу стилей для вашего сайта, а не в CSS-файл jQuery UI.

2. **Добавьте CSS-код, чтобы кнопки меню верхнего уровня отображались друг рядом с другом в горизонтальной панели:**

```
#mainMenu > li {
width: 10em;
float: left;
}
```

   Селектор CSS `#mainMenu > li` выбирает все элементы `li`, которые являются прямыми потомками элемента с идентификатором `mainMenu`. (Предполагается, что вы присвоили элементу-контейнеру

ul имя идентификатора, как в примере кода в разделе «Добавление меню на страницу» ранее в этой главе.

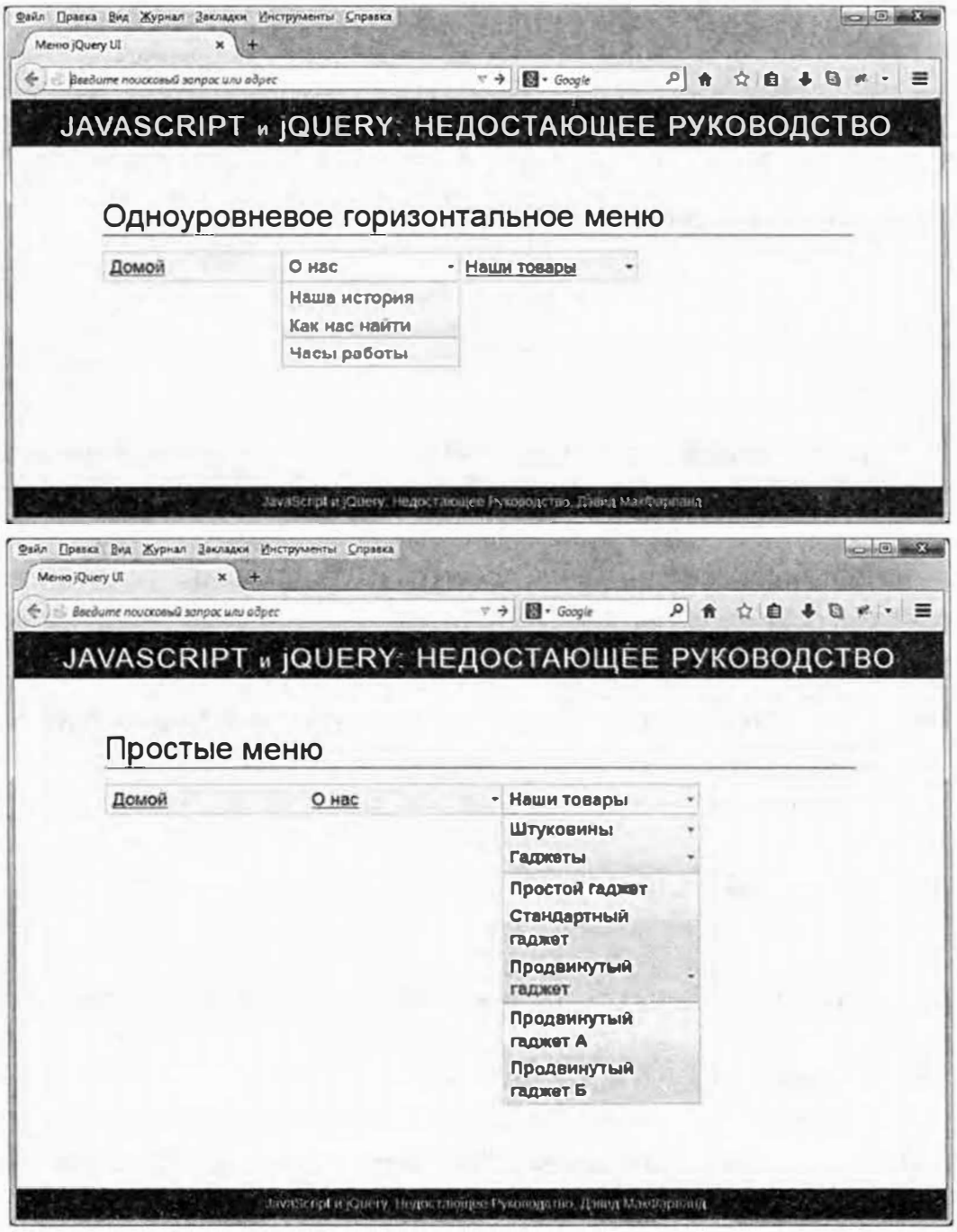

Рис. 9.13. Если вы хотите обеспечить единообразие внешнего вида вашего пользовательского интерфейса с помощью CSS-кода jQuery UI и виджетов на всем вашем сайте, вы можете создать полностью функциональное горизонтальное меню jQuery UI, если вам достаточно только одноуровневого выпадающего меню (верхнее изображение). Виджет Selectmenu jQuery UI не очень подходит для создания многоуровневого горизонтального меню (нижнее изображение). Поскольку все подменю находятся в том же положении относительно своих родительских элементов, они будут перекрывать друг друга. Вы можете найти примеры обоих типов меню в папке *глава09* в файлах *готовый_09_08_02.html* и *готовый_09_08_03.html*

Если вы использовали другой идентификатор для этого элемента, то используйте его). Фрагмент > представляет собой дочерний селектор, выбирающий только те элементы `li`, которые являются непосредственными потомками главных элементов `ul`, поэтому все элементы `li` в подменю будут игнорировать это правило.

Этот стиль задает ширину для каждой кнопки меню, а затем размещает их бок о бок. Теперь вам нужно указать ширину подменю.

3. **Добавьте еще один стиль CSS:**

```
.ui-menu .ui-menu {
width: 10em;
}
```

Этот стиль задает ширину подменю. Плагин jQuery UI добавляет в каждое меню (присваивает элементу `ul`) класс `.ui-menu`, поэтому селектор `.ui-menu .ui-menu` выбирает только те элементы `ul`, которые находятся в другом элементе `ul`. Другими словами, этот стиль относится только к вложенным меню и задает ширину этих подменю.

4. **Наконец, добавьте последний стиль, чтобы исправить проблему с главным меню:**

```
#mainMenu {
float: left;
}
```

Этот стиль исправляет ситуацию *escaping float*, при которой высота родительского элемента схлопывается, когда его дочерние элементы размещаются с использованием свойства `float`. Иными словами, граница и фон главного меню выглядят неправильно, когда ко всем кнопкам применено свойство `float`. Один из способов решения данной проблемы заключается в том, чтобы применить свойство `float` к родительскому элементу.

После окончания работы над кодом CSS вы можете добавить JavaScript.

5. **Вызовите функцию `menu()`, как описано в разделе «Добавление меню на страницу» ранее в этой главе, однако добавьте несколько параметров, чтобы контролировать расположение подменю и значка, используемого в главном меню:**

```
$('#menu').menu({
position: {
my: 'center top',
at: 'center bottom'
},
icons: {
submenu: 'ui-icon-triangle-1-s'
}
});
```

Параметр `position` (описан в разделе «Добавление меню на страницу» ранее в этой главе) контролирует положение подменю. В данном случае он помещает меню непосредственно под кнопкой меню, которая его открывает. Кроме того, виджет меню обычно отображает значок в виде стрелки вправо на кнопках, которые имеют подменю. Тем не менее, поскольку панель меню теперь располагается горизонтально, а подменю открывается под главной панелью, вы должны использовать значок в виде стрелки вниз (`'ui-icon-triangle-1-s'`).

Глава 10
СТИЛИЗАЦИЯ ФОРМ

Формы являются исходным интерактивным веб-элементом и частью большинства веб-приложений. Формы позволяют посетителям предоставить информацию, покупателям совершать покупки, участникам сообществ обмениваться сообщениями и т. д. В главе 8 были рассмотрены способы для того, чтобы сделать веб-формы «умнее» и проще в использовании. Плагин jQuery UI позволяет сделать с формами еще больше, а также обеспечивает согласованный дизайн, благодаря чему элементы формы выглядят и функционируют схожим образом.

В этой главе вы узнаете, как использовать четыре виджета jQuery UI — Datepicker, Autocomplete, Selectmenu и Button, которые обеспечат великолепный внешний вид и функциональность ваших форм.

Стильный способ выбора даты

Многие формы предусматривают возможность выбора даты. Формы для добавления события в календарь, бронирования авиабилета и резервирования столика в ресторане требуют указания даты. Многие формы содержат инструкции, например, «Введите дату в формате 12/10/2014», однако если вы просто попросите посетителей ввести дату в поле, вы можете получить слишком много различных результатов. Прежде всего, вы будете рассчитывать на то, что ваши посетители не допускают опечаток. Кроме того, поскольку люди часто пишут даты по-разному (в США, например, даты указываются в порядке: месяц, день, год, а во многих других странах используется порядок: день, месяц, год), даты, введенные вручную, могут быть неточными или вводить в заблуждение.

К счастью, виджет jQuery UI Datepicker упрощает процесс выбора даты. Вместо того чтобы вводить дату вручную, посетители просто щелкают по элементу формы, а затем используют визуальный календарь для выбора нужной даты (рис. 10.1). Этот виджет прост в использовании, и его легко настроить.

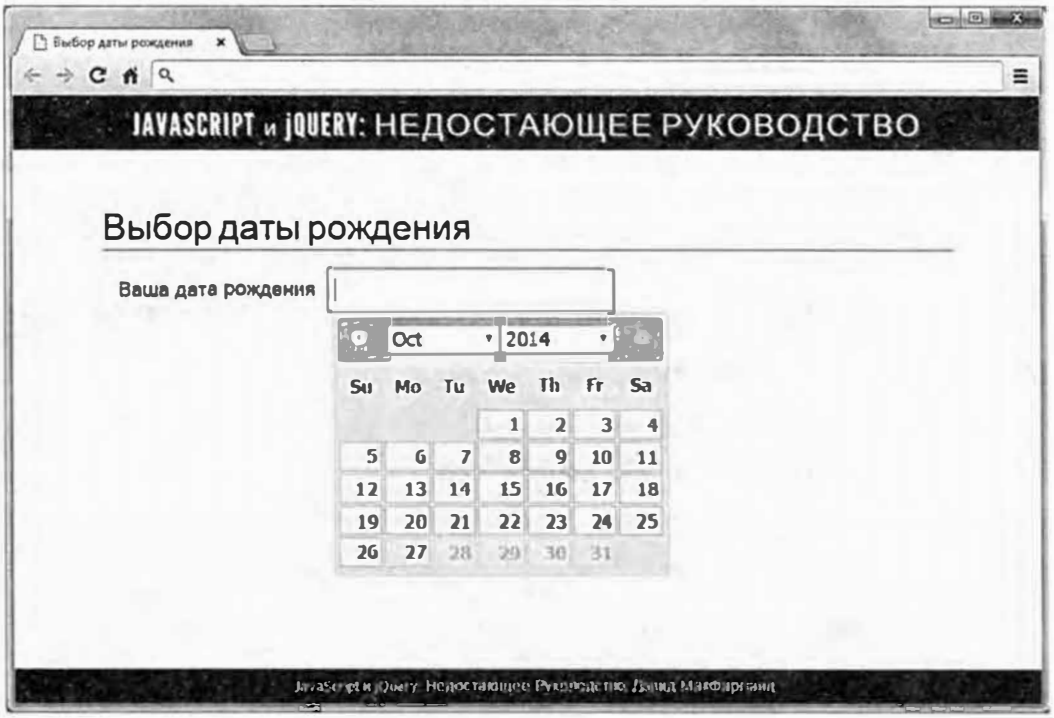

Рис. 10.1. Виджет jQuery UI Datepicker обеспечивает простой способ для точного указания даты в веб-форме

Как большинство виджетов jQuery UI, Datepicker чрезвычайно прост в использовании:

1. **Следуйте инструкции, описанной в разделе «Добавление jQuery UI на веб-страницу» главы 9, чтобы прикрепить файлы CSS и JavaScript плагина jQuery UI к своей веб-странице.**

   Вам также потребуется прикрепить файл jQuery, чтобы при использовании виджета jQuery UI к вашей странице были прикреплены файлы CSS jQuery UI, JavaScript jQuery и JavaScript jQuery UI (именно в таком порядке).

2. **Добавьте на страницу форму и текстовое поле `input` для выбора даты.**

   Обеспечьте способ идентификации этого поля ввода, чтобы вы могли выбрать его с помощью jQuery. Например, вы можете предоставить ему следующий идентификатор:

   ```
   <input type="text" name="birthdate" id="birthdate">
   ```

   Если форма предусматривает несколько элементов для выбора дат (например, прибытия и отъезда), то вы можете использовать имя класса для определения всех элементов формы, которые должны использовать виджет Datepicker следующим образом:

```
<input type="text" name="arrival" class="date">
<input type="text" name="departure" class="date">
```

3. **Добавьте на страницу функцию `$(document).ready()`:**

```
$(document).ready(function() {
});  // окончание ready
```

4. **Используйте jQuery для выбора входного элемента(ов) и вызовите функцию `datepicker()`:**

```
$(document).ready(function() {
    $('#birthdate').datepicker();
});  // окончание ready
```

Если вы использовали класс для идентификации более одного входного элемента, как в примере в шаге 2, то можете написать следующее:

```
$(document).ready(function() {
    $('.date').datepicker();
});  // окончание ready
```

Это все, что необходимо для создания панели выбора даты, показанной на рис. 10.1. И если бы кроме этого виджет Datepicker больше ничего не делал, то и этого было бы достаточно. Однако виджет Datepicker предусматривает множество различных параметров для настройки его внешнего вида и принципа работы.

ПРИМЕЧАНИЕ

Версия HTML5 предусматривает специальный элемент `date` формы, который предназначен для обеспечения некоторой части функциональности виджета Datepicker без необходимости в использовании JavaScript. Тем не менее он не одинаково поддерживается во всех браузерах и не предусматривает способа для настройки внешнего вида всплывающего календаря. Кроме того, виджет Datepicker jQuery UI предоставляет много дополнительных возможностей, которые вы не можете получить при использовании элемента `date` HTML5.

Настройка свойств виджета Datepicker

Вы можете установить свойства виджета Datepicker, например, формат даты, который будет использовать плагин jQuery UI при выводе

даты в элементе формы, передав объект функции `datepicker()`. Этот литерал объекта содержит параметры `datepicker` и значения для управления этими параметрами.

Например, один параметр — `numberOfMonths` — позволяет указать, сколько месяцев должно отобразиться в панели выбора даты. Обычно отображается всего 1, как показано на рис. 10.1, однако можно отобразить три месяца, как показано на рис. 10.2, установив значение 3:

```
$('.date').datepicker({
      numberOfMonths : 3
});
```

Рис. 10.2. Виджет Datepicker jQuery UI очень легко настроить. Вы можете изменить количество отображаемых месяцев, указать год перед названием месяца, а также использовать названия месяцев на различных языках

Далее приведены некоторые из самых полезных параметров:

- **`changeMonth`**. Как правило, посетители могут изменить месяц, отображаемый в панели выбора дат Datepicker, щелкнув по значкам в виде стрелок влево и вправо в верхней части календаря (см. рис. 10.1). Эти кнопки отображают либо предыдущий (слева), либо следующий месяц (справа). Тем не менее это очень утомительный способ выбора даты, которая наступит, например, через девять месяцев. Укажите значение `true` для параметра `changeMonth`, и появится раскрывающийся список, позволяющий посетителям быстро выбрать новый месяц (см. рис. 10.2).

  ```
  changeMonth : true
  ```

- **`changeYear`**. Как и в случае с `changeMonth`, при указании значения `true` для этого параметра плагин jQuery UI отображает раскрывающийся список для выбора нового года для календаря. Вы будете

часто использовать этот параметр с параметром `yearRange` (описывается далее), чтобы указать количество лет, отображаемых в раскрывающемся списке.

```
changeYear : true
```

- **`dateFormat`**. Предоставляет строку, определяющую формат, который должен использоваться плагином jQuery UI при выводе выбранной даты в элементе формы. Для определения различных вариантов выходных данных вы можете использовать предопределенные коды. Например, код `dd` используется для указания дня месяца, `mm` — для указания месяца года, а `yy` — для указания года. Вы также можете вставить в качестве части строки такие символы как пробел, / или -. Допустим, посетитель выбирает дату «27 января 2015» во всплывающем календаре, а вы хотите, чтобы в форме эта дата отобразилась в формате 01-27-2015. В этом случае вам нужно настроить параметр `dateFormat` следующим образом:

  ```
  dateFormat : 'mm-dd-yy'
  ```

  Плагин jQuery UI предусматривает множество различных кодов для форматирования даты. Некоторые полезные примеры вы можете найти в табл. 10.1. Полный список форматов, принимаемых виджетом Datepicker jQuery UI, вы можете найти по адресу: **api.jqueryui.com/datepicker/#utility-formatDate**.

ПРИМЕЧАНИЕ

Для виджета Datepicker jQuery UI предусмотрены и другие параметры, события и методы. Вы можете найти их по адресу **api.jqueryui.com/datepicker/**.

- **`monthNames`**. Предоставьте массив, содержащий 12 строк, чтобы заменить названия месяцев на английском языке. Например, чтобы отобразить названия месяцев на русском языке, добавьте следующий код к литералу объекта, переданного функции `datepicker ()`:

  ```
  monthNames: [ "Январь", "Февраль", "Март", ↵
  "Апрель", "Май", "Июнь", "Июль", "Август", ↵
  "Сентябрь", "Октябрь", "Ноябрь", "Декабрь" ]
  ```

- **`numberOfMonths`**. Укажите численное значение для данного параметра, чтобы определить количество месяцев, отображаемых в панели выбора даты.

Табл. 10.1. Полезные строки для передачи параметру `dateFormat`.

Пример	Значение	Пример выходных данных
`'yy-mm-dd'`	Полный год, тире, две цифры, означающие месяц, тире, две цифры, означающие день. Этот формат используется для указания даты в базах данных MySQL.	2015-02-05
`'m/d/y'`	1 или 2 цифры, означающие месяц, прямой слеш, 1 или 2 цифры, означающие день, прямой слеш, 2 цифры, означающие год	2/5/15
`'D, M d, yy'`	Сокращенное название дня недели, запятая, пробел, сокращенное название месяца, пробел, 1 или 2 цифры, означающие день, запятая, пробел, полный год	Thu, Feb 5, 2015
`'DD, MM dd, yy'`	Полное название дня недели, запятая, пробел, полное название месяца, пробел, 1 или 2 цифры, означающие день, запятая, пробел, полный год	Thursday, February 5, 2015
`'@'`	Временная метка Unix. Количество миллисекунд, прошедших с полуночи 1 января 1979 года (см. врезку «Объект `Date`, работающий за кадром» главы 16)	1423123200000

Обычно отображается один месяц, но вы можете обеспечить отображение нескольких месяцев. Если вы укажете значение большее, чем 3 (см. рис. 10.2), то виджет Datepicker станет немного громоздким. Панели выбора дат всегда отображаются бок о бок, так что если их больше трех или четырех, то вам придется пользоваться горизонтальной прокруткой, чтобы увидеть другие месяцы, доступные для выбора. Таким образом, для данного параметра рекомендуется использовать значения 1, 2, или 3.

- **`maxDate`**. Устанавливает последнюю доступную дату, которую посетитель может выбрать во всплывающем окне календаря Datepicker. Например, вы можете использовать данный параметр в системе бронирования номера в отеле. Многие отели не позволяют забронировать номер раньше, чем за один год до прибытия, поэтому вы можете ввести ограничение так, чтобы посетитель не мог указать дату, отстоящую от текущей более, чем на год. Одним из способов является присвоение числового значения, определяющего количество дней в будущем. Например, чтобы предотвратить возможность выбора

даты, отстоящей от текущей более чем на 30 дней, вы можете настроить параметр `maxDate` следующим образом:

```
maxDate : 30
```

В качестве альтернативы вы можете передать этому параметру строку, содержащую специальные символы, которая указывает продолжительность: `y` — соответствует количеству лет, `m` —месяцев, `w` — недель и `d` — количеству дней. Например, чтобы ограничить выбор до одного года, используйте следующий код:

```
maxDate : '+1y'
```

Разделите символы пробелом. Например, если вы хотите ограничить выбор даты тремя месяцами, двумя неделями и пятью днями, вы можете сделать так:

```
maxDate : '+3m +2w +5d'
```

Пример использования параметра `maxDate` вы найдете в руководстве далее в главе.

- **`minDate`**. Это противоположность параметра `maxDate`. Он указывает самую раннюю дату, доступную для выбора. Этот параметр очень полезен для форм, предназначенных для планирования. В конце концов, нет никакого смысла в том, чтобы позволить посетителю указать дату до сегодняшнего дня (если только со времени написания этой книги не была изобретена машина времени). Для указания самой ранней даты вы используете те же символы, что и в случае с параметром `maxDate`. Например, чтобы предотвратить выбор даты до текущего дня, укажите для данного параметра значение `0`:

  ```
  minDate : 0
  ```

  Положительные значения этого параметра означают, что посетитель должен выбрать день в будущем. Например, если номер в отеле заказан на следующие три недели, то вы можете предотвратить выбор даты в этом диапазоне:

  ```
  minDate : '+3w'
  ```

  Используйте отрицательные значения для определения самой ранней доступной для выбора даты в прошлом. Например, вы создали форму для осуществления поиска по базе данных вашей компании, в которой содержится архив электронной почты. Чтобы предотвратить переполнение этой базы данных, компания хранит переписку только за два последних года, поэтому нет смысла предоставлять

возможность выбора дат, отстоящих от текущей на три, четыре или пять лет — эти письма давным-давно стерты. Вот как можно ограничить поиск последними двумя годами:

```
minDate : '-2y'
```

Вы можете комбинировать даты, месяцы и годы. Например, далее приведен код для определения даты, отстоящей от текущей на один год, два месяца и три дня:

```
minDate : '-1y -2m -3d'
```

ПРИМЕЧАНИЕ

Параметры minDate и maxDate также принимают объект JavaScript `Date` в качестве приемлемого значения. Например, ваша компания начала свою деятельность 13 марта 2010 года. Вы можете установить самую раннюю дату для календаря Datepicker следующим образом:

```
minDate : new Date(2010, 2, 13)
```

В разделе «Дата и время» главы 16 вы узнаете, как с помощью JavaScript создавать объекты `Date`.

- **`yearRange`**. Этот параметр используется с параметром `changeYear` (см. выше) и определяет, сколько лет отображается в раскрывающемся списке. Например, вы хотите, чтобы пользователь предоставил свою дату рождения. Вы указываете значение `true` для параметра `changeYear`, чтобы посетитель легко мог вернуться на 20, 30 или 50 лет назад. Как правило, параметр `changeYear` обеспечивает отображение в раскрывающемся списке 10 лет в прошлом и 10 лет в будущем. Но было бы лучше перечислить больше прошедших лет вместо будущих. Чтобы сделать это, вы передаете параметру `yearRange` положительное или отрицательное числовое значение, за которым следует двоеточие и еще одно положительное или отрицательное число. Первое число представляет собой первый год, указанный в раскрывающемся списке, а второе число — последний год.

Вернемся к примеру с датой рождения. Вам нужно перечислить, скажем, 120 прошедших лет и ни одного будущего. Вот как это можно сделать:

```
yearRange : '-120:0'
```

Этот фрагмент кода дает плагину jQuery UI задание отобразить раскрывающийся список для выбора года, в котором первый год отстоит от текущего на 100 лет, а последний представляет собой текущий год. Если вы запутались, обратитесь к руководству далее в главе.

Добавление панели для выбора даты рождения на практике

Пришло время опробовать виджет Datepicker. В этом руководстве вы превратите текстовое поле в интеллектуальное устройство для выбора даты рождения.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *10_01.html* в папке *глава 10*.**

   Этот файл уже содержит ссылку на все файлы плагина jQuery UI и библиотеки jQuery, а также функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5). Первым шагом является выбор элемента формы.

2. **Внутри функции `$(document).ready()` введите код:**

   ```
   $('#dob')
   ```

   Если вы посмотрите на HTML-код этой страницы, то увидите элемент для выбора даты рождения: `<input type="text" id="dob" name="birthdate">`. Он имеет идентификатор `dob`, поэтому приведенный выше фрагмент кода выбирает данный элемент формы. Теперь вам нужно применить виджет Datepicker.

3. **Введите точку и значение `datepicker();`, чтобы код выглядел следующим образом:**

   ```
   $('#dob').datepicker();
   ```

   Вот и все, что нужно сделать!

4. **Сохраните файл и просмотрите его в браузере. Щелкните по элементу формы.**

Календарь всплывает как по волшебству. При оформлении данного календаря используется тема Lightness плагина jQuery UI, однако в следующей главе вы узнаете, как изменить внешний вид виджета.

При попытке выбрать дату рождения вы столкнетесь с трудностями. Вам придется щелкнуть по кнопке в виде стрелки влево в верхней части календаря 12 раз, только чтобы продвинуться на один год назад! Таким образом, вы можете облегчить выбор более ранней даты, добавив раскрывающиеся списки для выбора месяца и года.

5. **Вернитесь в текстовый редактор и добавьте литерал объекта в функцию `datepicker()`:**

```
$('#dob').datepicker({

});
```

Для изменения параметров вы передаете функции `datepicker()` объект `{}`, содержащий параметры, имена и значения. Сначала добавьте раскрывающийся список для выбора месяца.

6. **Добавьте в объект фрагмент `changeMonth : true`:**

```
$('#dob').datepicker({
changeMonth : true
});
```

Если вы сейчас сохраните страницу, откроете ее в браузере и щелкнете по элементу формы, вы увидите раскрывающийся список в заголовке календаря, который позволяет выбрать любой из 12 месяцев в году. Этот раскрывающийся список обеспечивает гораздо более быстрый способ выбора даты, отстоящей от текущей на 9 месяцев.

Далее вы создадите раскрывающийся список для выбора года.

7. **Добавьте запятую после только что введенной строки, щелкните по клавише Enter и введите фрагмент кода `changeYear : true`:**

```
$('#dob').datepicker({
changeMonth : true,
changeYear : true
});
```

Этот код добавляет в календарь еще один раскрывающийся список. К сожалению, он позволяет выбрать только среди 10 прошлых и 10

будущих лет, а это не то, что вам нужно. Если ваш посетитель не моложе 10 лет и у него нет машины времени, то от вариантов, содержащихся в этом раскрывающемся списке, не будет большой пользы. К счастью, вы можете изменить диапазон лет, отображаемых в раскрывающемся списке.

8. **Добавьте запятую после только что введенной строки, нажмите клавишу Enter и введите код `yearRange : '-120:+0'`:**

```
$('#dob').datepicker({
changeMonth : true,
changeYear : true,
yearRange : '-120:+0'
});
```

Это приводит к изменению диапазона лет, отображаемых в раскрывающемся списке. Теперь вы можете выбрать из 120 прошлых лет, а будущие годы не отображаются. Теперь все намного лучше. Тем не менее если вы протестируете последние изменения, то заметите, что у вас есть возможность выбрать завтрашний день или дату на следующей неделе или в следующем месяце. Вам нужно ограничить выбор даты текущим днем.

9. **Введите еще одну запятую после только что добавленной строки, нажмите клавишу Enter и введите `maxDate : 0`:**

```
$('#dob').datepicker({
changeMonth : true,
changeYear : true,
yearRange : '-120:+0',
maxDate : 0
});
```

Теперь вы не можете выбрать дату позднее текущей. Далее вы поменяете формат даты так, чтобы дата рождения записывалась следующим образом: 1-27-2015.

10. **Введите еще одну запятую после только что добавленной строки, нажмите клавишу Enter и введите фрагмент `dateFormat : 'm-dd-yy'`:**

```
$('#dob').datepicker({
changeMonth : true,
changeYear : true,
yearRange : '-120:+0',
maxDate : 0,
dateFormat : 'm-dd-yy'
});
```

Теперь у вас есть настроенная панель, которая идеально подходит для выбора даты рождения.

11. Сохраните файл и просмотрите его в браузере.

При щелчке по элементу формы появится настроенная панель выбора даты (рис. 10.3). Итоговый вариант этого кода вы можете найти в файле *готовый_10_01.html* в папке *глава10*.

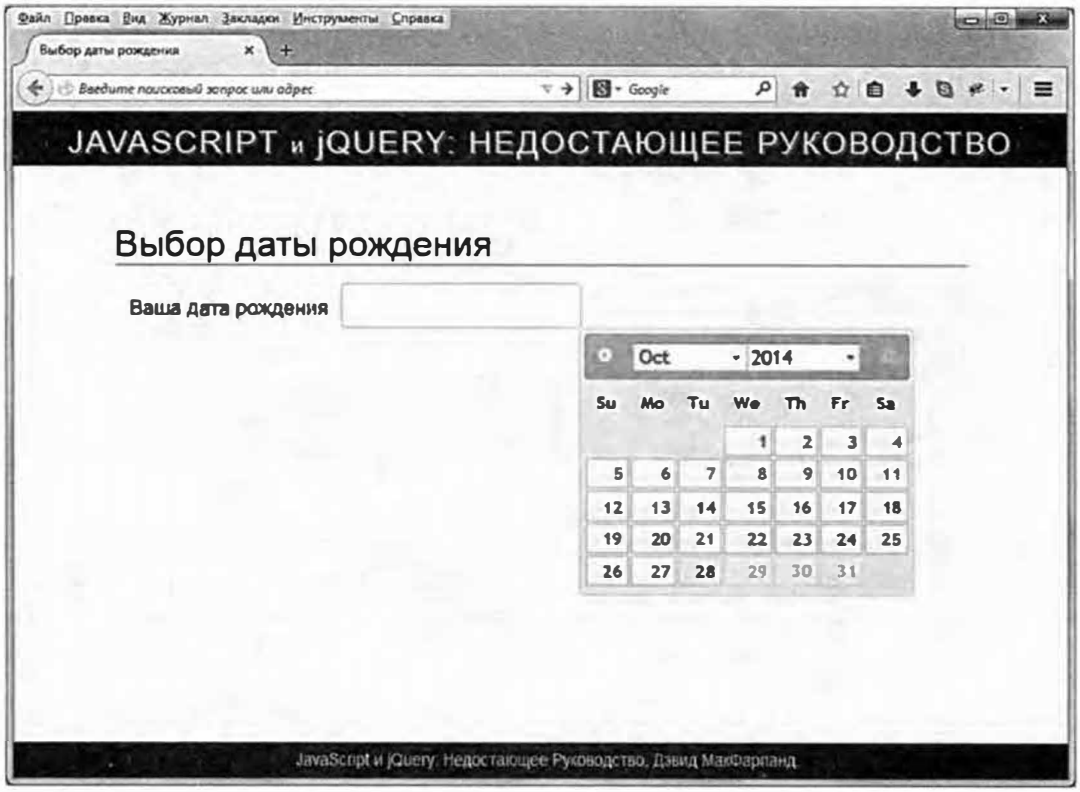

Рис. 10.3. Виджет Datepicker плагина jQuery UI является необходимым элементом для любой формы, требующей ввода даты. Существует практически бесконечное число способов его настройки, которые позволяют выбирать даты в будущем или ограничить выбор только прошлыми датами. В данном случае любая дата после 28 октября 2014 выделена серым цветом и поэтому не может быть выбрана, однако раскрывающиеся списки позволяют легко выбрать в календаре год и месяц в далеком прошлом

Стилизация раскрывающихся списков

Темы jQuery UI позволяют обеспечить единообразный внешний вид элементов пользовательского интерфейса. Например, панель выбора даты похожа на панель с вкладками, которая похожа на диалоговые окна. Раскрывающиеся списки, то есть элементы формы, позволяющие выбрать вариант из выпадающего списка, как известно, трудно стилизовать с помощью кода CSS. Каждый браузер предусматривает собственный способ отображения раскрывающихся списков, который зачастую определяется операционной системой (Windows, OS X, Linux), кроме того, браузеры не позволяют применять к раскрывающимся спискам каждое свойство CSS.

К счастью, плагин jQuery UI предусматривает удобный виджет Selectmenu, превращающий обычный HTML-код в привлекательный раскрывающийся список, внешний вид которого соответствует внешнему виду других виджетов jQuery UI (рис. 10.4). Виджет Selectmenu буквально воссоздает меню в виде неупорядоченного списка и серии элементов `span`, которые можно легче стилизовать с помощью CSS. Используя программный код JavaScript, этот виджет скрывает реальный раскрывающийся список, в то же время позволяя посетителям выбрать вариант из раскрывающегося списка, работающего на основе JavaScript. Выбор посетителя отражается в реальном элементе формы, поэтому при отправке формы выбор посетителя передается на веб-сервер в правильном виде.

Рис. 10.4. Обычный вид раскрывающегося списка в форме контролируется браузером и операционной системой (слева). Тем не менее с помощью плагина jQuery UI вы можете превратить серые раскрывающиеся списки в стильные элементы интерфейса, чей внешний вид соответствует виду других виджетов jQuery UI

За кулисами плагин jQuery UI выполняет некоторые сложные программные задачи, однако для вас превращение раскрывающегося списка в меню jQuery UI не составляет труда:

1. **Следуйте инструкциям, приведенным в разделе «Добавление jQuery UI на веб-страницу» главы 9, чтобы прикрепить файлы CSS и JavaScript плагина jQuery UI к вашей веб-странице.**

При каждом использовании виджета jQuery UI к вашей веб-странице должны быть прикреплены файлы jQuery UI CSS, jQuery JavaScript и jQuery UI JavaScript (именно в таком порядке).

2. **Добавьте на страницу форму и раскрывающийся список, то есть элемент `select`, содержащий элементы `option`:**

```
<select name="meal" id="meal">
<option>Нет</option>
<option>Для веганов</option>
<option>Без глютена</option>
<option>Для вегетарианцев</option>
<option>Для мясоедов</option>
</select>
```

Вы должны предусмотреть способ выбора раскрывающегося списка с помощью jQuery. Для этого примените к раскрывающемуся списку идентификатор, а при наличии нескольких раскрывающихся списков в форме примените к ним один и тот же класс, например, `class="select"`.

3. **Добавьте на страницу jQuery-функцию `$(document).ready()`:**

```
$(document).ready(function() {
}); // окончание ready
```

Вам нужно сделать это, только если вы размещаете свой программный код JavaScript в разделе заголовка страницы, как описано в разделе «Альтернатива для функции `$(document).ready()`» главы 5.

4. **Используйте jQuery для выбора раскрывающегося списка и вызова функции `selectmenu()`:**

```
$(document).ready(function() {
$('#meal').selectmenu();
}); // окончание ready
```

Если вы использовали класс для идентификации более одного раскрывающегося списка, вы можете написать следующий код:

```
$(document).ready(function() {
```

```
$('.select).selectmenu();
});  // окончание ready
```

Если параметры в раскрывающемся списке не очень короткие, то виджет jQuery UI Selectmenu часто не отображает первый параметр полностью. Другими словами, виджет не достаточно широкий, чтобы отобразить первый параметр. Это выглядит странно, поэтому вам всегда необходимо указывать ширину раскрывающегося списка.

5. **Передайте функции `selectmenu()` литерал объекта со свойством `width`:**

```
$(document).ready(function() {
$('#meal').selectmenu({
width: 200
});
});  // окончание ready
```

Как и в случае с другими виджетами jQuery UI (например, Datepicker, о котором вы читали в разделе «Настройка свойств виджета Datepicker» ранее в главе), вы можете указать различные параметры для виджета Selectmenu, передав литерал объекта, содержащий пары ключ/значение. Параметр `width` требуется почти всегда и принимает числовое значение, соответствующее ширине раскрывающегося списка в пикселах (обратитесь к следующему разделу, чтобы больше узнать о параметрах свойства `width`).

Это все, что вам необходимо сделать, чтобы создать раскрывающийся список, изображенный на рис. 10.4.

Настройка свойств раскрывающегося списка

Виджет Selectmenu предусматривает не очень много параметров. Это всего лишь инструмент, позволяющий обеспечить внешний вид раскрывающегося списка, соответствующий виду других виджетов jQuery UI. Тем не менее существует несколько параметров, которые могут оказаться полезными:

- **`width`**. Этот параметр практически всегда является обязательным. Обычно плагин jQuery UI создает раскрывающийся список, который недостаточно широк, чтобы отобразить весь текст пункта спи-

ска. Вам нужно сделать его достаточно широким, чтобы вы могли увидеть весь пункт раскрывающегося списка. Кроме того, если раскрывающийся список не достаточно широк, чтобы отобразить параметр, который состоит более чем из одного слова, то плагин jQuery UI отобразит этот параметр на двух строках. Для указания ширины в пикселах вы можете использовать числовое значение:

```
width : 300
```

Вы также можете задать ширину в процентах или единицах em, используя строку, содержащую число и символ % или em. Например, чтобы раскрывающийся список соответствовал своему родительскому элементу (например, элементу `div`), вы можете указать 100% в качестве значения его ширины:

```
width : '100%'
```

Если вы предпочитаете единицы em, вы можете указать значение ширины следующим образом:

```
width : '6em'
```

- **`icons`.** Вы также можете отобразить один из значков jQuery UI справа от раскрывающегося списка (например, треугольник на рис. 10.4). Плагин jQuery UI предусматривает большой набор значков в каждой теме (обратитесь на сайт **api.jqueryui.com/theming/icons/**, чтобы ознакомиться с полным списком). Вы можете добавить значки, передав параметр `icons` и значение, которое включает в себя литерал объекта:

```
icons : {
button: "ui-icon-circle-triangle-s"
}
```

ПРИМЕЧАНИЕ

Возможно, вас интересует, почему параметр `icons` требует наличия еще одного литерала объекта с парой имя/значение. В конце концов, было бы намного проще задать имя значка, используя свойство `icons`. Это имело бы смысл, если бы Selectmenu являлся единственным виджетом, для которого можно было бы назначить значок. Другие виджеты также могут отображать значки, а некоторые виджеты, например, Accordion (см. раздел «Экономия пространства с помощью аккордеонов» главы 9), позволяют назначать более одного значка. Для этого виджета вы должны передать литерал объекта,

содержащий несколько пар имя/значение, чтобы отобразить более одного значка. Для обеспечения соответствия между различными виджетами jQuery UI виджет Selectmenu также требует передачи значения объекта параметра `icons`.

- **`position`.** С помощью данного параметра вы можете контролировать положение раскрывающегося списка. Как правило, раскрывающийся список появляется непосредственно под его элементом управления — это обычный принцип его работы. Тем не менее при желании вы можете заставить всплывающий список пунктов раскрывающегося списка отобразиться слева или справа от элемента управления. Для этого вы можете установить параметр `position` с помощью jQuery UI-объекта position. Поскольку позиционирование раскрывающегося списка в любом месте, кроме как под элементом управления формой, является необычным поведением, будьте осторожны с этим свойством, поскольку вы можете ввести посетителей в заблуждение. (Это специализированная утилита обсуждалась в предыдущей главе в тексте в рамке «Точное позиционирование с помощью плагина jQuery UI».)

Выполнение действия при выборе пункта раскрывающегося списка

Когда посетитель выбирает пункт раскрывающегося списка, что-то должно произойти. Например, у вас есть форма онлайн-заказа одежды с раскрывающимся списком, в котором перечислены все доступные цвета. Когда посетитель выбирает из раскрывающегося списка цвет, вы можете обновить изображение предмета одежды, чтобы показать его в новом цвете. Другими словами, выбор пункта раскрывающегося списка приводит к изменению изображения на странице. Плагин jQuery UI позволяет вызвать функцию при каждом выборе пункта раскрывающегося списка.

Для этого вы используете параметр `change`. Он работает подобно другим параметрам, обсуждаемым в этой главе, в том плане, что вы помещаете его внутри литерала объекта, передаваемого функции `selectmenu()`. В качестве значения параметра вы передаете параметру `change` функцию. Например, у вас есть раскрывающийся список с идентификатором `colors`. Теперь вам нужно превратить этот раскрывающийся список в виджет Selectmenu jQuery UI и добавить параметр `change`:

```
$('#colors').selectmenu({
width : 300,
change : function (event, ui) {
// поместите сюда программный код
}
});
```

Всякий раз, когда посетитель делает выбор (то есть изменяет раскрывающийся список), выполняется эта функция. Функция предусматривает два параметра: `event` и `ui`. Параметр `event` содержит объект события jQuery UI (см. раздел «Объект события» главы 5). Вам, вероятно, не понадобится использовать этот параметр в функции — он содержит только информацию о событии, например, X и Y координаты указателя мыши и другие данные, которые не относятся к работе с пунктами раскрывающегося списка.

Тем не менее параметр `ui` содержит полезную информацию о раскрывающемся списке. В частности, он может сообщить вам значение индекса вновь выбранного пункта. То есть его место в раскрывающемся списке (индекс первого пункта имеет значение `0`). Параметр `ui` также содержит метку и значение выбранного варианта. Параметр `ui` сам по себе является объектом, состоящим из различных свойств, к которым вы можете получить доступ, используя точечную нотацию (см. раздел «Небольшой урок, посвященный объектам» главы 2).

- **`ui.item.index`** содержит значение индекса выбранного пункта раскрывающегося списка. Варианты в раскрывающемся списке нумеруются, как элементы массива: первый элемент раскрывающегося списка имеет индекс `0`, второй — `1` и т. д.
- **`ui.item.label`** содержит метку выбранного пункта раскрывающегося списка. Метка — это слово или слова, которые посетитель видит в раскрывающемся списке. Она находится в HTML-коде внутри элемента `option`. Поэтому если у вас на странице находится такой раскрывающийся список:

```
<select id="colors">
<option>Красный</option>
<option>Зеленый</option>
<option>Синий</option>
</select>
```

то метками для пунктов этого раскрывающегося списка являются: Красный, Зеленый и Синий.

- **ui.item.value** содержит значение выбранного пункта раскрывающегося списка. Это значение устанавливается с помощью атрибута value элемента option. Часто метка и значение бывают одинаковыми. В этом случае вам не нужно указывать атрибут value в вашем HTML-коде. Однако иногда вам может захотеться передать на обрабатывающий данные формы сервер другое значение. Например, компания может предусмотреть специальные коды, соответствующие цвету. Посетителю может понадобиться «красная» рубашка, но в компании по производству одежды, которая использует несколько оттенков красного для различных типов одежды, красный цвет рубашки может обозначаться специальным кодом, например, R785.

В этом случае форма будет включать в себя значения в дополнение к меткам:

```
<select id="colors">
<option value="R785">Красный</option>
<option value="G101">Зеленый</option>
<option value="B498">Синий</option>
</select>
```

Для данного раскрывающегося списка значениями являются "R785", "G101" и "B498", а метками по-прежнему Красный, Зеленый и Синий.

Теперь подумайте о том, как вы можете изменить изображение на странице, когда посетитель выбирает пункт в раскрывающемся списке. Скажем, вы используете HTML-код для описанного выше раскрывающегося списка, содержащего три варианта цвета: красный, зеленый и синий. Когда посетитель делает выбор, браузеру необходимо загрузить изображение рубашки, соответствующее выбранному цвету. Предположим, что при загрузке страницы на ней присутствует следующий HTML-код:

```
<img src="red_shirt.jpg" id="shirt" alt="Покупайте ↵
наши рубашки!">
```

Как вы помните из раздела «Смена изображений» главы 7, изменяя атрибут изображения src, вы можете приказать браузеру загрузить но-

вое изображение в этом месте. Таким образом, для загрузки на страницу изображения синей рубашки вы можете выбрать это изображение и изменить его атрибут `src` следующим образом:

```
$('#shirt').attr('src', 'blue_shirt.jpg');
```

Собрав все воедино, вы можете изменять изображение каждый раз, когда в раскрывающемся списке выбирается новый пункт:

```
$('#colors').selectmenu({
width : 300,
change : function (event, ui) {
var newImage;
if (ui.item.label === 'Красный') {
newImage = 'red_shirt.jpg';
} else if (ui.item.label === 'Зеленый') {
newImage = 'green_shirt.jpg';
} else {
newImage = 'blue_shirt.jpg';
}
$('#shirt').attr('src', newImage);
}
});
```

С помощью параметра `change` вы можете делать множество вещей, например, обновлять HTML-код, добавлять второй раскрывающийся список с другим набором пунктов, связанных с вариантом, выбранным в исходным раскрывающемся списке и т. д.

ПРИМЕЧАНИЕ

Для виджета jQuery UI Selectmenu существуют и другие варианты, события и методы. Чтобы узнать о них, посетите сайт: **api.jqueryui.com/selectmenu/**.

Стилизация кнопок

Плагин jQuery UI также предусматривает виджет, позволяющий обеспечить согласованность поведения элементов форм, имеющих вид

кнопок. Виджет Button может использоваться для стилизации кнопки отправки, кнопки сброса или элемента input, имеющего тип button: <input type="button">. Кроме того, вы можете использовать его для стилизации HTML-элемента button, чтобы его внешний вид соответствовал CSS-теме jQuery UI (рис. 10.5):

1. **Следуйте инструкции, описанной в разделе «Добавление jQuery UI на веб-страницу» главы 9, чтобы прикрепить файлы CSS и JavaScript плагина jQuery UI к своей веб-странице.**

   Вам также потребуется прикрепить файл jQuery, чтобы при использовании виджета jQuery UI к вашей странице были прикреплены файлы CSS jQuery UI, JavaScript jQuery и JavaScript jQuery UI (именно в таком порядке).

2. **Добавьте на страницу кнопку. Это может быть кнопка сброса, отправки, ввода или элемент `button`. Например:**

```
<inputtype="submit"id="submit"value="Отправитьформу!">
<inputtype="reset"id="reset"value="Очиститьформу.">
<input type="button" id="inputButton" ↵
value="Кнопка ввода.">
```

   Вы также должны предусмотреть способ выбора раскрывающегося списка с помощью jQuery. Вы можете либо применить к элементу идентификатор, либо при наличии нескольких кнопок в форме применить к ним один и тот же класс, например, class="button".

3. **Добавьте на страницу jQuery-функцию `$(document).ready()`:**

```
$(document).ready(function() {
});// окончание ready
```

   Вы должны это сделать только в том случае, если вы размещаете программный код JavaScript в разделе заголовка вашей страницы, как описано в разделе «Больше концепций для событий jQuery» главы 5.

4. **Используйте jQuery, чтобы выбрать кнопку и вызвать функцию `button()`:**

```
$(document).ready(function() {
$('#submit').button();
});// окончание ready
```

Если вы использовали класс для идентификации более одной кнопки, вы можете написать так:

```
$(document).ready(function() {
$('.button').button();
}); // окончание ready
```

При использовании этого виджета к элементу применяются стили, соответствующие используемой теме jQuery UI (см. рис. 10.5).

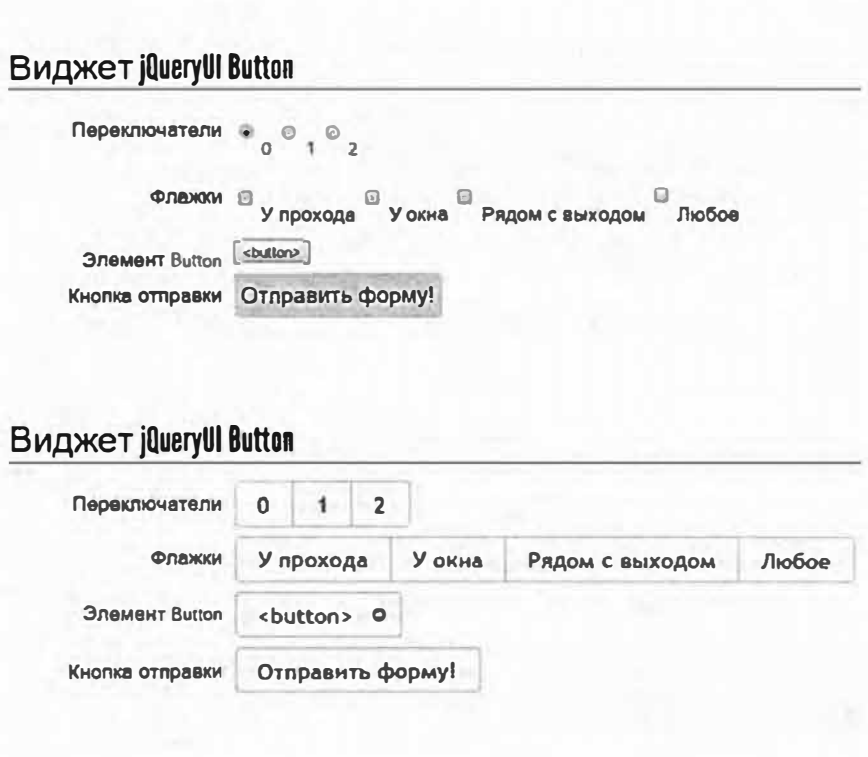

Рис. 10.5. Виджет jQuery UI Button позволяет быстро и легко превратить невзрачные элементы формы, например, переключатели, флажки, кнопки отправки и кнопки HTML (вверху) в элегантные элементы интерфейса, чей внешний вид соответствует друг другу (внизу)

Настройка кнопок

Виджет jQuery UI Button не предусматривает большого количества параметров для настройки. Текст, который вы видите на кнопке, определяется значением атрибута `value` в случае с такими элементами `input`, как кнопки отправки и сброса. В случае с элементом `button`, включающим в себя открывающий и закрывающий теги — `<button>Я кнопка</button>`, — текст, который вы видите на кнопке, определяется

содержимым, заключенным между тегами. Тем не менее вы можете настроить кнопки несколькими способами, передав значения параметров методу `button()`.

- **`icons`**. Вы также можете отобразить один из значков jQuery UI слева от кнопки и еще один значок справа (обратите внимание на элемент `button` на рис. 10.5). Плагин jQuery UI включает в себя большой набор значков в каждой теме (полный список вы можете найти на сайте: **api.jqueryui.com/theming/icons/**). Вы можете добавить значок, передав параметр `icons` и значение, которое включает в себя объект:

```
icons : {
primary: "ui-icon-gear",
secondary: "ui-icon-triangle-1-s"
}
```

  «Первичный» значок отображается слева, а «вторичный» — справа. Вам не нужно указывать два значка, поскольку это, как правило, выглядит довольно странно. Обычно вы будете указывать только один из двух. Например, для отображения стрелки вправо на всех элементах `button` на странице вы можете написать следующий код:

```
$('button').button({
icons : { secondary: 'ui-icon-circle-arrow-e' } });
```

ПРИМЕЧАНИЕ

Вы не можете применить параметр `icons` к кнопке, созданной с помощью элемента `input`, то есть к кнопкам отправки или сброса, — только к элементу `button`.

- **`text`.** Если вы используете элементы `button` и применяете к этим кнопкам параметр `icons`, то вы можете полностью скрыть текст на кнопке и отобразить только значок. Указав значение `false` для параметра `text`, вы скрываете метку кнопки и отображаете только значок. Например, у вас есть кнопка с меткой `Далее`:

```
<button id="next">Далее</button>
```

  Вы можете превратить эту кнопку в кнопку jQuery UI, добавить к ней значок в виде стрелки вправо и удалить с кнопки слово «Далее» следующим образом:

```
$('#next').button({
icons : { secondary : 'ui-icon-arrowthick-1-e' }, ↵
text : false
});
```

ПРИМЕЧАНИЕ

Виджет Button jQuery UI предусматривает несколько других параметров, методов и событий, о которых вы можете прочитать на сайте **api.jqueryui.com/button/**.

Стилизация переключателей и флажков

Переключатели и флажки — это элементы формы, которые в своем исходном виде смотрятся не очень хорошо (см. рис. 10.5 вверху). Браузеры отображают их в виде, предусмотренном операционной системой для кнопок и флажков, которые нельзя настроить с помощью кода CSS так же, как другие HTML-элементы.

К счастью, плагин jQuery UI предусматривает виджет, позволяющий привести внешний вид переключателей и флажков в соответствие с другими элементами jQuery UI (см. рис. 10.5 внизу). Хорошая новость состоит в том, что для улучшения их внешнего вида от вас практически ничего не требуется. Всю работу делает метод jQuery UI `.buttonset()`. На самом деле он просто применяет метод `.button()` к каждому переключателю или флажку в группе.

Все, что вам нужно сделать, — это немного настроить HTML-код: поместите группу кнопок в некоторый контейнер, например, в элемент `div`, чтобы иметь возможность выбрать его с помощью jQuery. Например, вы создаете форму для бронирования авиабилета, в которой покупатели могут указать количество багажа — 0, 1 или 2. Ваш HTML-код может быть организован следующим образом:

```
<div id="radio">
<p class="label">Количество багажа</p>
<input type="radio" id="none" name="bags" ↵
checked="checked">
<label for="none">0</label>
```

```
<input type="radio" id="one" name="bags">
<label for="one">1</label>
<input type="radio" id="two" name="bags">
<label for="two">2</label>
</div>
```

В данном коде присутствует три переключателя — `<input type="radio">`, которые заключены в элемент `div` с идентификатором `radio`. Чтобы превратить их в группу переключателей jQuery UI, просто выполните шаги 1-3 в разделе «Добавление jQuery UI на веб-страницу» главы 9, чтобы загрузить и прикрепить файлы CSS и JavaScript jQuery и jQuery UI, и добавьте на страницу функцию `$(document).ready()`. Затем добавьте следующий JavaScript-код:

```
$('#radio').buttonset();
```

Вот и все. Код `$('#radio')` выбирает элемент `div`, содержащий переключатели; фрагмент `.buttonset()` находит каждый переключатель или флажок внутри этого элемента `div` и превращает его в кнопку. Уникальность функции `buttonset()` заключается в том, что она группирует все эти элементы так, чтобы они казались одним блоком (рис. 10.6).

▶ СОВЕТ

По умолчанию CSS-код плагина jQuery UI форматирует группу кнопок или флажков, как единое целое (см. рис. 10.6). То есть они расположены бок о бок и соприкасаются друг с другом. Если вы предпочитаете, чтобы кнопки не соприкасались друг с другом и располагались по отдельности, просто примените функцию `button()` к каждому элементу в отдельности. Например, в случае с приведенным выше HTML-кодом и группой переключателей вы можете добавить следующий JavaScript-код, чтобы создать отдельные кнопки:

```
$('#radio input').button();
```

Этот код применяет функцию `button()` отдельно к каждой кнопке внутри элемента `div`.

Та же самая техника применяется к флажкам. Просто сгруппируйте прилегающие флажки в своего рода выбираемый контейнер, например, `<div id="check">`, выберите этот контейнер и примените функцию `buttonset()`. Будь то группа переключателей или группа флажков,

плагин jQuery UI форматирует их так, как показано на рис. 10.6. Тем не менее эти два типа элементов управления ведут себя по-разному: в случае с переключателями одновременно может быть выбрана только одна кнопка в группе. В случае с флажками вы можете выбрать любое количество флажков в группе (или вообще ни одного). Плагин jQuery UI выделяет выбранные кнопки или флажки.

Рис. 10.6. Вы можете выбрать только одну кнопку в группе переключателей (вверху). В данном примере выбрана средняя кнопка «1» (и выделена плагином jQuery UI). Вы, тем не менее, можете выбрать любое количество флажков в группе (внизу). В данном случае выбраны первый («У прохода») и третий («Рядом с выходом») флажки

Функция `buttonset()` не принимает никаких аргументов. Тем не менее за кулисами она просто применяет метод `button()` к каждой кнопке или флажку в группе. Это значит, что вы можете настроить кнопки, используя параметры, которые обсуждались ранее в этом разделе.

Предоставление подсказок с помощью функции автозаполнения

Многие сайты, содержащие строку поиска, предусматривают полезную функцию, которая предлагает варианты, соответствующие тому, что вы ввели в эту строку. Например, посетите сайт **ozon.ru** и введите в строку поиска слово «Лампы», и вы увидите раскрывающийся список, где перечислены категории продуктов, которые содержат слово «Лампы»: лампы светодиодные, настольные, энергосберегающие и т. д. (рис. 10.7). Вместо того чтобы вводить остальную часть запроса, вы можете щелкнуть по одному из предложенных вариантов или использовать клавиши ↓ и ↑, чтобы сделать свой выбор.

Эта функция называется «автозаполнением», и плагин jQuery UI предусматривает полезный виджет, который позволяет добавить эту функцию на ваш сайт. Вы можете добавить функцию автозаполнения к любому текстовому полю в форме. Например, вы создаете систему для бронирования авиабилетов, и клиенту необходимо указать аэропорт, из

которого он собирается вылетать. Когда он начинает вводить название аэропорта, появляется раскрывающийся список с вариантами, которые соответствуют нескольким введенным буквам. Затем пользователь может просто щелкнуть по одному из пунктов раскрывающегося списка вместо того, чтобы вводить полное название.

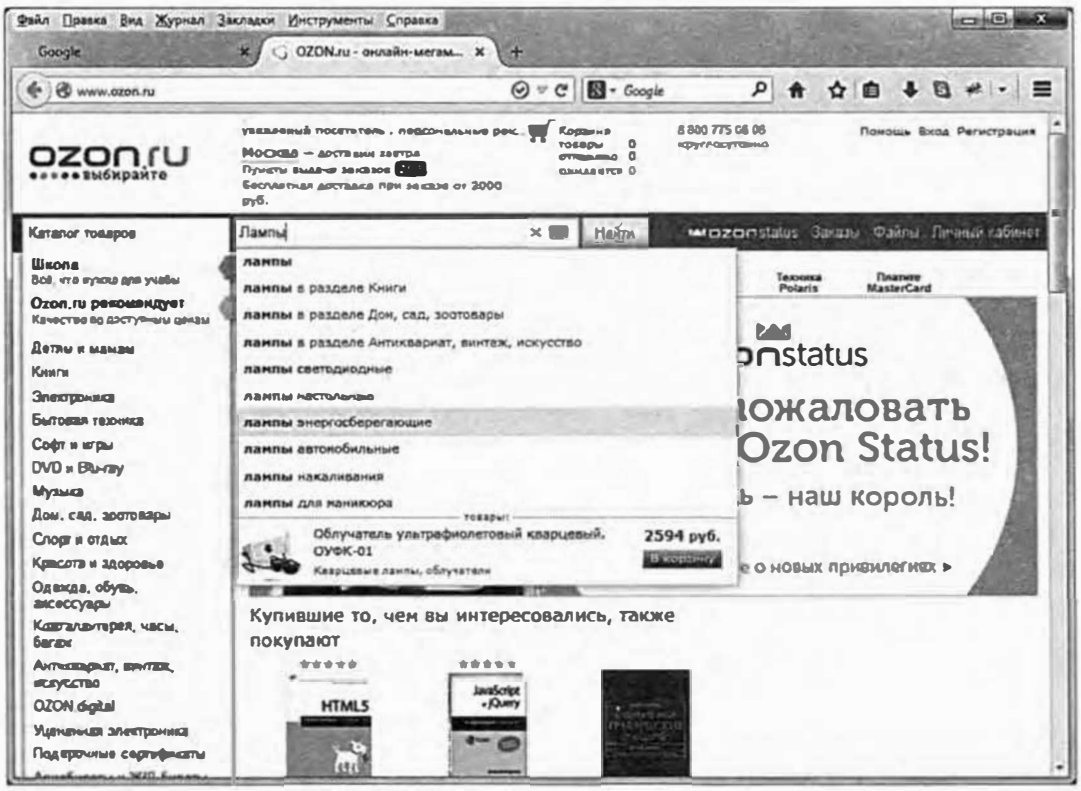

Рис. 10.7. Ваши пальцы еще не болят от постоянного печатания? Многие вебсайты пытаются облегчить для вас этот процесс, предлагая поисковые запросы, которые соответствуют тому, что вы начали вводить

Более того, вы можете использовать функцию автозаполнения для помещения другого содержимого в текстовое поле. В примере с бронированием авиабилетов клиент может ввести название города, из которого он вылетает, а при выборе названия из списка в форму добавляется код аэропорта. Например, при вводе слова «Портленд» может отобразиться список названий аэропортов, включающий пункт «Международный аэропорт Портленда». Когда клиент выбирает этот вариант, в текстовом поле отображается код аэропорта — PDX. Вы можете использовать эту функцию, чтобы найти номер товара в каталоге. Когда кто-то вводит название продукта и выбирает его наименование из раскрывающегося списка, в текстовое поле добавляется номер этого продукта.

Чтобы использовать виджет Autocomplete, необходимо предоставить плагину jQuery UI данные, которые он может использовать для

поиска соответствий, отображаемых в раскрывающемся списке автозаполнения. Например, вам нужен список названий аэропортов, чтобы плагин jQuery UI мог найти соответствие тому, что посетитель ввел в поле формы. Существуют два способа это сделать: создать массив с помощью JavaScript или отправить поисковый запрос на веб-сервер, используя технологию Ajax, и дождаться от веб-сервера списка с соответствующими данными. Вы ознакомитесь с обоими подходами далее в этой главе, однако сначала разберемся с методом добавления функции автозаполнения к форме:

1. **Следуйте инструкции, описанной в разделе «Добавление jQuery UI на веб-страницу» главы 9, чтобы прикрепить файлы CSS и JavaScript плагина jQuery UI к своей веб-странице.**

   Вам также потребуется прикрепить файл jQuery, чтобы при использовании виджета jQuery UI к вашей странице были прикреплены файлы CSS jQuery UI, JavaScript jQuery и JavaScript jQuery UI (именно в таком порядке).

2. **Добавьте на страницу форму и текстовое поле:**

   ```
   <input type="text" id="airport" name="airport">
   ```

   Вы также должны предусмотреть способ выбора раскрывающегося списка с помощью jQuery. Например, примените к полю идентификатор.

3. **Добавьте на страницу функцию `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // окончание ready
   ```

   Вам нужно сделать это, только если вы размещаете свой программный код JavaScript в разделе заголовка страницы, как описано в разделе «Альтернатива для функции `$(document).ready()`» главы 5.

4. **Используйте jQuery для выбора текстового поля и вызова функции автозаполнения:**

   ```
   $(document).ready(function() {
   $('#airport').autocomplete();
   }); // окончание ready
   ```

   По крайней мере, вы должны передать функции `autocomplete()` объект, содержащий свойство `source`, а также либо массив элемен-

тов, либо URL-адрес серверной программы, которая может возвратить список вариантов, соответствующих тому, что ввел посетитель. Следующие разделы посвящены описанию двух способов, позволяющих это сделать, — вы будете использовать либо массив, либо серверный сценарий.

Использование массивов с виджетом Autocomplete

Виджет Autocomplete требует наличия перечня терминов, которые используются плагином jQuery UI для поиска совпадений с тем, что ввел посетитель. Самый простой способ заключается в передаче массива элементов, которые будет использовать jQuery UI. Например, у вас есть вопрос: «Какой ваш любимый оттенок красного?» Вы можете создать массив с названиями цветов и передать его функции `autocomplete()` следующим образом:

```
var colors = ['Barn Red', 'Beetroot', ↵
'Brick', 'Bright Maroon', 'Burgundy']; ↵
$('#redInput').autocomplete( { source : colors } );
```

Другими словами, вы начинаете с создания массива `colors`. Затем вы передаете функции `autocomplete()` объект со свойством `source` и значением, которое соответствует названию массива: `source : colors }`.

После того как этот код будет выполнен, а посетитель введет в текстовое поле латинскую букву «B», появится раскрывающийся список, содержащий все элементы массива: Barn Red, Beetroot, Brick, Bright Maroon и Burgundy (рис. 10.8 вверху). Однако если затем посетитель введет букву «r», то есть в поле ввода будет введено «Br», то он увидит только два цвета, в названиях которых присутствуют буквы «Br»: Brick и Bright Maroon (рис. 10.8 внизу).

Тем не менее массив, содержащий пять элементов, не представляет большой ценности. В конце концов, если бы вариантов было так мало, вы могли бы просто использовать раскрывающийся список, содержащий эти пункты.

Полезность виджета Autocomplete проявляется в полной мере, когда вы можете предоставить своим посетителям множество вариантов — больше, чем помещается в обычном раскрывающемся списке. При нали-

чии большого количества вариантов хорошей идеей является создание отдельного файла JavaScript, который содержит только те данные, которые вы хотите использовать в качестве параметра `source` для виджета. Например, вам нужно предусмотреть функцию автозаполнения для текстового поля, в которое необходимо ввести название аэропорта.

Рис. 10.8. Виджет jQuery UI Autocomplete облегчает ввод распространенных поисковых запросов, предоставляя подсказки, соответствующие тому, что ввел посетитель

Вы можете создать отдельный файл JavaScript, например, *airports.js*, который создает массив и присваивает ему значения:

```
var airports = [
'Aberdeen Regional Airport, Aberdeen, South Dakota', ↵
'Abilene Regional Airport, Abilene, Texas', 'Abraham ↵
Lincoln Capital Airport, Springfield, Illinois', 'Adak↵
Airport, Adak Island, Alaska', 'Adirondack Regional ↵
Airport, Saranca Lake, New York'
]; // здесь могут быть перечислены другие аэропорты
```

ПРИМЕЧАНИЕ

Это только начало списка американских аэропортов! В реальной жизни в приведенном выше коде у вас бы получился очень длинный список, то есть очень большой массив с названиями.

Этот файл оказался бы очень большим, и вам не потребовалось бы использовать этот массив на каждой странице вашего сайта, поэтому его лучше было бы поместить в отдельный файл и сослаться на него только на странице, на которой используется виджет Autocomplete. Этот файл прикрепляется к странице так же, как и любой другой файл JavaScript (см. раздел «Как добавить JavaScript на страницу» главы 1). Например:

```
<script src="airports.js"></script>
```

Вы должны убедиться, что этот файл прикреплен к странице до того как использовать виджет Autocomplete, поскольку вам необходимо загрузить массив до вызова функции `autocomplete()`. Например, JavaScript-код в разделе заголовка веб-страницы выглядел бы примерно так:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
<script src="js/jquery.min.js"></script>
<script src="js/jquery-ui.min.js"></script>
<script src="js/airports.js"></script>
<script>
$(document).ready(function() {
$('#airport').autocomplete( { source : airports} );
}); // окончание ready
</script>
```

Обратите внимание на то, что файл *airports.js* загружается перед выполнением функции `autocomplete()`, что позволяет гарантировать то, что массив `airports` сначала создается, а затем передается виджету.

Использование отдельных меток и значений

Выбор варианта из раскрывающегося списка автозаполнения дает jQuery UI инструкцию вписать этот вариант в текстовое поле. В примере с формой, изображенной на рис. 10.8, если посетитель после ввода латинской буквы «B» щелкнет по варианту Bright Maroon, то плагин jQuery UI заполнит это текстовое поле словами Bright Maroon.

Вы также можете дать плагину jQuery UI инструкцию написать что-то отличное от того, что отображается в раскрывающемся списке авто-

заполнения. Вернемся к примеру с аэропортами и допустим, что посетитель ввел в текстовое поле Ab. Виджет Autocomplete отобразит список, включающий три аэропорта: Aberdeen Regional Airport, Abilene Regional Airport и Abraham Lincoln Capital Airport. Вы можете дать плагину jQuery UI задание вписать код аэропорта вместо его реального названия, так что если посетитель выбрал вариант Abilene Regional Airport, то вы можете приказать плагину jQuery UI написать в текстовом поле код ABI.

Чтобы это сделать, вы предоставляете массив объектов. Каждый объект содержит два свойства — `label` и `value`. Метка (`label`) — это то, что видит посетитель во всплывающей форме автозаполнения, а значение (`value`) — это то, что фактически добавляет плагин jQuery UI в элемент формы после того, как посетитель сделает выбор. В примере с аэропортами метка окажется названием аэропорта, а значение будет представлять собой код аэропорта. Например, объект, определяющий аэропорт Abilene Airport, станет выглядеть следующим образом:

```
{
label : 'Abilene Regional Airport, Abilene, Texas',
value : 'ABI'
}
```

А массив со значениями будет начинаться так:

```
var airports = [
{
label : 'Aberdeen Regional Airport, Aberdeen, South ↵
Dakota',
value : 'ABR'
},
{
label : 'Abilene Regional Airport, Abilene, Texas', ↵
value : 'ABI'
},
{
label : 'Abraham Lincoln Capital Airport, ↵
Springfield, Illinois',
value : 'SPI'
```

```
},
{
label : 'Adak Airport, Adak Island, Alaska',
value : 'ADK'
},
{
label : 'Adirondack Regional Airport, Saranca Lake, ↵
New York',
value : 'SLK'
}
]; // здесь могут быть перечислены другие аэропорты
```

ПРИМЕЧАНИЕ

Опять же, это не полный список названий аэропортов США и их кодов. Реальный файл *airports.js* был бы огромным.

Каждый из объектов в этом массиве имеет два свойства. Вы точно так же передаете массив функции `autocomplete()`, плагин jQuery UI запрограммирован так, что он изменяет свой принцип работы, когда получает массив объектов.

```
$('#airport').autocomplete( { source : airports} );
```

Когда виджету Autocomplete передается массив объектов, он пытается сопоставить введенные посетителем данные с соответствующей меткой в каждом объекте. Однако когда посетитель делает выбор из раскрывающегося списка, в форму вставляется второе свойство выбранного объекта — `value`. Вы увидите пример этого в шаге 12 следующего руководства.

Получение данных для функции автозаполнения с сервера

Виджет Autocomplete наиболее полезен, когда у вас есть много данных, среди которых вы можете произвести поиск. Помните приведенный в одном из предыдущих разделов пример, где вы предоставляли всего шесть оттенков красного цвета? Такое небольшое количество вариантов не сильно поможет пользователю в процессе поиска. Создать файл с очень

большим массивом не так просто, а во многих случаях — просто невозможно. Например, работа функции автозаполнения на сайте **ozon.ru**, конечно, не зависит от одного большого файла JavaScript, содержащего список всех товарных категорий и их описаний: такой файл был бы огромным, и на его загрузку из Интернета уходило бы очень много времени.

На многих сайтах, которые включают функцию автозаполнения (например, **yandex.ru** и **ozon.ru**), используется программный код на стороне сервера для отправки гораздо меньшего списка с вариантами для автозаполнения. Вот как это работает:

1. **Посетитель начинает вводить запрос в текстовое поле.**
2. **Введенные посетителем данные отправляются на сервер.**

   Используя технологию Ajax, браузер отправляет данные на сервер и ожидает ответа. В этом случае браузер отправляет текст, который успел ввести посетитель.

3. **Сервер отправляет обратно массив поисковых запросов, которые соответствуют тому, что ввел посетитель.**

   Сервер предоставляет список подходящих терминов. Как правило, это обеспечивается программным кодом на стороне сервера, который производит поиск по базе данных, получает результаты и форматирует эти результаты в виде массива, который посылается обратно браузеру.

4. **Список соответствующих терминов отображается в раскрывающемся списке автозаполнения.**

   Как видите, большая часть этого волшебства обеспечивается благодаря технологии Ajax, которая позволяет использовать код JavaScript для отправки и получения информации с веб-сервера без необходимости перезагрузки новой веб-страницы (вы узнаете подробнее о технологии Ajax в главе 13). Программный код на стороне сервера не является предметом данной книги, здесь не обсуждаются серверные программы, созданные на языке PHP, Ruby, Python и даже JavaScript (с использованием Node.js). За этой информацией вам придется обратиться к другому источнику. Тем не менее далее описан процесс использования виджета Autocomplete jQuery UI для общения с сервером: вместо того, чтобы предоставлять массив возможных поисковых запросов для свойства `source`, вы предоставляете URL-адрес, указывающий на серверный сценарий.

Например, представьте, что у вас есть текстовое поле, которое посетители используют для выполнения поиска в вашем каталоге, содержащем описание 250 000 товаров. Чтобы облегчить посетителям процесс поиска, вы решили добавить в эту область функцию автозаполнения. Однако ваш каталог товаров является слишком большим, чтобы иметь возможность поместить весь его материал в один массив в файле JavaScript. Вместо этого вы создаете программу на стороне сервера *products.php*, расположенную в корневом каталоге вашего сайта. Затем вы можете указать путь к этому файлу, используя свойство `source`. Предположим, что вы присвоили элементу формы идентификатор `"productSearch"`:

```
<input type="text" id="productSearch" ↵
name="productSearch">
```

В коде JavaScript вашей страницы вы можете применить виджет Autocomplete к этому элементу и указать на серверную страницу так:

```
$('#productSearch').autocomplete( { source : ↵
'/products.php'} );
```

Вы также можете предоставить полный URL-адрес, включая протокол и доменное имя:

```
$('#productSearch').autocomplete( { source : ↵
'http://myCompany.com/products. php'} );
```

Когда вы предоставляете массив элементов для параметра `source` (описан ранее в данной главе), плагин jQuery UI фильтрует этот массив с целью нахождения элементов, содержащих текст, введенный посетителем, и отображает только соответствующие этому тексту элементы массива. Виджет Autocomplete ведет себя немного по-другому, когда для параметра `source` передается URL-адрес. Плагин jQuery UI отображает все данные, возвращенные сервером, и не фильтрует эти данные с целью нахождения соответствий. Например, если посетитель ввел текст «Светлый», а сервер возвратил массив, содержащий элементы «Темный», «Артишок» и «Страус», то jQuery UI создаст раскрывающийся список с соответствующими пунктами.

Другими словами, возвращение корректных данных полностью зависит от сервера. Это означает, что вы (или другой программист) должны написать всю логику для генерирования надлежащего списка и его отправки браузеру. Чтобы помочь в решении этой задачи, плагин jQuery

UI добавляет параметр URL, чтобы сообщить серверу, что ему нужно искать. Этот параметр называется `term`, и его значением является то, что посетитель ввел в поле. В примере с поиском товара, если бы посетитель ввел слово «свет», то плагин jQuery UI отправил бы запрос на сервер следующим образом:

```
http://myCompany.com/products.php?term=light
```

Затем страница *products.php* выполняет над этими данными некоторые действия, например, производит в базе данных поиск товаров, которые так или иначе связаны с лампами. Это может быть буквальный поиск, в результате которого отображаются только товары, в названиях которых присутствует слово «свет»: «светильник», «светодиодная лента» и т. д. Более полная программа, предоставляющая список товаров, имеющих отношение к понятию света, может предоставить такие варианты, как «люминесцентные лампы», «антигравитационные машины» и т. д. Программист решает, как обрабатывать поисковый запрос и какие данные отправлять обратно.

ПРИМЕЧАНИЕ

Также можно использовать функцию автозаполнения для того, чтобы указать на другой домен и веб-сервер. Тем не менее из-за ограничений безопасности, предусмотренных технологией Ajax, сервер должен отвечать, используя специальный формат под названием JSONP, подробнее о котором вы можете узнать в разделе «Знакомство с форматом JSONP» главы 13.

Программа на стороне сервера должна вернуть список элементов, отформатированных в виде массива:

```
 [
"светильник",
"светодиодная лампа",
"светодиодная лента"
]
```

Кроме того, сервер может возвратить данные в виде массива объектов, содержащих метку и значение, как описано в разделе «Использование массивов с виджетом Autocomplete» ранее в этой главе. Тем не менее эти данные должны использовать формат JSON, обсуждаемый в разделе

«Формат JSON» главы 13. То есть, все имена свойств и строки должны быть заключены в двойные кавычки:

```
var airports = [
{
"label" : "Aberdeen Regional Airport, Aberdeen, ↵
South Dakota",
"value" : "ABR"
},
{
"label" : "Abilene Regional Airport, Abilene, Texas", ↵
"value" : "ABI"
},
{
"label" : "Abraham Lincoln Capital Airport, ↵
Springfield, Illinois",
"value" : "SPI"
},
{
"label" : "Adak Airport, Adak Island, Alaska", ↵
"value" : "ADK"
},
{
"label" : "Adirondack Regional Airport, ↵
Saranca Lake, New York",
"value" : "SLK"
}
];
```

Параметры автозаполнения

Параметров, контролирующих работу виджета Autocomplete, немного. Самый важный (и обязательный) параметр `source` описан в предыдущих разделах. Тем не менее существует несколько других полезных параметров. Как и в случае с параметрами других виджетов, параметры виджета Autocomplete должны передаваться функции `autocomplete()` как часть объекта. Например:

```
$('#airport').autocomplete({
source : '/airportSearch.php',
delay : 500,
minLength : 2
});
```

Далее перечислены наиболее полезные параметры:

- **source**. Наиболее важный и единственный обязательный параметр. Используется для передачи массива или URL-адреса страницы на стороне сервера. Массив содержит либо значения, либо объекты со свойствами `label` и `value` (см. раздел «Использование отдельных меток и значений» ранее в главе). В случае с URL-адресом программа на стороне сервера должна возвратить данные, отформатированные в виде массива (см. раздел «Использование массивов с виджетом Autocomplete» ранее в главе).
- **delay**. При использовании программы на стороне сервера в качестве источника данных для автозаполнения плагин jQuery UI посылает запрос на сервер каждый раз, когда посетитель вводит что-либо в поле. Многочисленные быстрые запросы могут замедлить работу сервера и скорость отклика виджета Autocomplete. Вы можете настроить небольшую задержку, чтобы снизить нагрузку на веб-сервер, передав численное значение этого параметра в миллисекундах. Например, чтобы подождать полсекунды перед отправкой запроса, вы можете настроить данный параметр следующим образом:

  ```
  delay : 500
  ```

- **minLength**. Этот параметр определяет минимальное количество букв, которое необходимо ввести, прежде чем виджет Autocomplete начнет предлагать варианты. Если источники данных для автозаполнения содержат сотни тысяч записей, то вы, вероятно, решите установить в качестве минимальной длины три символа. В конце концов, если посетитель введет в поле букву «а», то он может получить тысячи подсказок из крупного источника данных.

ПРИМЕЧАНИЕ

Вы можете найти полный список параметров, а также дополнительные функции виджета Autocomplete на сайте: **api.jqueryui.com/autocomplete/**.

Использование виджета jQuery UI Form на практике

В этом руководстве вы начнете с простой формы для бронирования авиабилета: она содержит текстовые поля, переключатели и элемент `button` (рис. 10.9 вверху). Добавив на страницу плагин jQuery UI и применив виджет jQuery UI Form, мы получим более привлекательную, интерактивную и функциональную форму (рис. 10.9 внизу).

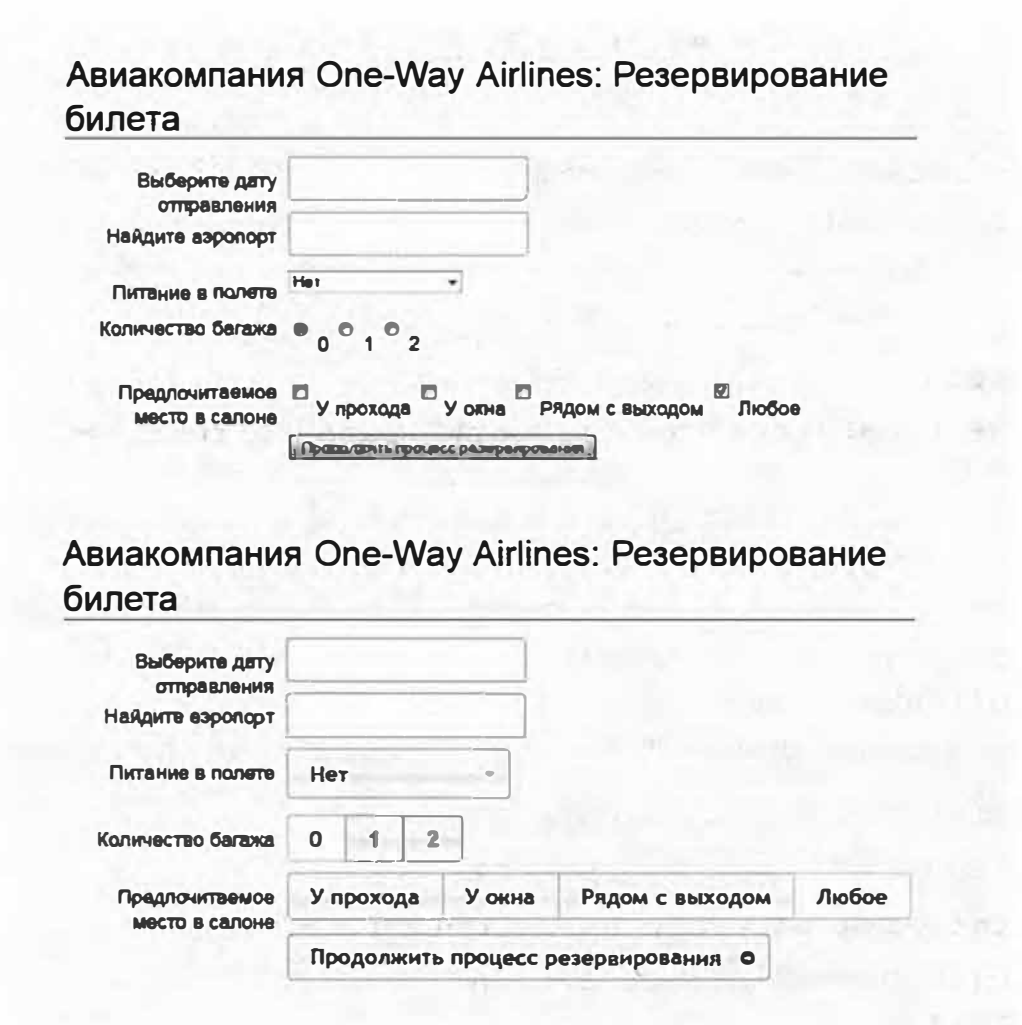

Рис. 10.9. Плагин jQuery UI позволяет обеспечить соответствие внешнего вида невзрачной формы и всего вашего приложения. Он может превратить элементы управления формы, например, раскрывающиеся списки, переключатели и флажки (которые обычно стилизуются браузером) в привлекающие внимание элементы пользовательского интерфейса

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *10_02.html* в папке *глава10*.**

   Этот документ уже содержит ссылку на все файлы jQuery и jQuery UI, а также функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5). HTML-код для формы на этой странице выглядит следующим образом:

```
<form>
<div>
<label for="departure" class="label">Выберите дату ↵
отправления</label>
<input type="text" id="departure" name="departure">
</div>
<div>
<label for="airport" class="label">Найдите аэропорт ↵
</label>
<input type="text" id="airport" name="airport">
</div>
<div>
<label for="meal" class="label">Питание в полете ↵
</label>
<select name="meal" id="meal">
<option>Нет</option>
 <option>Для веганов</option>
 <option>Без глютена</option>
 <option>Для вегетарианцев</option>
 <option>Для мясоедов</option>
</select>
</div>
<div id="bags">
<p class="label">Количество багажа</p>
<input type="radio" id="none" name="bags" ↵
checked="checked">
<label for="none">0</label>
<input type="radio" id="one" name="bags">
<label for="one">1</label>
```

```
<input type="radio" id="two" name="bags">
<label for="two">2</label>
</div>
<div id="seatTypes">
<p class="label">Предпочитаемое место в салоне</p>
<input type="checkbox" id="aisle" name="aisle">
<label for="aisle">У прохода</label>
<input type="checkbox" id="window" name="window">
<label for="window">У окна</label>
<input type="checkbox" id="exit" name="exit">
<label for="exit">Рядом с выходом</label>
<input type="checkbox" id="any" name="any">
<label for="any">Любое</label>
</div>
<div>
<button id="next"> Продолжить процесс резервирования ↵
</button>
</div>
</form>
```

Важные элементы формы — те, к которым вы примените плагин jQuery UI, выделены полужирным шрифтом. Вы начнете с добавления панели для выбора даты в первый элемент формы. При выполнении следующего шага обратите внимание на то, что элемент `input` имеет идентификатор `departure`.

2. **Внутри функции `$(document).ready()` выберите элемент ввода:**

```
$('#departure')
```

Теперь вам нужно применить виджет Datepicker.

3. **Введите точку и фрагмент `datepicker();`, чтобы ваш код выглядел следующим образом:**

```
$('#departure').datepicker();
```

Вот и все, что необходимо для добавления на страницу всплывающего календаря для выбора даты. Тем не менее если вы сейчас сохраните файл и просмотрите его в браузере, вы увидите, что можете

выбрать даты до сегодняшнего дня. Очевидно, что вы не можете попасть на рейс, произошедший на прошлой неделе, поэтому вам нужно удостовериться в том, что вы не можете забронировать билет на дату, предшествующую сегодняшней.

4. **Установите текстовый курсор внутри функции `datepicker()` и введите `{`. Дважды нажмите клавишу Enter и введите `}`, чтобы код выглядел следующим образом:**

```
$('#departure').datepicker({

});
```

Символы `{}` представляют собой пустой литерал объекта, и вы зададите параметры для виджета, добавив внутри него свойства. Во-первых, установите самую раннюю дату для бронирования билета.

5. **В пустой строке внутри литерала объекта введите `minDate : 0`.**

Значение параметра `minDate` указывает количество дней, начиная с сегодняшнего дня: `0` соответствует сегодняшнему дню, `-7` означает неделю назад, а `7` — через неделю.

Вы также можете указать максимально допустимую для выбора дату. В данном случае авиакомпания не отслеживает рейсы, которые должны состояться через год, начиная с текущего дня, поэтому вам нужно предотвратить возможность выбора более поздней даты.

6. **Добавьте запятую в конце последней введенной строки, нажмите клавишу Enter и введите `maxDate : '+1y'`, чтобы код выглядел следующим образом:**

```
$('#departure').datepicker({
minDate : 0,
maxDate : '+1y'
});
```

Параметры `minDate` и `maxDate` позволяют использовать число, чтобы указать конкретное количество дней, однако вы также можете использовать строковое значение, которое включает число и букву для указания продолжительности: фрагмент `'+1y'` означает через 1 год, `'-2w'` — 2 недели назад, а `'+1m +10d'` означает через 1 месяц и 10 дней.

7. **Сохраните страницу и просмотрите ее в браузере. Щелкните по полю «Выберите дату отправления».**

Появится всплывающий календарь. Обратите внимание на то, что диапазон доступных для выбора дат ограничен сегодняшним днем и датой, отстоящей от текущей на один год (если вы хотите убедиться в этом, щелкните по кнопке в виде указывающей вперед стрелки, чтобы перейти на 12 месяцев вперед).

Далее, вы будете использовать виджет Autocomplete для облегчения процесса выбора аэропорта. Однако сначала посмотрите на источник данных, который вы будете использовать. Как говорилось в разделе «Использование массивов с виджетом Autocomplete» данной главы, функция `autocomplete()` может принимать либо массив JavaScript с данными, либо данные, отправленные из серверного сценария, работающего на веб-сервере. Чтобы не усложнять этот пример, вы будете использовать небольшой объем данных, хранящихся в отдельном JavaScript-файле.

8. **В текстовом редакторе откройте файл *airports.js*.**

   Это простой файл JavaScript, который создает массив с именем `airports`, наполненный объектами, состоящими из двух свойств: `label` и `value`. Метка (`label`) окажется отображена в раскрывающемся списке автозаполнения, а значение (`value`) будет вписано в текстовое поле, когда посетитель выберет пункт из списка.

   Очевидно, что этот файл содержит не полный список, а просто небольшой объем данных для того, чтобы испробовать виджет. Чтобы получить доступ к данным в этом файле, вам нужно привязать его к странице.

9. **В текстовом редакторе откройте файл *10_02.html*. и под строкой `<script src ="../_js/jquery-ui.min.js"></script>` добавьте еще один элемент `script`, чтобы привязать файл с данными:**

   ```
   <script src="airports.js"></script>
   ```

   Это строка загружает внешний файл JavaScript, после чего выполняется код JavaScript. В данном случае файл просто создает массив данных, которые вы затем можете использовать с виджетом Autocomplete.

10. **Добавьте пустую строку после добавленного ранее кода виджета DatePicker и введите:**

    ```
    $('#airport').autocomplete({ source : 'airports.json'});
    ```

Текстовое поле, предназначенное для ввода названия аэропорта, имеет идентификатор `airport`, поэтому фрагмент `$('#airport')` выбирает этот элемент формы, а часть `.autocomplete()` применяет к нему виджет.

11. **Сохраните страницу и просмотрите ее в браузере. Щелкните по полю «Найдите аэропорт» и введите *Port*.**

    Появится раскрывающийся список автозаполнения с вариантами аэропортов, которые могут соответствовать тому, что вы ищете. Обратите внимание на то, что виджет предлагает аэропорты, в метках которых присутствует фрагмент Port, например, La Guardia Air*port*.

12. **Щелкните по варианту Portland International Airport, Portland, OR.**

    Обратите внимание, что в текстовом поле отображается PDX. Это связано с тем, что вы предоставили массив объектов с метками и значениями. Меткой, которая появляется в выпадающем списке, является Portland International Airport, Portland, OR, а в текстовое поле виджет jQuery UI добавляет код аэропорта «PDX».

    Далее вы превратите этот раскрывающийся список в привлекательный элемент пользовательского интерфейса.

13. **Вернитесь в текстовый редактор к файлу *10_02.html*. Введите следующий код после строки, добавленной в шаге 10:**

    ```
    $('#meal').selectmenu();
    ```

    Возможно, вы хотите добавить дополнительный код, однако это все, что вам нужно сделать для преобразования раскрывающегося списка. Если вы сохраните и просмотрите страницу, вы заметите, что раскрывающийся список выглядит немного странно. Первый пункт раскрывающегося списка отображен не полностью! Если метки в вашем раскрывающемся списке не очень короткие, то вам будет необходимо установить ширину этого виджета.

14. **Между круглыми скобками функции `selectmenu()` введите `{ width : 200 }`, чтобы строка кода выглядела следующим образом:**

    ```
    $('#meal').selectmenu( { width : 200 } );
    ```

    Параметр `width` виджета `selectmenu` (см. раздел «Настройка свойств раскрывающегося списка» данной главы) позволяет установить ширину раскрывающегося списка на странице. В данном

случае 200 означает 200 пикселей в ширину, хотя вы можете использовать другие размеры и единицы измерения, например, проценты или единицы em, как описано в разделе «Настройка свойств раскрывающегося списка» ранее в главе. Сохраните страницу и просмотрите ее в веб-браузере. Кто сказал, что программирование — это сложно?

Далее вы превратите переключатели и флажки в нечто более соответствующее остальной части вашей формы.

15. В текстовом редакторе введите следующие две строки после кода, добавленного в последнем шаге:

```
$('#bags').buttonset();

$('#seatTypes').buttonset();
```

Функция `buttonset()` работает как для переключателей, так и для флажков. Последний шаг заключается в преобразовании элемента `button` в кнопку jQuery UI.

16. Добавьте еще одну строку кода в свою программу:

```
$('#next').button();
```

Этот код превращает невзрачную HTML-кнопку в нечто более соответствующее форме и внешнему виду переключателей и флажков. Вы также можете добавить значки jQuery UI к элементам `button`, как описано в разделе «Настройка кнопок» ранее в главе. Давайте сделаем это в следующем шаге.

17. Внутри круглых скобок функции `button()` добавьте:

```
{
icons : {
secondary : 'ui-icon-circle-arrow-e'
}
}
```

Окончательный код JavaScript должен выглядеть следующим образом:

```
$(document).ready(function() {
$('#departure').datepicker({
minDate : 0,
maxDate : '+1y'
});
```

```
$('#airport').autocomplete({ source : airports});
$('#meal').selectmenu({width : 200}); $('#bags') ↵
.buttonset(); $('#seatTypes').buttonset();
$('#next').button({
icons : {
secondary : 'ui-icon-circle-arrow-e'
}
});
}); // окончание ready
```

С помощью нескольких строк кода вы полностью изменили внешний вид и функциональность веб-формы.

18. Сохраните страницу и просмотрите ее в браузере.

Вид страницы должен соответствовать нижнему изображению на рис. 10.9. Если ваша страница выглядит иначе, убедитесь в том, что ваш код соответствует коду в шаге 17. Вы также можете проверить консоль JavaScript на предмет наличия ошибок (см. раздел «Отслеживание ошибок» главы 1). Тем не менее, имейте в виду то, что библиотека jQuery часто скрывает ошибки из консоли, что может усложнить поиск ошибок в коде jQuery.

Вы можете найти итоговую версию данного руководства в файле *готовый_10_02.html* в папке *глава10*.

Глава 11
НАСТРОЙКА ВНЕШНЕГО ВИДА JQUERY UI

Виджеты jQuery UI имеют единообразный внешний вид — виджет для выбора даты похож на панели с вкладками, которые похожи на диалоговое окно, которое похоже на всплывающую подсказку. Если бы вы собрали несколько разных плагинов jQuery, созданных разными авторами, чтобы создать эти же виджеты, вам пришлось бы повозиться с кодом CSS, чтобы обеспечить соответствие внешнего вида плагинов. Соответствие внешнего вида различных виджетов jQuery UI означает, что вы можете создать веб-приложение с единообразным стилем, не тратя много времени на доработку кода CSS.

Тем не менее, что если внешний вид вашего сайта уже разработан, и вы хотите сделать так, чтобы плагин jQuery UI соответствовал существующему дизайну? Для таких случаев команда jQuery UI предоставила много полезных советов, а также замечательный онлайн-инструмент для дополнительной помощи. В этой главе рассказывается, как переопределить или изменить существующие стили jQuery UI, а также создать новые.

Знакомство с приложением ThemeRoller

В плагине jQuery UI существует много движущихся частей: создание кода CSS, обеспечивающего хороший внешний вид (и визуальное соответствие) — это большая задача. К счастью, команда jQuery UI создала онлайн-инструмент под названием ThemeRoller. Этот инструмент позволяет выбрать одну из 24 тем, разработанных дизайнерами для использования с плагином jQuery UI. Он также предусматривает инструменты для изменения существующей темы, позволяя вам выбирать шрифты и цвета, которые соответствуют внешнему виду вашего сайта.

Чтобы использовать приложение ThemeRoller, посетите страницу **jqueryui.com/themeroller/** (рис. 11.1). Вы можете просмотреть коллекцию заранее разработанных тем, щелкнув по вкладке **Gallery** (Галерея)

в левой боковой панели (обведена). Основная часть страницы демонстрирует различные виджеты jQuery UI: выбор темы из списка слева немедленно применяет эту тему к виджетам на странице, обеспечивая предварительный просмотр их дизайна.

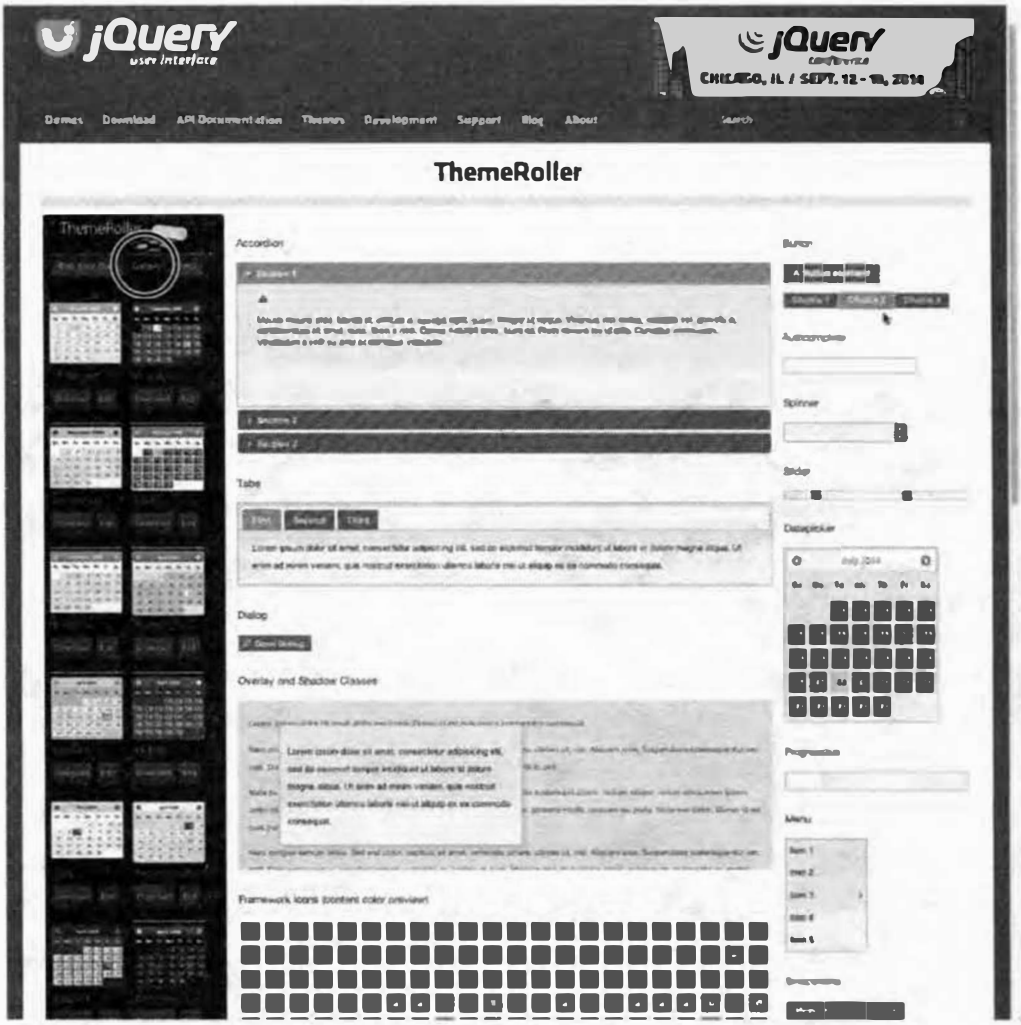

Рис. 11.1. Страница ThemeRoller позволяет скачать уже разработанный дизайн jQuery UI. Вы можете выбрать среди 24 различных дизайнов. Щелкните по кнопке **Download** (Загрузить) для загрузки темы. Вы также можете использовать созданную тему в качестве отправной точки для создания вашего собственного дизайна. Для этого щелкните по кнопке **Edit** (Редактировать) под темой, и приложение ThemeRoller загрузит соответствующий дизайн, после чего вы сможете использовать собственные параметры для настройки цвета и шрифтов

Если вам нравится вид темы в галерее, щелкните по кнопке **Download** (Загрузить) под ее эскизом в боковой панели слева, чтобы перейти к конструктору загрузки jQuery, описанному в разделе «Использование плагина jQuery UI» главы 9. На этой странице выберите нужные вам виджеты, взаимодействия и эффекты, а затем щелкните по кнопке **Download** (Загрузить) в нижней части страницы, чтобы загрузить файлы.

ПРИМЕЧАНИЕ

Подробное описание процесса загрузки и организации файлов jQuery UI для их использования на сайте вы можете найти в разделе «Использование плагина jQuery UI» главы 9.

Если вы хотите добавить плагин jQuery UI на сайт, который вы уже создали, вы можете использовать приложение ThemeRoller для создания собственной темы, которая соответствует дизайну вашего сайта. Щелкните по вкладке **Roll Your Own** (Создайте собственную тему) в левой боковой панели (выделена на рис. 11.2): этот инструмент предоставляет доступ к параметрам шрифтов, цветов и т. д.

Рис. 11.2. Чтобы открыть категорию **Font Settings** (Параметры шрифта), щелкните по стрелке рядом с названием категории. Обычно все категории закрыты, однако на приведенном изображении они открыты и расположены бок о бок (чтобы поместиться на этой странице). На сайте ThemeRoller эти категории фактически расположены друг над другом

Параметры сгруппированы по категориям, которые можно открыть или закрыть, щелкнув по стрелке слева от названия категории. Далее приведен список категорий и их параметров:

- Категория **Font Settings** (Параметры шрифта) позволяет указать шрифты, а также их размер и начертание. Введите название шрифта

(ов), которые вы хотите использовать, в поле **Family** (Семейство). Обычно шрифты указываются в виде группы, включающей три варианта: шрифт, который вы хотите загрузить первым, резервный шрифт, а затем запасной шрифт. Если первый шрифт недоступен на компьютере вашего посетителя, то используется второй шрифт. Если недоступен и этот шрифт, то выбирается шрифт общего типа (`serif` (с засечками), `sans-serif` (без засечек), `monospace` (моноширинный) или `fantasy` (фантазийный)).

Используйте те же шрифты, которые вы используете на своем сайте. Например, если на вашем сайте в качестве основного шрифта используется Helvetica Neue, то вы можете ввести следующее: "Helvetica Neue", Arial, sans-serif. (При использовании шрифта, название которого состоит более чем из одного слова, например, Helvetica Neue, вы должны заключить название шрифта в кавычки.) Плагин jQuery UI применит один и тот же шрифт ко всем виджетам и их составляющим, включая вкладки и панели. В противном случае, вы можете указать различные шрифты для разных элементов, как описано далее в разделе «Переопределение стилей jQuery UI».

ПРИМЕЧАНИЕ

Если вы используете веб-шрифт, который находится на вашем сайте или предоставляется таким сервисом, как Typekit или Google Fonts, или любой шрифт, отсутствующий на вашем компьютере, то этот шрифт не отобразится в области предварительного просмотра ThemeRoller.

Параметр **Weight** (Вес) определяет начертание шрифта и может принимать такие значения, как **normal** (нормальный) или **bold** (полужирный). Выбор значения **bold** не приведет к тому, что весь текст внутри виджетов jQuery UI окажется выделен полужирным начертанием, оно будет применено только к определенным элементам, например, к тексту на вкладках, кнопках, в раскрывающихся списках и заголовках аккордеонов. К остальному тексту будет применено «нормальное» начертание.

Размер шрифта задается базовым шрифтом для всех виджетов jQuery UI. Внутри некоторых виджетов текст окажется больше. Например, размер текста в заголовке диалогового окна и в подсказках будет превышать размер текста в других виджетах.

- **Corner radius** (Радиус закругления). Большинство виджетов jQuery UI имеет углы, а данный параметр позволяет сделать эти углы более или менее скругленными. Значение 0 создает прямые углы, более высокие значения делают углы более округлыми. Вводите различные значения, пока не найдете то, которое вас устраивает.

- Категория **Header/Toolbar** (Заголовок/Панель инструментов) позволяет задать цвета и узоры для заголовка панели выбора даты или диалогового окна, а также цвет в панели прогресса или слайдере. Эта категория предусматривает шесть различных параметров: цвет фона, текстура фона, непрозрачность текстуры фона, цвет границы, цвет текста и цвет значка. Эти же шесть параметров также отображаются в большинстве других категорий, поэтому с ними следует ознакомиться:

- **Background color** (Цвет фона). Щелкните по этому элементу управления, чтобы отобразить панель для выбора цвета. Используйте эту панель для выбора оттенка (щелкните по внешнему колесу), а затем выберите определенный цвет (щелкните по внутренней рамке). См. рис. 11.3.

- **Background texture** (Текстура фона). Плагин jQuery UI может применять различные узоры в качестве фона элемента, например, горизонтальные или диагональные полосы. Щелкните по текстовому полю, чтобы открыть палитру текстур. Щелкните по понравившемуся варианту или по образцу в верхнем левом углу, чтобы не назначать никакую текстуру. Текстура создается с помощью небольшого файла изображения на заднем плане. См. рис. 11.3.

- **Background texture opacity** (Непрозрачность текстуры фона) определяет степень непрозрачности узора. Установите значение 0, и узор вообще не будет отображен. Значение 10 добавляет слегка заметный узор, а значение более 75% заставляет узор четко выделяться. См. рис. 11.3.

- **Border color** (Цвет границы) задает цвет границы вокруг заголовка или панели инструментов. Установите флажок для этого параметра, и появится панель выбора цвета. Если вам не нужны границы, просто выберите цвет, соответствующий цвету фона, и граница сольется с ним. (Та же самая панель для выбора цвета, которая отображается для всех связанных с цветом параметров, изображена на рис. 11.3.)

- **Text color** (Цвет текста). Щелкните по этой кнопке, чтобы открыть палитру и выбрать цвет текста, который будет использоваться внутри заголовка или панели инструментов.

Рис. 11.3. Многие из категорий приложения ThemeRoller jQuery UI позволяют установить одни и те же три свойства: цвет фона, текстуру и непрозрачность текстуры. Познакомьтесь с этими элементами управления, и вы сможете настраивать стиль фона любого виджета jQuery UI

- **Icon color** (Цвет значка). В некоторых виджетах отображаются значки. Например, виджет Datepicker включает значки «Предыдущий месяц» и «Следующий месяц» во всплывающем календаре. Чтобы задать цвет этих значков, щелкните по образцу цвета и используйте открывшуюся панель, чтобы выбрать цвет, который хорошо сочетается с дизайном вашего сайта.

ПРИМЕЧАНИЕ

Для всех параметров цвета вы можете также щелкнуть по полю с настройками цвета и ввести шестнадцатеричное значение, например, `#e64c4c`.

- **Content** (Содержимое). Данная категория позволяет установить фон, границы и цвета текста для областей контента виджетов, к которым относятся панели аккордеонов и панели с вкладками, пункты меню в виджете Selectmenu и календарь виджета Datepicker. Содержащиеся в данной категории параметры аналогичны параметрам категории **Header/Toolbar** (Заголовок/Панель инструментов): цвет

фона, текстура фона, непрозрачность текстуры фона, а также цвет границы, текста и значков.

- **Clickable items** (Интерактивные элементы). Приложение ThemeRoller предусматривает три категории интерактивных элементов, к которым относятся такие части виджета, по которым вы можете щелкнуть, чтобы совершить какое-либо действие: заголовки аккордеона, вкладки, кнопки, даты на календаре виджета Datepicker, значки, кнопки в диалоговом окне и пункты раскрывающегося списка. Три категории соответствуют трем состояниям интерактивного элемента: «состояние по умолчанию» (внешний вид интерактивного элемента при первой загрузке страницы), «состояние наведения» (когда посетитель наводит указатель мыши на интерактивный элемент) и «активное состояние». Активное состояние относится к выбранной вкладке в панели с вкладками, выбранному заголовку аккордеона, выбранной дате в виджете Datepicker или к выбранной кнопке (например, к переключателю в форме). В каждой категории интерактивных элементов присутствуют те же шесть параметров, что и в описанной выше категории **Header/Toolbar** (Заголовок/Панель инструментов): цвет фона, текстура фона, непрозрачность текстуры фона, а также цвет границы, текста и значков.

- Категории **Highlight** (Выделение) и **Error** (Ошибка) на самом деле сами по себе не форматируют виджеты. Тем не менее они создают два класса CSS: `.ui-state-highlight` и `.ui-state-error`, которые вы можете применить к виджетам или к любому элементу на странице. Например, виджет Dialog (см. раздел «Создание диалоговых окон с сообщениями» главы 9) позволяет назначить имя класса диалоговому окну. Поэтому, если бы вам понадобилось открыть особое диалоговое окно со стилем **Highlight** (Выделение), вы бы просто передали имя класса (без точки) функции `dialog()`:

```
$('#dialogDiv').dialog({ dialogClass : ↵
'ui-state-highlight' });
```

Кроме того, вы можете использовать стиль `.ui-state-error`, чтобы отформатировать элемент страницы, который необходимо выделить. Например, если кто-то неправильно заполнил элемент формы, вы можете выбрать метку этого элемента и динамически применить к нему класс `error`:

```
$('#userNameLabel').addClass('ui-state-error');
```

В данном случае вы даже не используете виджет jQuery — только стиль класса, созданный приложением ThemeRoller. Вы можете применить любой класс jQuery UI к другим элементам на странице (об этом рассказывается в следующем разделе).

- **Модальные экраны для наложения**. Вы можете использовать виджет Dialog (см. раздел «Создание диалоговых окон с сообщениями» главы 9) для создания модальных диалоговых окон. Модальное диалоговое окно не позволяет посетителю взаимодействовать с другими элементами страницы до закрытия этого окна. Такие окна используются для отображения важных сообщений и действий, например, «Вы уверены, что хотите безвозвратно удалить свой профиль на Facebook?» При открытии модального диалогового окна плагин jQuery UI затемняет остальную часть экрана путем добавления полупрозрачного «наложения» между модальным окном и страницей. Вы можете управлять внешним видом этого экрана с помощью таких параметров, как цвет фона, текстура и непрозрачность (эти же параметры используются на настройки фона в описанной выше категории **Header/Toolbar** (Заголовок/Панель инструментов)).
- **Тени**. Не обращайте внимания на эту категорию: она не применяется к виджетам jQuery UI, даже к виджету Tooltip, у которого есть тень. Странно, но факт.

СОВЕТ

При использовании приложения ThemeRoller для создания нового дизайна jQuery UI URL-адрес в адресной строке браузера изменяется: каждая модификация темы сопровождается изменением URL-адреса. Почему? Потому что приложение ThemeRoller создает URL-адрес, который будет точно дублировать вашу собственную тему. Это означает, что вы можете добавить этот URL-адрес в закладки, чтобы иметь возможность легко вернуться к приложению ThemeRoller со всеми вашими параметрами. Вы также можете скопировать ссылку и отправить ее другу или коллеге, чтобы он мог быстро повторить созданную вами тему.

Загрузка и использование новой темы

После создания новой темы jQuery UI щелкните по кнопке **Download Theme** (Загрузить тему), которая находится слева на странице ThemeRoller. После этого вы попадете на страницу **Download Builder**

(Конструктор загрузки) (см. рис. 11.2). Если вам нужны все виджеты и функции, предусмотренные плагином jQuery UI, просто щелкните по кнопке **Download** (Загрузить) в нижней части страницы. Тем не менее если вам нужны только конкретные виджеты, например, Dialog, Datepicker и Tabs, то сбросьте флажки, соответствующие тем виджетам, эффектам и взаимодействиям, которые вас не интересуют, а затем щелкните по кнопке **Download** (Загрузить).

Добавление новой темы на существующий веб-сайт

Если вы уже используете на своем сайте плагин jQuery UI и просто хотите обновить его внешний вид, применив новую тему, то все что вам нужно, — это новые стили. Вы можете повторить процесс, описанный в предыдущем разделе, чтобы создать и загрузить новую тему. Однако на этот раз, когда вы попадаете на страницу **Download Builder** (Конструктор загрузки) (см. рис. 11.2), убедитесь в том, что вы выбрали те же варианты (виджеты, взаимодействия и эффекты), которые вы используете в данный момент. Если вы этого не сделаете, то у вас окажется либо больше, либо меньше кода CSS, чем вам нужно.

Например, если на своем сайте вы используете все виджеты jQuery UI, но на странице конструктора загрузки сбрасываете флажок, соответствующий виджету Dialog, то вы не скачаете CSS-код, необходимый для стилизации данного виджета. Точно так же, если вы используете только виджет Dialog, но загружаете весь набор функций jQuery UI, то в новом файле CSS будет содержаться намного больше CSS-кода, чем вам на самом деле нужно, а это значит, что посетителям придется тратить время на скачивание ненужного кода CSS.

После загрузки новой копии плагина jQuery UI у вас будут файлы, которые вам не нужны (например, если вы уже используете jQuery UI, то вам не нужно заменять имеющиеся у вас файлы JavaScript). Вам нужно заменить только папку *images* и файлы CSS:

1. **Замените старый файл или файлы CSS плагина jQuery UI новой темой.**

   Как уже упоминалось в разделе «Добавление jQuery UI на веб-страницу» главы 9, плагин jQuery UI включает шесть CSS-файлов. Большинство из них является дубликатами в различных форматах. По-настоящему вам нужен только один файл — *jquery-ui.min.css*. Этот документ содержит весь необходимый jQuery код CSS в умень-

шенном (сжатом) файле. Как правило, рекомендуется использовать файлы, в именах которых содержится фрагмент *.min*, поскольку они имеют наименьший размер. Поместите новый файл CSS туда, где находился старый файл, — например, в папку *css* в корневом каталоге сайта.

2. **Замените старую папку *images* jQuery UI новой папкой *images*.**

   В разных темах используются различные цветные значки, а также различные стили фоновых узоров. Значки и графика новой темы, вероятно, не будут соответствовать вашей старой теме, поэтому вам придется заменить старые изображения. Папка *images* jQuery UI должна находиться там, куда вы поместили файл или файлы CSS jQuery UI, например, в папке *css* в корневом каталоге сайта или в папке *jquery-ui*. Вне зависимости от того, в какую папку вы поместили CSS-файл jQuery, поместите туда же папку с изображениями.

ПРИМЕЧАНИЕ

При загрузке новой темы убедитесь в том, что версия jQuery UI не изменилась. Плагин jQuery UI часто меняется, поэтому если вы решили обновить вашу тему, имейте в виду то, что файл JavaScript jQuery UI мог измениться. В этом случае вам нужно заменить изображения, файлы CSS, а также файл JavaScript jQuery UI (*jquery-ui.min.js*) на своем сайте.

Подробнее о CSS-файлах jQuery UI

При загрузке плагина jQuery UI вы получаете шесть CSS-файлов, которые на самом деле представляют собой две группы, включающие по три файла: одна группа — это уменьшенный или сжатый вариант (например, *jquery-ui.min.css*), а другая содержит простые для чтения CSS-файлы (например, *jquery-ui.css*). Как упоминалось ранее, на веб-сервер лучше помещать уменьшенную версию, поскольку она быстрее загружается. Другие файлы можно открыть в текстовом редакторе, чтобы разобраться в принципе работы CSS-кода jQuery UI.

Самый большой из трех файлов — *jquery-ui.css* — представляет собой сочетание двух других файлов (*jquery-ui.structure.css* и *jquery-ui.theme.css*). Команда jQuery UI разделила весь код на два отдельных файла, поскольку существуют определенные правила CSS, которые применяются ко всем виджетам независимо от используемого шрифта или цвета. Эти

правила обеспечивают «структуру» виджетов и взаимодействий плагина jQuery UI и включают такие свойства, как `z-index` диалогового окна или свойство `position` всплывающей подсказки. Эти правила находятся в файле *jquery-ui.structure.css*.

Однако другие стили меняются в зависимости от темы. Например, семейство шрифтов, используемое для виджетов, и цвет фона заголовка аккордеона специфичны для конкретной темы. Эти стили находятся в файле *query-ui-theme.css*. Вы можете открыть этот файл и увидеть специфические свойства, которые влияют на текст и фон.

Соответственно, вы должны добавить ссылку на файл *jquery-ui.min.css* на страницы вашего сайта, но использовать файл *jquery-ui-theme.css* в качестве способа изучения имен и свойств стилей различных виджетов jQuery UI.

Переопределение стилей jQuery UI

Темы jQuery UI хорошо выглядят, но не всегда могут вам подходить. Например, при использовании приложения ThemeRoller (см. раздел «Знакомство с приложением ThemeRoller» ранее в данной главе) для задавания семейства шрифтов у вас есть только один вариант для диалоговых окон, панелей выбора дат, панелей с вкладками и т. д. Но что если вам нужно использовать один шрифт для вкладки, а другой — для содержимого панели?

Существует несколько подходов к решению этой проблемы, однако есть одна вещь, которую вы не должны делать — вы не должны редактировать сами CSS-файлы плагина jQuery UI. CSS-стили jQuery изменяются при выходе новых версий jQuery UI, поэтому если вы измените файл *jquery-ui.css*, а затем вам потребуется обновить jQuery UI, то вам придется потратить время на внесение каждого сделанного в исходном CSS-файле изменения в новый файл *jquery-ui.css*.

Даже если вы не обновляете плагин jQuery UI, вы не сможете использовать приложение ThemeRoller для создания нового дизайна без необходимости скопировать внесенные вами изменения в новый файл темы.

Вместо этого вам следует создать новые стили jQuery UI в отдельном файле CSS. Это может быть отдельный файл CSS, предназначенный только для сделанных вами изменений jQuery UI, или основная таблица

стилей вашего сайта. Проще всего добавить новые стили в главную таблицу стилей вашего сайта. Как вы узнаете дальше, существует несколько методов переопределения стилей jQuery UI, однако все они опираются на использование концепции специфичности CSS. Вы узнаете об этом подробнее чуть позже, а пока имейте в виду то, что вам нужно привязать таблицу стилей, которую вы будете использовать для переопределения стилей jQuery UI, после загрузки таблицы стилей jQuery UI:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
```

Поместив таблицу стилей вашего сайта после таблицы стилей jQuery UI, вы сможете свободно создать стиль с точно таким же именем, что и в таблице стилей jQuery UI. Созданный вами стиль будет иметь приоритет, поскольку он создан после такого же стиля в таблице стилей jQuery UI.

Понятие специфичности

Стили CSS часто взаимодействуют и конфликтуют между собой. Вы можете применить к одному и тому же элементу страницы более одного стиля, и браузер будет определять, какие свойства и из каких стилей следует применять. Это каскадная часть языка CSS, и по-настоящему важным понятием является специфичность. В языке CSS более специфичное правило всегда получает приоритет. Далее приведены некоторые из самых важных правил специфичности:

- **Селекторы идентификатора являются более специфичными, чем классы, а классы являются более специфичными, чем селекторы элементов.** Например:

```
<p id="susan" class="person">Сьюзан Джонс: ↵
Суперзвезда CSS</p>
```

В данном HTML-коде элемент `p` имеет идентификатор и класс. Предположим, что у нас есть следующий CSS-код:

```
#susan {
color: green;
font-size: 24px;
}
.person {
```

```
color: blue;
text-align: center;
font-weight: bold;
}
p {
color: orange;
font-weight: normal;
font-family: Arial, sans-serif;
}
```

Все три стиля применены к абзацу, и все три имеют свойство `color`. Однако поскольку селекторы идентификатора являются более специфичными, для этого абзаца будет использоваться зеленый цвет, так как селектор идентификатора переопределяет два других стиля.

Тем не менее при отсутствии конфликта, как в случае со свойством `text-align` в стиле класса `.person`, это свойство применяется независимо от того, к какому стилю оно принадлежит.

В данном примере текст абзаца зеленый размером 24px (селектор `#susan`), имеет полужирное начертание и выравнивание по центру (селектор `.person`), а также использует шрифт Arial (селектор `p`).

Поскольку плагин jQuery UI использует для форматирования стили класса, важно знать, как специфичность работает с различными типами селекторов.

- **Правила, которые определены позднее, получают приоритет**. Если в двух разных таблицах стилей присутствует одно и то же имя правила (или даже в одной и той же таблице стилей), то приоритет получает то, которое определено последним. Именно поэтому вы должны сначала привязать таблицу стилей jQuery UI. Вы можете добавить новые стили с тем же именем, что и классы jQuery UI во второй таблице стилей и переопределить их из jQuery UI. Например, в таблице стилей jQuery UI вы найдете стиль вроде этого:

  ```
  /* в таблице стилей jquery-ui */
  .ui-widget-content a {
  color: #222222;
  }
  ```

Если вы привязываете таблицу стилей своего сайта после таблицы стилей jQuery UI, вы можете добавить этот стиль в таблицу стилей для сайта и полностью переопределить стиль, применяемый к этому элементу плагином jQuery UI:

```
/* в таблице стилей сайта*/
.ui-widget-content a {
color: #FF0;
}
```

- **Специфичность суммируется**. Использование селекторов потомков является мощным средством для создания очень специфических стилей. Браузер суммирует все селекторы, перечисленные в селекторе потомка для создания общего правила. Простой способ расчета этой суммы заключается в том, чтобы принять селекторы идентификатора за 100 баллов, селекторы класса за 10 баллов, а селекторы элемента за 1 балл. Например, стиль с селектором `#main a` переопределит стиль `.ui-state-default a`, поскольку стоимость первого равна 101 баллу (один идентификатор и один элемент), а второго — всего 11 баллам (один класс и один элемент). Другими словами, вы можете переопределить правило jQuery UI с помощью селектора потомков, который обеспечивает большее количество баллов.

- **При возникновении сложностей используйте правило `!important`**. В языке CSS предусмотрено правило `!important`. Если добавить это правило после значения в декларации CSS, то это значение переопределит любой другой стиль вне зависимости от его специфики. Например, в коде CSS, приведенном в первом пункте данного списка, стиль `p` задан следующим образом:

```
p {
color: orange !important;
font-weight: normal !important;
font-family: Arial, sans-serif !important;
}
```

Поскольку каждое из свойств в этом стиле имеет правило `!important`, то эти свойства переопределяют значения в более специфических правилах `#susan` и `.person`. Правило `!important` следует использовать только при крайней необходимости. Если вы станете использовать его

очень часто, то в конечном итоге ваши стили окажутся наполнены правилами `!important`, и вам будет трудно разобраться в их специфичности. (Если каждое свойство в каждом стиле имеет правило `!important`, то станут применяться обычные правила специфичности, то есть селекторы идентификатора являются более специфичными, чем классы, а классы являются более специфичными, чем селекторы элементов.)

Как плагин jQuery UI стилизует виджеты

Плагин jQuery UI использует модульный подход к стилизации виджетов: вместо создания одного стиля `.ui-dialog` со всеми настройками диалоговых окон jQuery, внешний вид виджета создается из нескольких стилей класса. Например, существует стиль `.ui-widget`, который применяется ко всем виджетам. Он содержит несколько очень простых стилей:

```
.ui-widget {
font-family: Verdana,Arial,sans-serif;
font-size: 1.1em;
}
```

Поскольку этот класс применяется ко всем виджетам, все они имеют один и тот же базовый шрифт и размер шрифта. Однако к виджету Dialog jQuery также применяет множество других классов. Например, в случае с диалоговым окном к внешнему контейнеру применяются следующие классы:

```
class="ui-dialog ui-widget ui-widget-content ↵
ui-corner-all ui-front ui-draggable ui-resizable"
```

Каждый из этих классов отвечает за один небольшой аспект общего внешнего вида диалогового окна. Объединяя таким способом несколько классов, плагин jQuery UI имеет возможность повторно использовать их для создания внешнего вида различных виджетов. Например, далее приведены классы, примененные к внешнему контейнеру меню jQuery UI (см. раздел «Добавление меню на страницу» главы 9):

```
class="ui-menu ui-widget ui-widget-content"
```

Обратите внимание на то, что два класса — `ui-widget` и `ui-widget-content` — также применяются к диалоговому окну. Эти

два стиля содержат некоторое базовое форматирование, применяемое к этим двум виджетам.

Как вы видели, простой элемент `div` можно очень легко превратить в сложное диалоговое окно (см. раздел «Создание диалоговых окон с сообщениями» главы 9), а вложенный список — в раскрывающееся меню (см. раздел «Добавление меню на страницу» главы 9). За кулисами плагин jQuery UI берет ваш простой HTML-код и добавляет слои `div`, `span` и других HTML-элементов, чтобы создать итоговый внешний вид. В результате вы не можете определить, как происходит стилизация, глядя на HTML-код или даже на таблицы стилей jQuery UI. Вам приходится разбираться в визуализированном HTML-коде (то есть в HTML-коде, к которому применена магия jQuery UI), чтобы понять, как плагин jQuery UI форматирует свои виджеты.

Лучший способ проанализировать то, как jQuery UI применяет классы к виджетам, — это использовать встроенные инструменты вашего браузера. Большинство браузеров предусматривает возможность просмотра визуализированного HTML-кода. В программе Chrome, например, вы можете щелкнуть правой кнопкой мыши по виджету jQuery UI и выбрать в контекстном меню пункт **Inspect Element** (Просмотр кода элемента). В нижней части окна браузера откроется консоль, в которой будет отображен визуализированный HTML-код на вкладке **Elements** (Элементы) (рис. 11.4).

Щелчок по значку в виде лупы (обведен на рис. 11.4) активирует инструмент инспектирования. После активации этого инструмента вы можете щелкнуть по любому элементу на странице, в результате чего соответствующий HTML-код будет выделен во вкладке **Elements** (Элементы). Это отличный способ идентифицировать заголовок диалогового окна, чтобы увидеть, какие классы к нему применены.

После определения классов вы можете создавать стили с тем же именем в своей собственной таблице стилей. Например, инспектор браузера Chrome показывает, что HTML-код, используемый для создания заголовка диалогового окна, включает класс `.ui-dialog-title` (рис. 11.5). Чтобы сделать заголовок диалоговых окон красным, вы можете добавить стиль в таблицу стилей вашего сайта следующим образом:

```
.ui-dialog-title {
color: red;
}
```

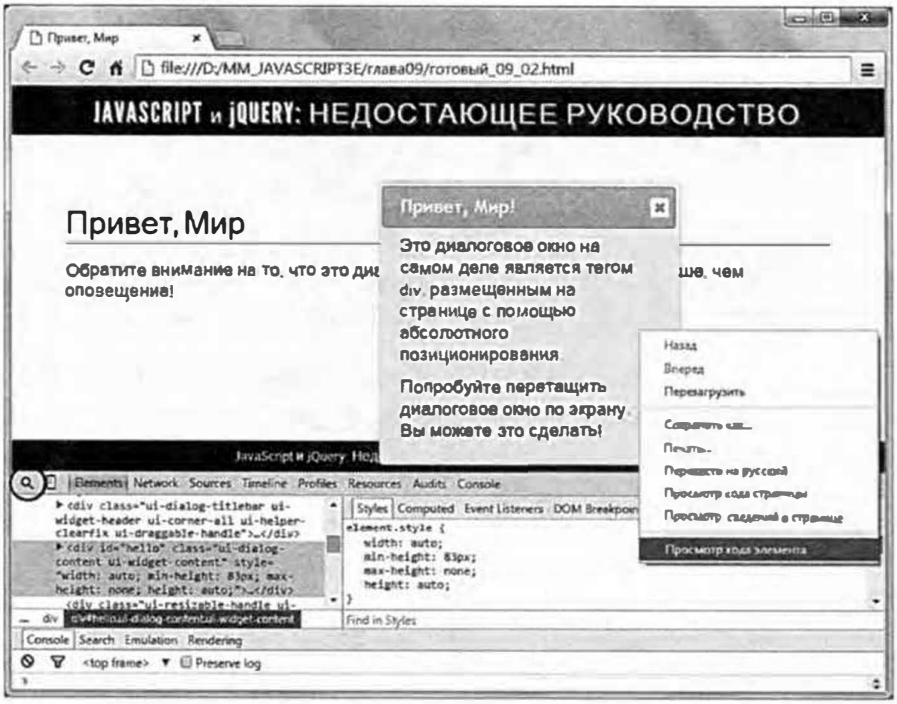

Рис. 11.4. После открытия вкладки **Elements** (Элементы) в консоли вы можете навести указатель мыши на интересующий вас HTML-элемент. Элементы в окне браузера, которые соответствуют HTML-коду, выделятся при наведении на них указателя мыши. Вы можете сопоставить HTML-код с его визуальным представлением в браузере

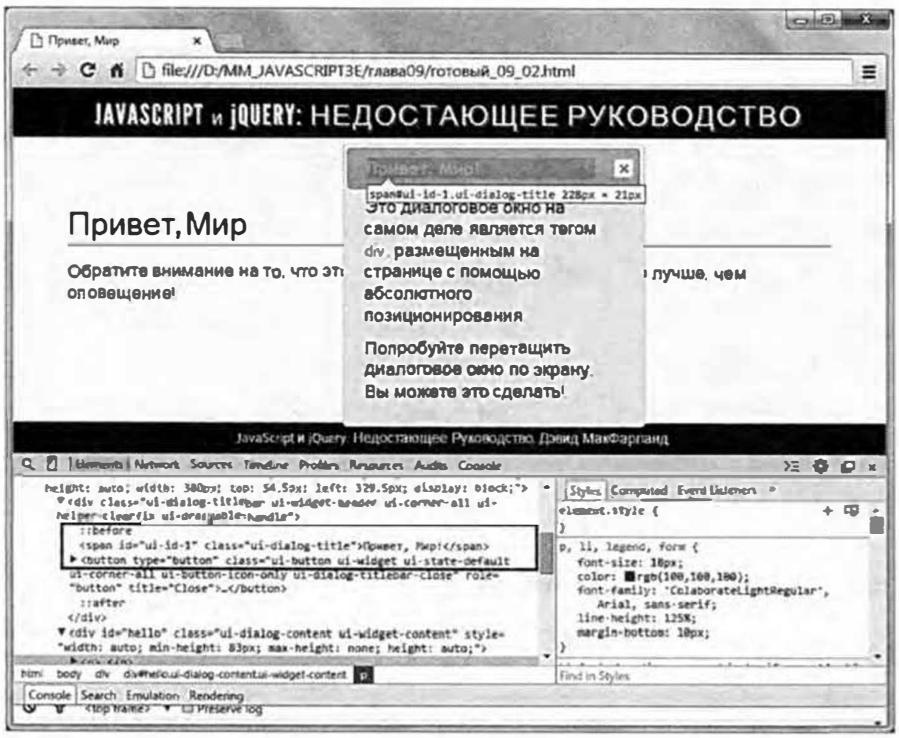

Рис. 11.5. Используя инструмент **Inspector** (Инспектор) (значок в виде увеличительного стекла), вы можете быстро определить визуализированный HTML-код, относящийся к элементам в окне браузера. На данном изображении представлен HTML-код, производимый плагином jQuery UI для создания заголовка диалогового окна

Глава 12
ВЗАИМОДЕЙСТВИЯ И ЭФФЕКТЫ JQUERY UI

Плагин jQuery UI может предложить не только виджеты для пользовательского интерфейса. Он также предусматривает множество полезных функций, обеспечивающих интерактивность веб-страниц и веб-приложений. Эти взаимодействия jQuery UI предоставляют инструменты, которые позволяют посетителям передвигать элементы на странице. Например, некоторые виджеты можно перетаскивать на другое место. Это означает, что посетители могут перетащить элемент и поместить его на другой, например, перетащить файл в мусорную корзину или перетащить продукт в корзину для покупок. Вы также можете составлять списки сортируемых элементов, которые пользователь может переупорядочить, просто перетащив элемент на новую позицию в списке, например, в списке дел или списке воспроизведения.

Кроме того, плагин jQuery UI включает набор эффектов, которые можно использовать для добавления на ваш сайт визуальной составляющей с высококачественной анимацией. Например, вы можете привлечь внимание посетителя, заставив цвет фона элемента мигать, элемент страницы отскакивать, дрожать, увеличиваться, уменьшаться или взрываться. Подробнее об эффектах вы узнаете в разделе «Эффекты jQuery UI» этой главы.

Виджет Draggable

Перетаскивание является очень распространенным действием при работе на компьютере. Вам часто приходится перетаскивать файлы в папки или в мусорную корзину, а также перетаскивать окна, чтобы освободить место для других окон, и т. д. Эти взаимодействия естественны для поколения, выросшего с персональным компьютером и мышью. Плагин jQuery UI обеспечивает этот же тип взаимодействия с веб-страницами с помощью виджета Draggable и его компаньона — виджета Droppable (обсуждается в разделе «Виджет Droppable» далее в главе).

Виджет Draggable позволяет сделать любой элемент страницы подвижным. Вы уже видели его в действии в примере с виджетом jQuery UI Dialog (см. раздел «Создание диалоговых окон с сообщениями» главы 9): диалоговые окна jQuery UI появляются на экране, но их можно свободно перетаскивать с места на место. Вы можете использовать виджет Draggable в сочетании с виджетом Droppable для создания интерактивной корзины покупок. Покупатели будут перетаскивать изображения товаров в корзину на странице, и эти предметы станут добавляться в нее.

Добавление виджета Draggable на веб-страницу

Вы можете превратить в перетаскиваемый элемент любую часть страницы. Конечно, перемещение того или иного элемента должно иметь смысл. Например, обеспечение подвижности абзацев текста не сделает страницу более читабельной и не пойдет на пользу вашим посетителям. Однако обеспечение подвижности всплывающих диалоговых окон имеет смысл, а наличие подвижных частей является обязательным условием для любой онлайн-игры в шашки.

Начать работать с виджетом Draggable легко:

1. **Следуйте инструкциям, приведенным в разделе «Добавление плагина jQuery UI на веб-страницу» главы, чтобы прикрепить файлы CSS и JavaScript.**

   Следует помнить, что у плагина jQuery UI есть свои собственные файлы CSS и JavaScript, и что ссылка на JavaScript-файл jQuery UI должна следовать после ссылки на JavaScript-файл библиотеки jQuery.

2. **Добавьте функцию `$(document).ready()` на свою страницу или во внешний JavaScript-файл:**

   ```
   $(document).ready(function() {

   }); // окончание ready
   ```

   Как уже говорилось в разделе «Больше концепций для событий jQuery» главы 5, этот шаг необходим только в том случае, если вы помещаете свой JavaScript-код в раздел заголовка страницы перед большей частью кода HTML. Некоторые программисты помещают свой код JavaScript в нижней части страницы перед закрывающим тегом `</body>`. В этом случае вы можете пропустить функцию `$(document).ready()`.

3. **Используйте jQuery, чтобы выбрать элемент страницы и применить к нему виджет Draggable:**

```
$(document).ready(function() {
$('#dialog').draggable();
}); // окончание ready
```

В данном примере вы выбираете один элемент, используя селектор идентификатора. Тем не менее если вы хотите сделать подвижными несколько элементов страницы, вы можете применить класс к каждому из их HTML-элементов, например, к `<div class="draggable">`, а затем использовать селектор класса, чтобы применить функцию `draggable()` ко всем этим элементам сразу:

```
$('.draggable').draggable();
```

4. **Сохраните файлы и просмотрите их в браузере.**

Вот и все, что нужно сделать. Тем не менее вы, вероятно, решите настроить поведение данного виджета. К счастью, плагин jQuery UI предусматривает для этого множество различных параметров. Вы узнаете о них в разделе «Параметры виджета Draggable» далее в главе.

Применение виджета Draggable на практике

Пришло время опробовать этот виджет и посмотреть, насколько легко можно сделать любой элемент на странице подвижным.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *12_01.html* в папке *глава 12*.**

Этот файл уже содержит ссылку на необходимые файлы jQuery UI и jQuery, а также функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5).

Просмотрите код HTML и найдите элемент `div` с идентификатором `note` и небольшим объемом содержимого. Вы превратите его в перетаскиваемый элемент.

2. **В пустой строке внутри функции `$(document).ready()` добавьте строку, выделенную полужирным шрифтом:**

```
$(document).ready(function() {
$('#note').draggable();
}); // окончание ready
```

Этот код выбирает элемент `div` и делает его подвижным.

3. **Сохраните страницу и просмотрите ее в браузере (рис. 12.1).**

Щелкните в любом месте элемента `div` с заголовком «Ты можешь перетащить меня» и переместите его по экрану. Существует несколько способов настройки этого виджета, которые мы рассмотрим далее.

ПРИМЕЧАНИЕ

Вы найдете готовую версию этого руководства в файле *готовый_12_01.html*.

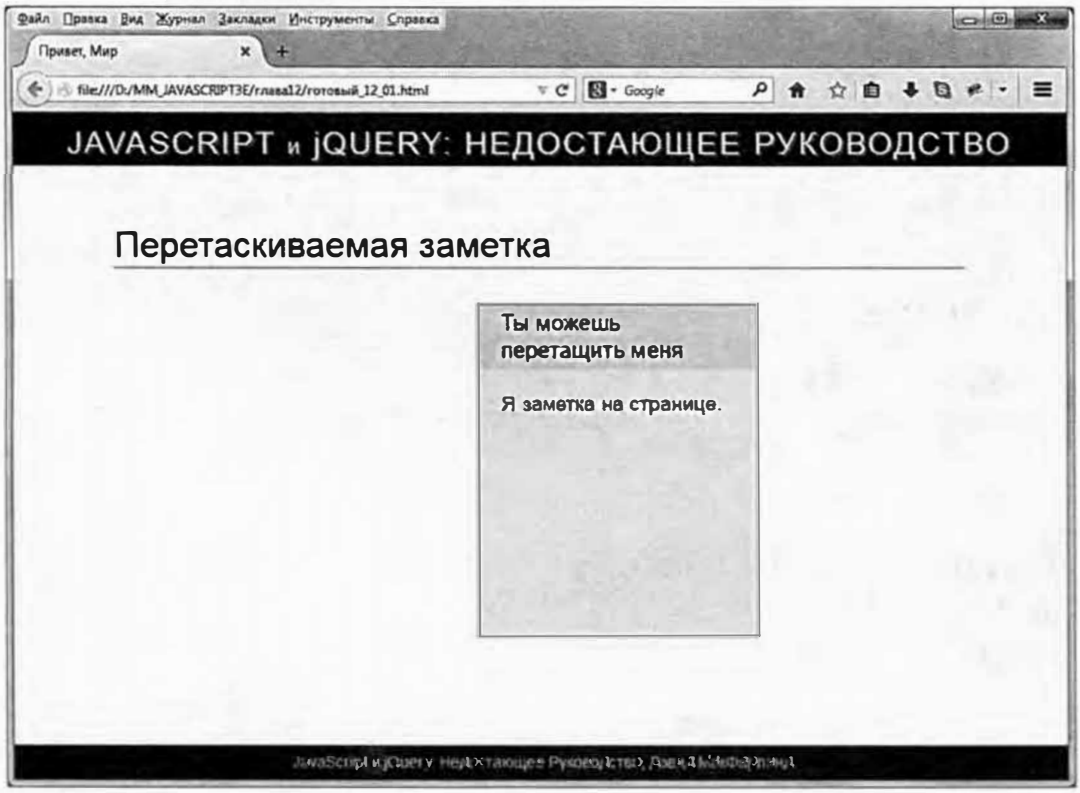

Рис. 12.1. С помощью лишь одной строки кода вы можете превратить статичный элемент `div` в перетаскиваемое по странице окно. Виджет Draggable также используется виджетом jQuery UI Dialog (см. раздел «Создание диалоговых окон с сообщениями» главы 9)

Параметры виджета Draggable

jQuery UI Draggable — это очень мощный виджет, предусматривающий множество параметров. Вы можете контролировать направление, расстояние, поведение элемента при перетаскивании, а также определить часть, за которую вы можете перетаскивать этот элемент. Как и в случае с другими виджетами jQuery UI, вы можете задать параметры, передав функции `draggable()` объект. Например, чтобы изменить вид курсора на указательный палец и сделать элемент `h2` частью, за которую посетитель может перетащить элемент, вам следует передать объект со следующими двумя параметрами:

```
$('#controls').draggable({
cursor : 'pointer',
handle : 'h2'
});
```

Далее перечислены наиболее распространенные параметры:

- **`axis`**. С помощью данного свойства вы можете ограничить перемещение перетаскиваемого элемента только направлениями влево и вправо или вверх и вниз. Например, вам необходимо перетащить элемент вдоль временной шкалы. Вам необходимо, чтобы перетаскиваемый элемент оставался расположенным вдоль этой шкалы и не смещался с нее. Используйте значение `'x'` (чтобы элемент передвигался только влево и вправо) или `'y'` (чтобы элемент мог быть перемещен только вверх и вниз по странице):

  ```
  axis : 'x'
  ```

- **`cancel`**. Этот параметр позволяет предотвратить перетаскивание при щелчке по конкретному элементу. Например, вы создали окно с объявлением, которое посетители могут перетаскивать по экрану. Это окно имеет заголовок и несколько абзацев текста, содержащих такую важную информацию, как адрес и номер телефона. Обычно, если посетитель пытается выбрать почтовый адрес (например, чтобы скопировать и вставить в строку сервиса Google Maps), то вместо этого он в конечном итоге перетаскивает объявление в окне. Вы можете предотвратить перетаскивание этих абзацев с помощью параметра `cancel`:

  ```
  cancel : 'p'
  ```

  Теперь, если кто-то щелкнет по абзацу внутри объявления, он не будет инициировать перетаскивание и сможет легко выбрать и ско-

пировать текст. Параметр `handle`, который обсуждается далее, также позволяет посетителям выбрать содержимое перетаскиваемого элемента.

- **`connectToSortable`**. Данный параметр позволяет вам указать набор сортируемых элементов, к которым можно добавить перетаскиваемый элемент. Плагин jQuery UI предусматривает взаимодействие `sortable`, позволяющее вам сгруппировать элементы, которые можно сортировать, например, список дел (вы узнаете о сортируемых элементах в разделе «Сортировка элементов страницы» далее в главе). Чтобы использовать этот параметр, просто укажите идентификатор для элемента, который вы уже превратили в сортируемую группу (см. раздел «Сортировка элементов страницы» далее в главе):

```
connectToSortable : '#toDoList'
```

- **`containment`**. Вы можете предотвратить возможность перетаскивания элемента за пределы контейнера. Например, вы создаете действие «Магнит на холодильник», позволяющее посетителю перетаскивать слова, чтобы составить предложение. Вы можете добавить множество небольших перемещаемых элементов `div`, содержащих слова, и определить область страницы, где посетитель может построить фразу. Вам нужно убедиться, что он не сможет перетащить слова, например, на навигационную панель, заголовок или нижний колонтитул. Параметр `containment` позволяет ограничить взаимодействие небольшой областью страницы. Данный параметр может принимать несколько значений:

- **Селектор**. Если вы укажите имя селектора, то плагин jQuery UI будет удерживать перетаскиваемый объект в границах этого элемента. Например, если на вашей странице есть элемент `div` с идентификатором `refrigerator`, то вы можете удерживать перетаскиваемый элемент внутри этого элемента `div`, используя следующий код:

```
containment : '#refrigerator'
```

- **`parent`, `document` или `window`**. Чтобы удержать перетаскиваемый элемент внутри его родительского элемента, используйте в качестве значения фрагмент `parent`. Например, если перетаскиваемым элементом является `div` внутри другого элемента `div`, и вы хотите ограничить перемещение областью этого внешнего элемента `div`, используйте значение `parent` следующим образом:

```
containment : 'parent'
```

Значения `document` и `window` практически одинаковы за исключением того, что значение `document` ограничивает область перетаскивания площадью документа, а значение `window` позволяет перетаскивать элемент частично за пределы окна.

- **Массив координат**. Наконец вы можете предоставить массив координат, которые соответствуют левой верхней и правой нижней точкам контейнера. Вы предоставляете значения в пикселах относительно верхнего левого угла окна браузера. Например, вам нужно, чтобы перетаскивание происходило только в области, которая начинается в точке, находящейся в 50 пикселах от левого края и 100 пикселах от верхнего края и заканчивается в точке, находящейся в 500 пикселах от левого края и 600 пикселах от верхнего края. В этом случае вы зададите эти значения следующим образом:

  ```
  containment : [50, 100, 500, 600]
  ```

  Поскольку такое использование координат требует передачи точных значений, это не очень хорошо подходит при создании резинового дизайна. В этом случае вам лучше использовать элемент-контейнер и указать селектор.

- **`cursor`**. Вы можете обеспечить изменение внешнего вида указателя мыши при перетаскивании. Обычно он имеет вид стрелки (если вы щелкаете по пустому месту элемента) или текстового курсора (если вы щелкнули по тексту в заголовке). Тем не менее вы можете приказать плагину jQuery UI использовать другой вид курсора при перетаскивании элемента, передав параметр `cursor` с действительным значением CSS. Вы можете найти список значений на странице: **developer.mozilla.org/en-US/docs/Web/CSS/cursor**, однако чаще всего при перетаскивании используются такие значения, как `pointer`, `crosshair` и `default` (стрелка):

  ```
  cursor : 'pointer'
  ```

- **`cursorAt`**. Вы можете контролировать положение указателя мыши при перетаскивании элемента. Обычно он остается в том месте, на которое посетитель изначально его навел. Например, если вы захватите перетаскиваемый элемент `div` за его нижнюю часть, то указатель мыши останется в нижней части этого элемента `div` в процессе его перемещения по экрану. Тем не менее вы можете захотеть, чтобы курсор находился в определенном месте, например, на изображении маркера, который вы добавили к перетаскиваемому элементу. В этом случае вам следует предоставить параметру `cursorAt` объект. Этот

объект может содержать значения, относящиеся к свойствам `left`, `right`, `top` и `bottom`. Например, для того чтобы при перемещении элемента указатель мыши находился в верхнем левом углу элемента, укажите свойства `left` и `top` следующим образом:

```
cursorAt : {
left : 5,
top : 5
}
```

Вы не можете одновременно установить свойства `left` и `right`, поскольку они соответствуют разным положениям на одной и той же оси. По этой же причине вы не можете одновременно использовать свойства `top` и `bottom`. Тем не менее вы можете использовать по одному значению на каждой оси (`left` или `right` и `top` или `bottom`). Аналогично, при желании вы можете установить только одно значение. Например, если вам важно, чтобы указатель мыши находился внутри полосы в верхней части перетаскиваемого элемента, вы можете использовать только значение `top`. Горизонтальное значение будет соответствовать горизонтальному положению, в которое посетитель изначально поместил указатель мыши:

```
cursorAt : { top : 5 }
```

- **`disabled`**. Определяет, является ли элемент перетаскиваемым. Если значением этого свойства является `true`, то вы не сможете перетащить элемент. Вы, вероятно, думаете, что для предотвращения возможности перетаскивания элемента вам достаточно просто не использовать функцию `draggable()`. Это правда, вы, вероятно, не будете использовать параметр `disabled`, когда вы впервые применяете к элементу виджет Draggable.

  Тем не менее этот параметр может пригодиться после того, как кто-нибудь притащит элемент. Например, вы можете позволить посетителю перетащить элемент в корзину покупок, но не из нее. В этом случае вы можете применить данный параметр после завершения операции перетаскивания (вы рассмотрите данный пример в разделе «Событие `drop`» далее в этой главе).

- **`grid`**. Обычно при перетаскивании элемента он свободно перемещается в окне. Тем не менее вы можете захотеть, чтобы элемент перемещался на определенное количество шагов по горизонтали и вертикали. Например, представьте онлайн-игру в шашки. У вас есть сетка

площадью 8 × 8 клеток. Шашки можно перетаскивать в границах этой сетки, однако, если бы их можно было перетащить в любое место, это выглядело бы неаккуратно. Вам нужно, чтобы шашки передвигались по клеткам на доске.

Плагин jQuery UI позволяет обеспечить перемещение элемента по горизонтали и вертикали на конкретное расстояние. Параметр `grid` принимает массив, состоящий из двух чисел. Первое число соответствует расстоянию по горизонтали, а второе — по вертикали. Например, вам нужно, чтобы при перетаскивании влево или вправо элемент перемещался на расстояние 50 пикселов, а при перетаскивании вверх или вниз — на расстояние 100 пикселов. В этом случае вы можете настроить параметр `grid` следующим образом:

```
grid : [50, 100]
```

- **`handle`**. Вы можете ограничить область, по которой может щелкнуть посетитель для перетаскивания элемента. Как правило, вы можете щелкнуть в любом месте элемента, чтобы перетащить его. Тем не менее вы можете ограничить эту область только заголовком или визуальным индикатором перемещения. Используйте параметр `handle`, чтобы определить селектор в перетаскиваемом элементе, который будет выступать в качестве маркера перетаскивания. Например, вы можете сделать так, чтобы перетащить элемент можно было бы, щелкнув по его заголовку, например, по элементу `h2`:

```
handle : 'h2'
```

- **`helper`**. Возможно, вы не хотите, чтобы пользователь перемещал по странице сам перетаскиваемый элемент. Например, вы создали страницу с покупательской корзиной. На этой странице отображаются изображения товаров, и посетители могут добавить эти товары в корзину, перетаскивая соответствующие изображения на значок в виде корзины покупок. В этом случае, вы не хотите, чтобы покупатель перетаскивал в корзину сами изображения товаров, поскольку это привело бы к перемещению изображения по странице. Вместо этого вам нужно, чтобы покупатель просто перетащил копию изображения, не затрагивая исходное изображение. В этом случае вы можете указать для параметра `helper` значение `'clone'`:

```
helper : 'clone'
```

Более сложный вариант заключается в передаче параметра `helper` функции, генерирующей HTML-код, который плагин jQuery UI

должен отобразить при перетаскивании мыши. Например, для отображения пользовательского вспомогательного элемента `div` (вместо перетаскивания фактического HTML-элемента) вы можете задать параметр `helper` следующим образом:

```
helper: function( event ) {
return $( "<div class='ui-widget-header'> ↵
Я пользовательский вспомогательный элемент</div>" );
}
```

То, что вы предоставите в качестве возвращаемого значения, плагин jQuery UI отобразит под указателем мыши пользователя. Обратите внимание на то, что вы должны возвратить объект jQuery, поэтому для создания помощника вы должны использовать функцию `$()`. Простая строка HTML-кода не сработает.

- **`opacity`**. При перетаскивании элемента вы можете изменить степень его прозрачности. Например, вы можете задать значение 50%, чтобы придать элементу призрачный вид. Эта техника является популярным способом показать то, что вы перемещаете элемент с места на место. Используйте значение от 0 (невидимый) до 1 (полностью непрозрачный). Этот параметр работает так же, как свойство CSS `opacity`. Например, чтобы сделать перетаскиваемый элемент на 50% прозрачным, укажите значение `0.5`:

  ```
  opacity : 0.5
  ```

- **`revert`**. Этот параметр определяет, возвращается ли перетаскиваемый элемент в исходное положение, после того как пользователь прекращает его перемещение. Как правило, для этого параметра устанавливается значение `false`, а это значит, что после того как вы перетащили элемент, он остается там, где вы его оставили. Это обычно хорошо для перетаскиваемых или диалоговых окон, поскольку позволяет перетащить окно и освободить место на странице для чего-то другого.

  Тем не менее иногда вам требуется вернуть перетаскиваемый элемент в исходное положение. Например, если вы используете виджет Droppable (см. раздел «Виджет Droppable» далее в главе), то вы, возможно, захотите, чтобы перемещаемый элемент вернулся в исходное положение, если он не попал в определенное место. Например, вы создали приложение для организации изображений. Вы можете перетащить эскизы изображений в разные папки на странице или на значок

в виде мусорной корзины. Тем не менее пользователь может перетащить изображение вверх на заголовок страницы или в нижний колонтитул, а не в одну из предназначенных папок или в корзину. В этом случае вам нужно, чтобы изображение вернулось в исходное место на странице, а не загораживало заголовок или нижний колонтитул.

Параметр `revert` принимает несколько различных значений: значение `true` заставляет перетаскиваемый элемент всегда возвращаться в исходное положение (что отлично подходит для сайта с приколами). Значение `'invalid'` заставляет перетаскиваемый элемент возвращаться в исходное положение только в том случае, если он не был помещен на допустимый бросаемый элемент (см. раздел «Виджет Droppable» далее в главе). В случае с упомянутым выше приложением для организации изображений вы бы использовали параметр `invalid` следующим образом:

```
revert : 'invalid'
```

Наконец вы можете передать этому параметру функцию. Если эта функция возвратит значение `true`, то элемент вернется в исходное положение. Вы можете использовать функцию в основанной на JavaScript игре в шашки, например: если игрок перемещает фигуру и бросает ее на другую клетку доски, функция может проверить, был ли этот ход действительным (например, не занята ли клетка другой фигурой и не передвинута ли она дальше, чем это допустимо). В этом случае функция возвратит значение `true`, и плагин jQuery UI переместит фигуру в исходное положение.

```
revert : function() {
// выполнить тестирование в этой функции и
// возвратить значение true, чтобы вернуть
// элемент в исходное положение или false,
// если элемент может остаться там, где
// пользователь его оставил
}
```

- **`revertDuration`**. Когда плагин jQuery UI возвращает бросаемый элемент (перемещает его в исходное положение), он анимирует его передвижение. По умолчанию применяется значение 500 миллисекунд или полсекунды. Это выглядит здорово, но вы можете ускорить анимацию (для этого следует использовать число меньше 500) или замедлить ее (в этом случае число должно быть больше 500). Для этого

используйте параметр `revertDuration`, который принимает числовое значение, соответствующее количеству миллисекунд, определяющему длительность анимации. Например, если вы хотите, чтобы длительность анимации составляла четверть секунды (250 миллисекунд), то вы можете установить значение параметра `revertDuration` так:

```
revertDuration : 250
```

- **scope**. Параметр `scope` позволяет сгруппировать перетаскиваемые и бросаемые элементы в наборы. Например, на вашей странице есть календарь. Пользователи могут перетаскивать событие с одного дня в календаре на другой. Все перетаскиваемые события принадлежат календарю, поэтому вы можете указать для данного параметра значение `calendar`:

  ```
  scope : 'calendar'
  ```

  Строка `'calendar'` в данном случае не относится к селектору или элементу на странице. Это просто имя, используемое для группировки перетаскиваемых и бросаемых элементов. Вы можете использовать любой термин, просто убедитесь в том, что вы используете то же имя для параметра `scope`, примененного к бросаемому элементу (см. раздел «Виджет Droppable» далее в главе).

  Вам нужно устанавливать параметр `scope` только в том случае, если на странице присутствуют разные типы перетаскиваемых и бросаемых элементов. Например, если на странице с календарем у вас есть еще и мозаика, то вы не захотите, чтобы пользователи перетаскивали события из календаря на мозаику, а кусочки мозаики — на календарь. Указав различные параметры `scope` для каждой группы, вы можете контролировать, какие перетаскиваемые элементы могут быть брошены на другие элементы (см. раздел «Виджет Droppable» далее в главе для получения дополнительной информации о виджете Droppable).

ПРИМЕЧАНИЕ

Вы можете увидеть эти параметры в действии в файле *12_01.html*, над которым вы работали в предыдущем мини-руководстве. Если вы не доделали его, откройте файл *готовый_12_01.html* и добавьте некоторые из этих параметров в функцию `draggable()`.

- **snap**. Вы можете сделать так, чтобы перетаскиваемый элемент привязался к другому элементу страницы или к другому перетаскивае-

мому элементу. Например, вы создали игру, в которой необходимо составить исходную фотографию из дюжины фрагментов. Вы можете сделать так, чтобы один фрагмент головоломки привязывался к другому фрагменту, используя данный параметр, который может принимать одно из двух возможных значений:

- Значение `true` сообщает плагину jQuery UI о том, что перетаскиваемый элемент должен привязаться к другому перетаскиваемому элементу на странице. Это хорошо работает в случае с мозаикой, состоящей из множества отдельных фрагментов.
- Имя селектора. Имя любого селектора на странице. Например, если вы хотите, чтобы перетаскиваемый элемент привязывался к элементу `div` с идентификатором `photoholder`, вы можете использовать следующий код:

  ```
  snap : '#photoholder'
  ```

  Способ привязки элемента к другому элементу контролируется параметрами `snapMode` и `snapTolerance`, которые обсуждаются далее.

- **`snapMode`**. Этот параметр работает только при использовании параметра `snap` и принимает одно из трех ключевых слов: `inner`, `outer` или `both`. Если вы хотите, чтобы перетаскиваемый элемент привязывался только тогда, когда он находится внутри элемента, используйте ключевое слово `inner`, которое заставляет перетаскиваемый элемент привязываться к любому из внутренних краев элемента, предусмотренного параметром `snap`.

  Используйте ключевое слово `outer`, если вы хотите, чтобы элемент привязывался к внешнему краю другого элемента. Этот параметр идеально подходит для таких вещей, как мозаика, фрагменты которой должны привязываться друг к другу.

  Наконец вы можете обеспечить привязку элемента к внешнему или внутреннему краю другого элемента, используя значение `both`:

  ```
  snapMode : 'both'
  ```

- **`snapTolerance`** определяет, насколько близко должен располагаться перетаскиваемый элемент, чтобы произошла привязка. Чем больше значение данного параметра, тем дальше может быть расположен элемент, прежде чем он попытается привязаться к другому элементу. Для данного параметра следует указывать значение в пикселах:

```
snapTolerance : 30
```

- **zIndex**. Данный параметр позволяет задать значение `zIndex` для перетаскиваемого элемента. Свойство CSS `zIndex` определяет порядок размещения элементов на странице. Элемент с более высоким значением параметра `zIndex` будет находиться поверх всех элементов страницы, которые он перекрывает. Это свойство очень удобно, когда вы хотите удостовериться в том, что при перемещении элемента по странице он всегда будет располагаться поверх остального содержимого. (Пример использования этого параметра вы найдете в разделе «Drag-and-Drop на практике» далее в главе.)

```
zIndex : 100
```

ПРИМЕЧАНИЕ

Вы можете найти множество дополнительных параметров для виджета Draggable на странице **api.jqueryui.com/draggable/**. См. рис. 12.2.

События виджета Draggable

Виджет Draggable поддерживает несколько различных событий, каждое из которых запускается в разные моменты в процессе перетаскивания. Вы добавляете к событию функцию, чтобы заставить вашу программу выполнить некоторое действие в ответ на перетаскивание элемента.

Допустим, вы создали игру, которая требует от игрока быстро перетащить элемент на цель. Чем быстрее он может перетащить этот элемент, тем больше очков он набирает. Вы можете заметить момент начала и окончания процесса перетаскивания элемента.

Различным этапам процесса перетаскивания соответствуют четыре события.

Событие `create`

Это событие запускается каждый раз, когда вы используете функцию `draggable()` для создания нового перетаскиваемого элемента. Вы можете использовать это событие для вызова всплывающего диалогового окна с инструкцией «Перетащите этот товар в свою корзину». Это событие срабатывает только один раз при создании виджета Draggable.

Вы можете использовать это событие, передав свойство `create` с функцией методу `draggable()`:

```
$('.product').draggable({
create : function (event){
// здесь располагается программный код
}
});
```

Рис. 12.2. Документация по работе плагина jQuery UI обширна. На странице, посвященной виджету Draggable, перечислены многочисленные параметры, методы и события, которые вы можете задействовать для настройки виджета

Событие `start`

Данное событие срабатывает, как только пользователь начинает перетаскивать элемент. Вы назначаете это событие в качестве свойства объекта, содержащего параметры виджета Draggable. Имя события, `start`,

представляет собой имя свойства, а значением должна быть функция, которая выполняется, когда посетитель начинает перетаскивать элемент.

Например, у вас есть элемент `div` с идентификатором `raceCar`. Вы можете превратить этот элемент в перетаскиваемый и назначить ему событие `start` следующим образом:

```
$('#raceCar).draggable({
start : function (event, ui) {
// здесь располагается программный код
}
});
```

Функция, назначенная для события `start`, предусматривает два параметра: `event` и `ui`. Параметр `event` является объектом события jQuery и содержит подробную информацию об элементе, которому назначено событие, экранные координаты указателя мыши и прочие данные. (Вы можете прочитать об объекте события и о способе его использования в разделе «Объект события» главы 5.) Параметр `ui` — это объект с четырьмя свойствами:

- **Свойство `ui.helper`** является объектом jQuery, содержащим ссылку на элемент, который перетаскивается по экрану. Как правило, это тот же элемент, к которому вы применили метод `draggable()`. Тем не менее так бывает не всегда. Если вы указали для параметра `helper` значение `'clone'` или передали ему функцию для генерирования вспомогательного HTML-кода (как описано в предыдущем разделе), то свойство `ui.helper` ссылается на элемент, отличный от перетаскиваемого: либо на его копию, либо на объект jQuery, созданный в качестве вспомогательного элемента (см. предыдущий раздел). Это полезно, например, в случае с корзиной покупок, когда перемещение товара с одного места в корзину нарушает макет страницы. Вместо этого, покупатель должен перетащить в корзину копию изображения товара.

  Используйте свойство `ui.helper`, когда вы хотите выполнить действие над элементом, который визуально перетаскивается по странице. Например, вам нужно, чтобы при перетаскивании размер элемента увеличивался в два раза. В этом случае вы можете задать свойство CSS для объекта `ui.helper` следующим образом:

  ```
  $('#photo').draggable({
  start : function (event, ui) {
  ```

```
ui.helper.css('transform', 'scale(2)');
}
});
```

В данном случае используется метод jQuery `css()` (см. раздел «Чтение и изменение свойств CSS» главы 4), а свойство CSS `transform` удваивает размер элемента. Вы также можете использовать метод jQuery `addClass()`, чтобы добавить класс к элементу, когда пользователь начнет перетаскивать его, например, стиль, выделяющий данный элемент. Когда посетитель прекратит перетаскивание, вы можете удалить класс с помощью метода `removeClass()` (см. описание события `stop` далее).

ПРИМЕЧАНИЕ

Чтобы узнать о преобразованиях CSS подробнее, посетите страницу: **developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_transforms**.

- **Свойство `ui.position`** предоставляет `x` и `y` координаты левого верхнего угла элемента `helper` (который визуально перетаскивается по экрану). Это значение свойства CSS `position`, и на него может повлиять положение родительского элемента (или любого другого позиционированного предка). Например, если у вас есть перетаскиваемый элемент внутри `div`, и этот `div` абсолютно позиционирован на странице, то значения свойств `left` и `top` перетаскиваемого элемента будут указаны относительно левого верхнего угла позиционированного предка.

  Свойство `ui.position` является объектом JavaScript, состоящим из двух свойств `top` и `left`. Свойство `top` — это положение верха элемента, соответствующее расстоянию в пикселах от верхней части ближайшего позиционированного предка (то есть элемента с абсолютным или относительным позиционированием, в который заключен перетаскиваемый элемент). Свойство `left` — это положение левого края вспомогательного элемента относительно левого края ближайшего позиционированного предка. Доступ к этому свойству может быть полезным для определения местоположения вспомогательного элемента после прекращения перетаскивания (или во время перетаскивания) (см. описания событий `drag` и `stop` далее).

Вы можете получить доступ к этим значениям так: `ui.position.top` и `ui.position.left`.

- **Свойство `ui.offset`** также предоставляет объект с двумя свойствами `top` и `left`. Местоположение рассчитывается относительно левого верхнего угла окна браузера. Свойства `ui.position` и `ui.offset` будут иметь одинаковые значения, если перетаскиваемый элемент не находится внутри какого-либо другого элемента, чье свойство CSS `position` имеет значение `absolute` или `relative`. Свойство `ui.offset.top` определяет, на каком расстоянии от верхнего края окна находится верхний край вспомогательного элемента. Свойство `ui.offset.left` определяет, на каком расстоянии в пикселах от левого края окна браузера находится перетаскиваемый элемент.
- **Свойство `ui.originalPosition`** содержит те же два свойства `top` и `left`, как и `ui.position` и `ui.offset`. Тем не менее свойство `ui.originalPosition` указывает начальное положение перетаскиваемого элемента, то есть точку, в которой находился элемент перед тем, как пользователь начал его перетаскивать. Как и `ui.position`, свойство `ui.originalPosition` зависит от позиционированных предков.

Событие `drag`

Перетаскиваемые элементы также активируют событие `drag` при их перемещении по экрану. Вы можете создать функцию, которая работает постоянно в процессе перетаскивания элемента. Например, это событие можно использовать, чтобы оставить искрящийся след от объекта при его перемещении в окне. Поскольку это событие срабатывает множество раз при перетаскивании элемента, попытайтесь ограничить объем работы, выполняемой функцией. Если в ответ на событие `drag` выполняется много сложных операций, то работа браузера может замедлиться.

Функция события `drag` принимает те же два параметра, что и событие `start`, описанное ранее, — `event` и `ui`.

Допустим, вы хотите отобразить текущее положение перетаскиваемого элемента при его перемещении по экрану. Предположим, что для отображения этих значений на странице присутствуют два элемента `span` с идентификаторами `left` и `top`. В процессе перетаскивания элемента вы можете постоянно обновлять эти значения следующим образом:

```
$('#raceCar').draggable({
drag : function (event, ui) {
$('#left').text(ui.position.left); $('#top') ↵
.text(ui.position.top);
}
});
```

Событие `stop`

Событие `stop` виджета Draggable работает так же, как событие `start`, за исключением того, что оно срабатывает, когда пользователь прекращает перетаскивать элемент и отпускает кнопку мыши. Это не обязательно означает, что пользователь бросил элемент на соответствующее целевое место (обсуждается в следующем разделе). Это также не означает, что пользователь полностью завершил процесс перетаскивания элемента. Например, пользователь мог щелкнуть по перетаскиваемому элементу и начать его перемещение (при этом срабатывает событие `start`), продолжить перемещать его по странице (постоянно срабатывает событие `drag`), прекратить перетаскивание (срабатывает событие `stop`), а затем щелкнуть по этому элементу снова и опять перетащить его, запустив все те же события.

Вы можете использовать событие `stop` так же, как и событие `start`. Например, вы использовали вышеприведенный код, чтобы обеспечить удвоение размера элемента при его перетаскивании. Вы можете восстановить нормальный размер элемента при прекращении его перемещения с помощью следующего кода:

```
$('#photo').draggable({
stop : function (event, ui) {
ui.helper.css('transform', 'scale(1)');
}
});
```

Виджет Droppable

Виджет Draggable бывает полезен для диалоговых окон или других элементов страницы, которые вы можете перемещать по экрану (например, панели инструментов). Тем не менее добавив виджет Droppable, вы

можете создать интерактивные приложения, в которых перетаскивание одного элемента на другой может привести к некоторому результату.

Например, приложение для обмена фотографиями может предусматривать возможность перетащить эскиз изображения в мусорную корзину, чтобы удалить соответствующее фото с сайта. Ресурс, посвященный изучению языков, может предусматривать тестирование пользователей, при котором они могут перетащить слово на соответствующую картинку, чтобы продемонстрировать знание лексики.

Использование виджета Droppable

Виджеты Droppable полезны только в сочетании с виджетами Draggable. Виджет Droppable работает как «область бросания» для перетаскиваемых элементов. Когда элемент бросается в эту область, виджет Droppable может также активировать дополнительный программный код. Вы можете превратить в бросаемый элемент любой элемент страницы. Разумеется, посетитель должен понимать, что он может перетащить что-нибудь в эту область, например, значки корзины покупок и мусорной корзины являются визуальными метафорами, которые ясно указывают на то, что в них можно бросить некие элементы. Однако вы также можете превратить в виджет Droppable элементы `div`, если явно дадите посетителю понять, что и куда они могут перетащить.

Использовать виджет Droppable очень просто:

1. **Следуйте инструкциям, приведенным в разделе «Добавление плагина jQuery UI на веб-страницу» главы, чтобы прикрепить файлы CSS и JavaScript.**

   Помните, что у плагина jQuery UI есть свои собственные файлы CSS и JavaScript, и что ссылка на JavaScript-файл jQuery UI должна следовать после ссылки на JavaScript-файл библиотеки jQuery.

2. **Добавьте функцию `$(document).ready()` на свою страницу или во внешний JavaScript-файл:**

   ```
   $(document).ready(function() {

   }); // окончание ready
   ```

   Как уже говорилось в разделе «Больше концепций для событий jQuery» главы 5, этот шаг необходим только в том случае, если вы помещаете свой JavaScript-код в раздел заголовка страницы перед

большей частью кода HTML. Некоторые программисты помещают свой код JavaScript в нижней части страницы перед закрывающим тегом </body>. В этом случае вы можете пропустить функцию `$(document).ready()`.

Виджеты Droppable не имеют смысла без элементов, которые вы можете на них перетащить, поэтому вам следует всегда добавлять на страницу, по крайней мере, один перетаскиваемый элемент.

3. **Используйте jQuery, чтобы выбрать один или несколько элементов страницы и применить к ним виджет Draggable:**

```
$(document).ready(function() {
$('#dialog').draggable();
}); // окончание ready
```

В данном примере у вас есть страница каталога, содержащая несколько товаров. К каждому изображению товара применен класс `product`, поэтому приведенный выше код сделает каждое из этих изображений перетаскиваемым.

4. **Используйте jQuery, чтобы выбрать область бросания и примените к ней виджет Droppable:**

```
$('.product').draggable();
$('#cart').droppable();
}); // окончание ready
```

Элемент страницы с идентификатором `cart` теперь является областью бросания. Однако это только начало. Вы должны установить различные параметры бросаемого элемента для того, чтобы что-то произошло, например, расчет общей суммы заказа при добавлении очередного товара в корзину. Вы можете сделать это, передав объект, содержащий параметры и функции методу `droppable()`.

5. **Добавьте параметры в функцию `droppable()`:**

```
$(document).ready(function() {
$('.product').draggable();
$('#cart').droppable({
activeClass : 'highlight',
drop : function (event,ui) {
```

```
alert('Товар добавлен');
}
});
}); // окончание ready
```

Вы узнаете о различных параметрах этого виджета далее, однако давайте кратко рассмотрим то, что делает приведенный выше код. Во-первых, параметр `activeClass` дает jQuery UI задание добавить класс с именем `highlight` к элементу, являющемуся областью бросания. Этот класс вы используете для применения правила CSS к бросаемому элементу. Код CSS может визуально выделить область бросания путем добавления яркого фона или яркой красной границы.

Второй параметр, `drop`, является обработчиком события. Он позволяет запускать функцию при бросании элемента в соответствующую область. В этом случае появляется окно предупреждения с соответствующим сообщением.

Параметры виджета Droppable

Так же как и виджет Draggable, виджет Droppable предусматривает не много параметров (главная роль виджета Droppable заключается в том, чтобы принимать элементы и, самое главное, запускать при этом функции):

- **`accept`**. Данный параметр определяет, какие перетаскиваемые элементы могут быть добавлены в виджет Droppable. Вы можете назначить селектор или функцию. Селектор соответствует селектору перетаскиваемого элемента. Например, если у вас есть множество изображений с классом `photo`, и вы хотите иметь возможность бросить их на определенный элемент, то вы можете задать данный параметр следующим образом:

  ```
  accept : '.photo'
  ```

- **`activeClass`**. Вы можете выделить область бросания в процессе перемещения перетаскиваемого элемента. Например, вы создали файловую систему, которая позволяет пользователям просматривать файлы на сервере, перемещать их в другие папки, переименовывать их и т. д. Также предусмотрена возможность перетащить файл в корзину, чтобы удалить его. Мусорная корзина представляет собой виджет Droppable, а файл — виджет Draggable. Когда посетитель начинает перетаскивать

файл, вы можете применить класс, который каким-то образом выделяет мусорную корзину (например, меняет фоновое изображение на изображение мусорной корзины с открытой крышкой, готовой к приему файла). Просто назначьте имя класса (без точки) этому свойству, и плагин jQuery UI добавит этот класс к области бросания при перемещении допустимого перетаскиваемого элемента:

```
activeClass : 'highlight'
```

- **disabled**. Данный параметр определяет, является ли элемент активной областью бросания. Если значением этого свойства является `true`, то вы ничего не можете бросить на элемент. Вы можете использовать это свойство после создания области бросания в качестве способа отключения данной возможности. Например, у вас есть область бросания, и вы хотите, чтобы пользователи могли бросить в нее только 5 элементов. После того как в этой области появятся пять элементов, вы отключаете эту область, чтобы пользователи больше не могли ничего в нее поместить (вы можете использовать событие `drop`, описанное далее, чтобы вызвать функцию, которая отключает область бросания):

  ```
  $('#dropZone').droppable({
  disabled : true
  });
  ```

  Вы, наверное, считаете, что для превращения элемента в область бросания, вам достаточно просто не использовать параметр `disabled`. Это правда, вы, вероятно, не будете задействовать данный параметр при первом применении к элементу виджета Droppable.

  Тем не менее он может пригодиться после того, как кто-то бросил элемент на виджет Droppable. Например, вам нужно, чтобы посетитель мог добавить в корзину покупок только пять товаров. После того как покупатель поместит в нее пятый элемент, вы можете отключить область бросания, чтобы предотвратить дальнейшее пополнение корзины.

- **hoverClass**. Вы также можете применить класс, когда над областью бросания оказывается допустимый перетаскиваемый элемент (рис. 12.3). Например, вы можете добавить класс к упомянутой ранее мусорной корзине, только когда над ней оказывается перетаскиваемый файл:

  ```
  hoverClass : 'openTrashcan'
  ```

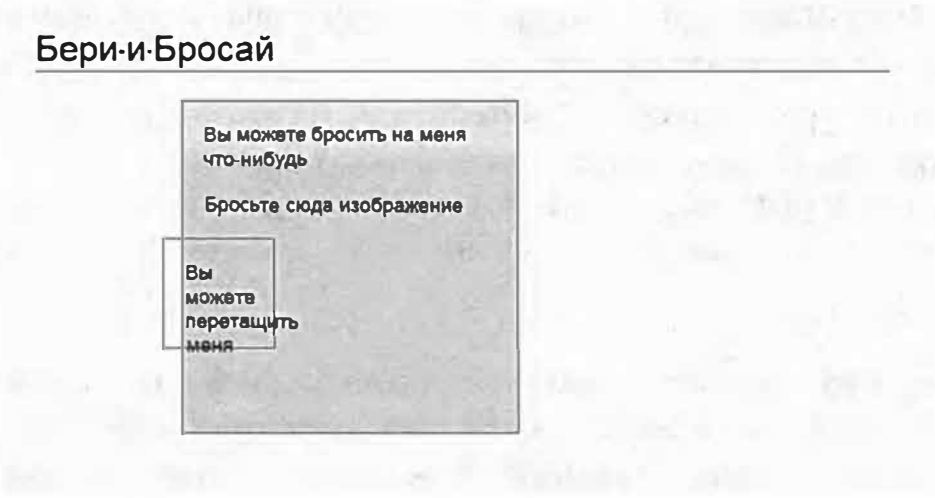

Рис. 12.3. Используя параметр `hoverClass` виджета Droppable, вы можете применить к области бросания класс, когда над ней оказывается перетаскиваемый элемент. В данном случае большая рамка становится ярко-желтой, когда над ней оказывается рамка меньшего размера (изменение цвета обеспечивается стилем класса CSS). Этот стиль добавляет и фоновый цвет, и фоновое изображение, однако вы можете создать стиль класса, который добавляет к области бросания яркий контур или даже дополнительную CSS-анимацию, заставляющую эту область пульсировать и изменять цвет с белого на другой

- **`scope`**. Данный параметр работает аналогично одноименному параметру виджета Draggable (см. раздел «Параметры виджета Draggable» ранее в данной главе). Он позволяет группировать связанные виджеты Draggable и Droppable.

  ```
  scope : 'calendar'
  ```

  Используемое в данном примере имя `'calendar'` не имеет особого значения. Вы можете выбрать любое имя, только не забудьте использовать то же самое имя для связанных виджетов Draggable и Droppable.

- **`tolerance`**. Этот параметр определяет, когда перетаскиваемый элемент считается находящимся над областью бросания, и предусматривает четыре возможных значения:

  - **`'fit'`** — перетаскиваемый элемент должен полностью находиться в пределах области бросания;
  - **`'intersect'`** — перетаскиваемый элемент должен перекрывать область бросания не менее чем на 50% в обоих направлениях. Другими словами, большая часть перетаскиваемого элемента должна находиться внутри области бросания. Это значение является нормальным для виджетов Droppable;

- `'pointer'` — в пределах области бросания должен находиться только указатель мыши;
- `'touch'` — перетаскиваемый элемент должен всего лишь касаться одной из сторон области бросания.

Параметр `'fit'` хорошо подходит для подтверждения того, что пользователь действительно хочет поместить перетаскиваемый элемент в область бросания, однако он требует, чтобы размер перетаскиваемого элемента был меньше размера этой области. Наиболее распространенным значением является `'intersect'`, поскольку оно не требует от пользователя чрезмерной точности:

```
tolerance : 'intersect'
```

ПРИМЕЧАНИЕ

Полный список параметров, методов и событий виджета Droppable вы можете найти на странице: **api.jqueryui.com/droppable/**.

События виджета Droppable

Самое интересное при работе с виджетами Droppable состоит в том, чтобы выполнить некоторое действие, когда элемент попадает в область бросания, перемещается на нее или из нее. Например, вы можете рассчитывать и отображать общую стоимость покупки каждый раз, когда покупатель помещает товар в корзину, а затем пересчитывать эту стоимость, если покупатель вытаскивает товар из нее.

Виджет Droppable поддерживает несколько различных событий, каждое из которых инициируется различными взаимодействиями с областью бросания. Вы добавляете функцию к событию, чтобы ваша программа выполнила некоторое действие в ответ на перетаскивание элемента, перемещение его над областью бросания или оставление элемента в этой области. Например, у вас есть приложение для создания списка дел: когда посетитель перетаскивает элемент из списка и помещает его на значок в виде мусорной корзины, это элемент удаляется из списка дел и со страницы. В этом случае используется событие `drop`.

Далее мы рассмотрим каждое событие, начиная с наиболее часто используемого.

Событие `drop`

Данное событие вызывает функцию, когда элемент помещается в область бросания. Бросаемый элемент должен быть допустимым объектом, то есть объектом, который указан в параметре `accept` (см. выше), или имеет тот же параметр `scope`, что и область бросания (см. раздел «Параметры виджета Draggable» ранее в главе).

Данное событие назначено в качестве свойства, передаваемого виджету Droppable в качестве части объекта с параметрами. Например, у вас есть элемент `div` с идентификатором `trashcan`. Вы можете превратить этот элемент `div` в область бросания и назначить ему событие `drop` следующим образом:

```
$('#trashcan').droppable({
drop : function (event, ui) {
// здесь располагается программный код
}
});
```

Функция, назначенная событию `drop`, имеет два параметра: `event` и `ui`. Параметр `event` является объектом события jQuery (см. раздел «Объект события» главы 5), который содержит информацию о том, какой элемент получает событие, экранные координаты указателя мыши и другие данные. Параметр `ui` соответствует параметру, используемому в событиях виджета Draggable (см. раздел «События виджета Draggable» ранее в главе).

- **Свойство `ui.helper`** является объектом jQuery, содержащим ссылку на элемент, который визуально перетаскивается по экрану, и соответствует свойству `ui.helper`, описанному в разделе «Событие `start`» ранее в главе.

  Используйте свойство `ui.helper`, когда вы хотите выполнить действие над предметом, который визуально перетаскивается по странице. Например, если вам нужно, чтобы элемент «взрывался» при попадании в мусорную корзину, вы можете использовать следующий код:

  ```
  $('#trashcan').droppable({
  drop : function (event, ui) {
  ui.helper.hide('explode');
  }
  });
  ```

Значение `'explode'` в данном коде представляет собой замечательный эффект jQuery UI, о котором вы узнаете далее в главе в разделе «Эффекты».

- **Свойство `ui.draggable`** является объектом jQuery, содержащим ссылку на элемент, к которому применен метод `draggable()`. Во многих случаях значение данного свойства совпадает со значением свойства `ui.helper`. Однако если вы укажете для свойства `helper` (см. раздел «Параметры виджета Draggable» ранее в главе) значение `'clone'`, то по экрану станет перемещаться клонированный элемент, а исходный элемент будет оставаться на месте.

  Данное свойство также может пригодиться в случае со страницей оформления заказа. Допустим, вы создали систему, которая отслеживает запас товара. Страница каталога может отображать фотографию товара и метку с указанием оставшегося количества, например, 10. Когда покупатель щелкает по изображению товара, он перетаскивает в корзину его клон. Когда он помещает этот клон в корзину, функция `drop` может обновить элемент `ui.draggable`, чтобы количество оставшегося товара уменьшилось на единицу (и в то же время вам нужно использовать технологию Ajax (глава 13) для отправки данных заказа на сервер).

- **Свойство `ui.position`** предоставляет `x` и `y` координаты положения верхнего левого угла вспомогательного элемента, когда он помещается в целевую область. Оно аналогично одноименному свойству, описанному в разделе «Событие `start`» ранее в главе.

- **Свойство `ui.offset`** предоставляет данные о положении верхнего и левого края перетаскиваемого элемента относительно окна браузера. Это свойство соответствует свойству `ui.offset`, описанному ранее в разделе «Событие `start`».

- **Свойство `ui.originalPosition`** предоставляет данные о положении верхнего и левого края перетаскиваемого элемента перед тем, как пользователь начинает его перемещать. Это свойство соответствует свойству `ui.originalPosition`, описанному ранее в разделе «Событие `start`».

При использовании области бросания вы, вероятно, будете часто использовать событие `drop`. Одна из задач, которую вам может понадобиться решить, заключается в том, чтобы предотвратить перемещение элемента за пределы области бросания. Вернемся к примеру с игрой в шашки. Вы можете сделать так, чтобы, передвинув фигуру, пользова-

тель не мог вернуть ее на исходную позицию. Другими словами, вы хотите, чтобы эта фигура «закрепилась» в том месте, куда ее переместили. Вы можете сделать это путем отключения перетаскиваемого элемента после его попадания в область бросания:

```
$('.square').drop({
drop : function (event, ui) {
ui.helper.draggable({
disabled : true
});
}
});
```

Вы используете объект `helper` параметра `ui` (то есть объект, который был перемещен в область бросания), а затем вызываете метод `draggable()`, чтобы отключить перетаскивание, указав для свойства `disabled` этого элемента значение `true`. Имейте в виду, что свойство `disabled` относится к перетаскиваемому элементу (см. раздел «Параметры виджета Draggable» ранее в главе), а не к области бросания. Чтобы снова сделать элемент перетаскиваемым, вам нужно будет указать для свойства `disabled` значение `false` (см. раздел «Параметры виджета Draggable» ранее в главе).

Событие `activate`

Когда пользователь начинает перетаскивать элемент, являющийся допустимым для области бросания (см. приведенные выше описания параметров виджета Droppable `accept` и `scope`), срабатывает событие `activate`. Вы можете добавить программный код для обеспечения реакции на это событие. Например, вы создали веб-приложение, которое позволяет пользователю перетащить одну или несколько фотографий на элемент `div`, а затем щелкнуть по кнопке «Отправить», чтобы отправить эти фотографии другу по электронной почте. Вы можете обеспечить появление в области бросания метки «Бросьте фотографию сюда», когда пользователь начинает перетаскивать фотографию.

Вы можете добавить в событие `activate` функцию, которая будет создавать в области бросания надпись «Бросьте фотографию сюда». Если предположить, что областью бросания является элемент `div` с идентификатором `photoZone`, то вы можете добавить сообщение в область бросания при перетаскивании фото следующим образом:

```
$('#photoZone').droppable({
activate : function (event, ui) {
$(this).append('<p id="dropMessage">Бросьте ↵
фотографию сюда</p>');
}
});
```

Фрагмент `$(this)` относится к области бросания (обратитесь к разделу «Ключевые слова `this` и `$(this)`» главы 4, если вам нужно вспомнить о том, что такое ключевое слово `$(this)` и как оно работает). Метод `append()` просто добавляет в тег HTML-код в качестве последнего дочернего элемента (см. раздел «Добавление содержимого на страницу» главы 4). Как и событие `drop`, функция `activate` принимает те же два параметра: `event` и `ui` (см. раздел «Событие `drop`» ранее в главе).

Событие `deactivate`

Событие `deactivate` — этот противоположность события `activate`. Оно запускает функцию, когда пользователь прекращает перетаскивание допустимого элемента, то есть, отпускает кнопку мыши. Вы можете использовать это событие, чтобы отменить действие, выполняемое событием `activate`. Например, чтобы добавить сообщение в область бросания в процессе перетаскивания элемента, а затем удалить это сообщение, когда пользователь прекращает перетаскивание, вы можете использовать следующий код:

```
$('#photoZone').droppable({
activate : function (event, ui) {
$(this).append('<p id="dropMessage">Бросьте ↵
фотографию сюда</p>');
},
deactivate : function (event, ui) {
$('#dropMessage').remove();
}
});
```

Событие `over`

Вы даже можете запустить функцию, когда посетитель перетаскивает элемент над областью бросания. Событие `over` срабатывает, как

только перетаскиваемый элемент оказывается над областью бросания перед тем, как посетитель бросит его там. Например, вы можете использовать это событие, чтобы добавить сообщение, отображаемое, если нерешительный покупатель делает паузу, прежде чем бросить товар в корзину: «Вы знаете, что будете отлично смотреться в этих туфлях. Просто купите их!»

Или, допустим, вы создали область бросания в виде мусорной корзины, чтобы пользователи могли удалить фотографии со своей страницы. Обычно мусорная корзина закрыта, но когда над ней оказывается бросаемый элемент, крышка корзины открывается. После бросания элемента в корзину вы можете использовать событие `drop`, чтобы заменить изображение пустой корзины изображением корзины, заполненной мусором.

Событие `over` программируется так же, как любое другое событие виджета Droppable — вам нужно просто назначить функцию свойству `over`:

```
$('#trashcan').droppable({
over : function (event, ui) {
$('#trashCanImage').attr('src', ↵
'images/open-lid-can.png');
}
});
```

В данном примере вы просто заменяете изображение другим файлом, однако вы можете создать гораздо более сложные функции для этого (или любого другого) события.

ПРИМЕЧАНИЕ

Свойство `tolerance` (см. раздел «Параметры виджета Droppable» ранее) определяет условия, при которых плагин jQuery UI считает, что перетаскиваемый элемент находится над, бросается в или удаляется из области бросания.

Событие `out`

Наконец вы можете запустить функцию, когда посетитель удаляет элемент из области бросания. Например, покупатель бросил товар в корзину, и в этот момент вы рассчитываете общую стоимость покупки.

Затем, когда покупатель решает, что он больше не заинтересован в этом товаре, он вытаскивает его из корзины. Событие `out` позволяет вычесть стоимость удаленного товара из общей суммы покупки:

```
$('#shoppingCart').droppable({
drop : function (event, ui) {
// выполнить вычисления, чтобы обновить общую сумму
},
out : function (event, ui) {
// вычесть стоимость товара из общей суммы
}
});
```

Эти несколько примеров показывают, что вы будете часто использовать пары событий, чтобы произвести противоположные действия. Например, событие `out` с событием `drop`, а событие `deactivate` с событием `activate`.

Drag-and-Drop на практике

В данном примере в одной программе используются виджеты Draggable и Droppable. Вы создадите простое приложение типа «Бери-и-Бросай», которое демонстрирует работу основных компонентов каждого из виджетов. Законченная программа позволит вам перетащить фото в мусорную корзину, чтобы зрелищно удалить его (рис. 12.4).

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *12_02.html* в папке *глава 12*.**

   Этот файл уже содержит ссылку на необходимые файлы CSS и JavaScript, а также функцию `$(document).ready()` (см. раздел «Больше концепций для событий jQuery» главы 5).

Если вы сейчас просмотрите эту страницу, то увидите слева изображение мусорной корзины и ряд фотографий справа (см. рис. 12.4). HTML-код для мусорной корзины и фотографий выглядит следующим образом:

```
<img src="../_images/trashcan-empty-icon.png" ↵
id="trashcan">
<div id="photos">
<img src="../_images/small/slide1_h.jpg">
<img src="../_images/small/slide2_h.jpg">
<img src="../_images/small/slide3_h.jpg">
<img src="../_images/small/slide4_h.jpg">
<img src="../_images/small/slide5_h.jpg">
<img src="../_images/small/slide6_h.jpg">
</div>
```

Рис. 12.4. Добавление на страницу перетаскиваемых и бросаемых элементов создает веселую и интерактивную атмосферу для посетителей вашего сайта. В данном руководстве вы узнаете, как создавать перетаскиваемые элементы и заставлять их исчезать при попадании в соответствующую целевую зону

Изображение мусорной корзины имеет идентификатор `trashcan`, а фотографии находятся внутри элемента `div` с идентификатором `photos`. Сначала вы сделаете фотографии перетаскиваемыми.

2. **В пустой строке внутри функции `$(document).ready()` добавьте код, выделенный полужирным шрифтом:**

```
$(document).ready(function() {
$('#photos img').draggable();
}); // окончание ready
```

Этот код выбирает все элементы `img` внутри элемента `div` с идентификатором `photos` и превращает их в перетаскиваемые объекты. Поскольку цель данного проекта заключается в том, чтобы добавить изображение в мусорную корзину, вам нужно обеспечить возврат фотографии на прежнее место, если посетитель бросит ее не в мусорную корзину, а куда-либо еще.

3. **Добавьте в функцию `draggable()` литерал объекта со свойством `revert` и значением `'invalid'`:**

```
$('#photos img').draggable({
revert : 'invalid'
});
```

Параметр `revert` (см. раздел «Параметры виджета Draggable» ранее в главе) позволяет возвратить перетаскиваемый элемент на исходную позицию. Задав для этого параметра значение `'invalid'`, вы просто говорите: «Верни этот элемент на исходную позицию (`revert`), если он будет брошен в недопустимом месте (`'invalid'`). Что значит недопустимое место? Таким местом считается любой элемент, который не определен в качестве виджета Droppable. Поскольку вы еще не добавили виджет Droppable, фотографии будут всегда возвращаться на исходную позицию после того, как вы их бросите.

4. **Сохраните страницу и просмотрите ее в браузере. Перетащите фотографию и бросьте ее.**

Фотография должна вернуться на исходную позицию. Если этого не произошло, проверьте свой код и проанализируйте содержимое консоли JavaScript на предмет наличия ошибок (см. раздел «Отслеживание ошибок» главы 1).

Далее вы будете использовать события виджета Draggable, чтобы обеспечить выполнение некоторого действия при перетаскивании фотографии по странице. В частности, вы примените к перетаскиваемой фотографии преобразование CSS.

5. **Введите запятую после фрагмента `invalid` в коде, который вы только что добавили, нажмите клавишу Enter и добавьте обработчик события `start` следующим образом:**

```
$('#photos img').draggable({
revert : 'invalid',
start : function (event, ui) {
}
});
```

Параметр `start` требует использования функции в качестве значения. Эта функция является обработчиком события, подобным тому, который вы использовали для jQuery-функций `click()` и `mouseover()` (см. раздел «Использование событий: способ jQuery» главы 5). Тем не менее `start` не является настоящим событием браузера. Это специальное событие, созданное командой jQuery UI, которое срабатывает в тот момент, когда кто-то начинает перетаскивать элемент. Функция пуста, поэтому вам предстоит добавить в нее программный код. В данном случае вы добавите простой код CSS с помощью jQuery-функции `css()` (см. раздел «Чтение и изменение свойств CSS» главы 4).

6. **Внутри только что добавленной функции введите следующий код (выделен полужирным шрифтом):**

```
$('#photos img').draggable({
revert : 'invalid',
start : function (event, ui) {
ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
}
});
```

Как вы помните из раздела «Событие `start`» данной главы, функция, присвоенная событиям виджета Draggable, принимает два параметра: объект `event` и объект, представляющий перетаскиваемый элемент — параметр `ui`. Вы можете использовать параметр `ui`, чтобы выбрать перетаскиваемый элемент и совершить над ним некое действие. Фрагмент `ui.helper` (см. раздел «Событие `start`» ра-

нее в данной главе) представляет собой фактический элемент, который пользователь перетаскивает по экрану, а поскольку это объект jQuery, вы можете применить к нему любую из множества jQuery-функций. В данном случае функция `css()` применяет к элементу преобразование CSS, слегка поворачивая его и увеличивая в 1,5 раза. Другими словами, при перетаскивании изображение будет вращаться и увеличиваться в размере. Далее вы сделаете так, чтобы фото принимало исходный вид, когда посетитель прекращает его перетаскивание.

ПРИМЕЧАНИЕ

Подробнее о преобразованиях CSS вы можете узнать на странице: **www.sitepoint.com/css3-transformations-2d-functions/**.

7. Добавьте в объект параметр события `stop`:

```
$('#photos img').draggable({
revert : 'invalid',
start : function (event, ui) {
ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
},
stop : function (event, ui) {
ui.helper.css('transform', 'rotate(0deg) scale(1)');
}
});
```

Не забудьте про запятую после функции `start`.

Эта функция просто отменяет действия кода CSS, который был применен в начале процесса перетаскивания, восстанавливает исходный размер и ориентацию фотографии. Если вы сохраните файл и просмотрите его в браузере, то вы увидите нечто странное: если вы щелкнете по фотографии и начнете ее перетаскивать, то заметите, что она появляется из-под фото, расположенного справа от нее (рис. 12.5). Причина этого явления объясняется в подписи к рис. 12.5. Чтобы исправить это, просто увеличьте значение параметра `z-index` перетаскиваемого элемента.

8. Добавьте в объект последний параметр `zIndex`.

```
$('#photos img').draggable({
revert : 'invalid',
start : function (event, ui) {
ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
},
stop : function (event, ui) {
ui.helper.css('transform', 'rotate(0deg) scale(1)');
},
zIndex : 100
});
```

Не забывайте про запятую после функции `stop`. Параметр `zIndex` (см. раздел «Параметры виджета Draggable» ранее в главе) просто регулирует значение свойства CSS `z-index` перетаскиваемого элемента. Чем выше значение `z-index`, тем выше в стопке расположен элемент. Значение 100 является достаточно большим, чтобы гарантировать, что перетаскиваемый элемент не будет перекрываться другими элементами на странице.

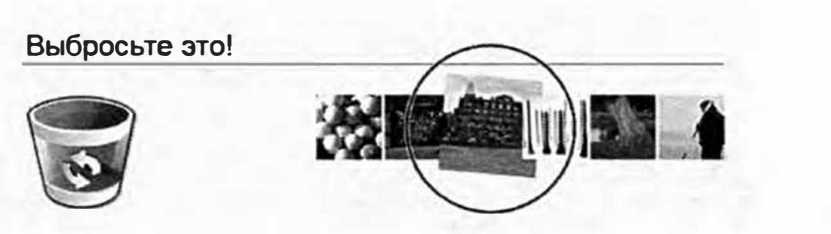

Рис. 12.5. Значение `z-index` элемента зависит от нескольких факторов. Например, порядок фрагментов кода HTML может повлиять на порядок размещения элементов в стопке. В данном случае элемент `img` выбранной фотографии отеля (выделена на рисунке) находится в HTML-коде страницы перед элементом `img`, соответствующим изображению площадки для гольфа (см. фрагмент HTML-кода в шаге №1 ранее). Соответственно, фотография отеля отображается под фотографией, на которой изображаются площадки для гольфа

9. Сохраните файл и просмотрите его в браузере. Перетащите фотографию по странице.

Фотография должна немного повернуться и увеличиться при перетаскивании, а после его прекращения — вернуться к своему нормальному размеру и ориентации. Кроме того, в начале процесса перетаскивания фотография должна располагаться над всеми остальными изображениями, находящимися рядом с ней.

Работа с фотографиями закончена. Далее вы превратите изображение мусорной корзины в область бросания.

10. **После функции `draggable()` создайте новую строку и добавьте в нее следующий код:**

```
$('#trashcan').droppable();
```

Данный код превращает изображение мусорной корзины в виджет Droppable. Если вы сейчас сохраните страницу и просмотрите ее, то увидите, что вы можете бросить фотографии на мусорную корзину.

Тем не менее фотографии не возвращаются на свое исходное место на экране, поскольку мусорная корзина является действительной областью бросания.

Чтобы добавить зрелищности, вы обеспечите выделение мусорной корзины в тот момент, когда посетитель начинает перетаскивать одну из фотографий.

Кроме того, вы сделаете изображение корзины более ярким, чтобы посетители знали о возможности переместить в нее фотографию.

11. **Добавьте объект с параметрами в функцию `droppable()`:**

```
$('#trashcan').droppable({
activeClass : 'highlight'
});
```

Параметр `activeClass` (см. раздел «Параметры виджета Droppable» ранее в главе) просто добавляет виджету Droppable имя класса, когда посетитель перемещает по экрану допустимый перетаскиваемый элемент. В данном случае класс `highlight`, определенный в файле *interactions.css*, изменяет значение параметра `opacity` изображения мусорной корзины на 1 (полностью непрозрачный). Другое правило в этом файле CSS делает данное изображение на 60% непрозрачным (значение 0,6) при загрузке страницы. Благодаря изменению степени непрозрачности с 0,6 до 1 с помощью кода CSS, мусорная корзина кажется более яркой при перетаскивании фотографии.

Хотя это незначительное изменение производит впечатление и дает подсказку посетителям страницы, вы еще не предусмотрели действий, выполняемых в тот момент, когда фотография помещается в мусорную корзину. Для этого вам придется использовать событие `drop` виджета Droppable.

12. **Введите запятую после только что добавленного фрагмента `'highlight'`, нажмите клавишу Enter и добавьте обработчик события `drop` следующим образом:**

```
$('#trashcan').droppable({
activeClass : 'highlight',
drop : function (event, ui) {
}
});
```

Параметр `drop` требует использования функции в качестве значения, как и события `start` и `stop` перетаскиваемых элементов. Эта функция пуста и пока не производит никаких действий, однако вы можете это исправить, например, сделать так, чтобы брошенная фотография взорвалась!

13. **В функции `drop` добавьте код, выделенный полужирным шрифтом:**

```
$('#trashcan').droppable({
activeClass : 'highlight',
drop : function (event, ui) {
ui.helper.hide('explode');
}
});
```

Фрагмент `ui.helper` относится к перетаскиваемому элементу (см. раздел «Событие `start`» ранее в главе). Метод `hide()` является функцией jQuery, которую вы использовали ранее (см. раздел «Основы отображения и сокрытия» главы 6). Тем не менее плагин jQuery UI предусматривает дополнительные эффекты для сокрытия и отображения элементов. Параметр `explode` производит интересный анимационный эффект исчезновения фотографии со страницы. (Вы прочитаете об этих эффектах подробнее далее в главе в разделе «Эффекты jQuery UI».)

И последнее: вам предстоит заменить изображение пустой мусорной корзины изображением корзины, заполненной мусором, после помещения в нее фотографии.

14. **Добавьте последнюю строку кода внутри функции `drop`:**

```
$('#trashcan').droppable({
activeClass : 'highlight',
```

```
drop : function (event, ui) {
ui.helper.hide('explode'); $(this).attr('src', ↵
'../_images/trashcan-full-icon.png');
}
});
```

Ключевое слово `$(this)` относится к изображению мусорной корзины, то есть к элементу `img` с идентификатором `trashcan`. Вы просто изменяете атрибут `src` данного элемента, чтобы указать на новое изображение. (Вы узнали о том, как это делается, в главе 7.)

Итоговый код должен выглядеть следующим образом:

```
$(document).ready(function() {
$('#photos img').draggable({
revert : "invalid",
start : function (event, ui) {
ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
},
stop : function (event, ui) {
ui.helper.css('transform', 'rotate(0deg) scale(1)');
},
zIndex : 100
});
$('#trashcan').droppable({
activeClass : 'highlight',
drop : function (event, ui) {
ui.helper.hide('explode');
$(this).attr('src','../_images/trashcan-full-icon.png');
}
});
}); // окончание ready
```

15. Сохраните файл и просмотрите его в браузере.

Если вы перетащите фотографии на изображение мусорной корзины, то они взорвутся и исчезнут со страницы (см. рис. 12.4). Вы мог-

ли бы добавить гораздо больше программного кода в событие `drop` в данном примере. Например, вы можете отправить запрос Ajax на сервер и дать ему команду удалить фотографию из учетной записи пользователя (вы узнаете о технологии Ajax в главе 13). Однако этот краткий урок, наверняка, заставил вас задуматься о множестве возможных способов использования виджетов Draggable и Droppable.

ПРИМЕЧАНИЕ

Итоговую версию кода данного руководства вы найдете в файле *готовый_12_02.html*.

Сортировка элементов страницы

Плагин jQuery UI также предусматривает виджет для работы со списками, например, со списками дел, музыкальными плейлистами, а также со списками внутри списков, вроде каталогов внутри папок. Виджет Sortable позволяет легко изменить порядок элементов в группе, просто перетащив элемент на новое место. Этот виджет может пригодиться, например, для управления списком воспроизведения, — пользователи могут создать свой собственный плейлист, перетащив композиции в список, а также изменить их порядок, перетащив элемент списка на новое место (рис. 12.6).

Виджет Sortable может применяться к любой сгруппированной коллекции элементов. Хотя этот виджет является естественным выбором для неупорядоченного или упорядоченного списка, вы также можете превратить в сортируемую группу коллекцию элементов `div`, абзацев или изображений.

Использование виджета Sortable

Виджеты Sortable представляют собой набор элементов, которые можно перетаскивать на другое место в группе: например, в списке дел каждый элемент можно перетащить на другую позицию в списке. Таким образом, список является виджетом Sortable, а пункты списка являются элементами, которые можно сортировать. Другими словами, виджет Sortable должен представлять собой контейнер, например, неупорядоченный список (`ul`), упорядоченный список (`ol`) или элемент `div`, содержащий другие элементы, например, дополнительные элементы `div`, абзацев или изображений.

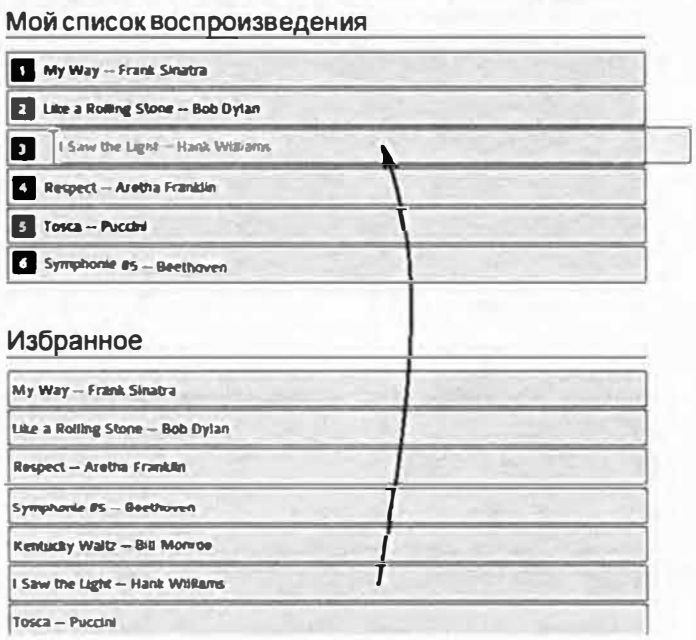

Рис. 12.6. Виджет jQuery UI Sortable позволяет легко переупорядочить пункты списка. Перетащите элементы из одного списка, например, из списка «Избранное» в список «Мой список воспроизведения». Такой тип взаимодействия часто применяется во многих компьютерных программах, например, iTunes, или в каталогах с файлами

Использовать виджет Sortable очень просто:

1. **Следуйте инструкциям, приведенным в разделе «Добавление плагина jQuery UI на веб-страницу» главы, чтобы прикрепить файлы CSS и JavaScript.**

   Помните, что у плагина jQuery UI есть свои собственные файлы CSS и JavaScript, и что ссылка на JavaScript-файл jQuery UI должна следовать после ссылки на JavaScript-файл библиотеки jQuery.

2. **Добавьте на веб-страницу элемент-контейнер.**

   Это может быть неупорядоченный список или элемент `div`:

   ```
   <ul id="playlist">

   </ul>
   ```

   Имеет смысл добавить для этого элемента идентификатор или класс, чтобы иметь возможность выбрать этот элемент с помощью jQuery.

3. **Добавьте элементы в контейнер**.

   В случае с неупорядоченным или упорядоченным списком это будет серия элементов `li`. Элементы, содержащиеся в контейнере, можно перетаскивать на новое место в списке. В элемент `div` вы можете

поместить изображения, абзацы или другие элементы div, которые станут сортируемыми элементами виджета:

```
<ul id="playlist">
<li>My Way -- Frank Sinatra</li>
<li>Like a Rolling Stone -- Bob Dylan</li>
<li>Respect -- Aretha Franklin</li>
</ul>
```

4. **Добавьте jQuery-функцию `$(document).ready()` на вашу страницу или во внешний JavaScript-файл:**

```
$(document).ready(function() {
}); // окончание ready
```

Как уже говорилось в разделе «Больше концепций для событий jQuery» главы 5, этот шаг необходим только в том случае, если вы помещаете свой JavaScript-код в раздел заголовка страницы перед большей частью кода HTML. Некоторые программисты помещают свой код JavaScript в нижней части страницы перед закрывающим тегом </body>. В этом случае вы можете пропустить функцию `$(document).ready()`.

5. **Используйте jQuery для выбора элемента-контейнера, добавленного в шаге 2:**

```
$(document).ready(function() {
$('#playlist').sortable();
}); // окончание ready
```

Этот фрагмент кода выбирает элемент-контейнер (тег <ul>, представленный кодом в шаге 3) и делает все содержащиеся в нем элементы сортируемыми. Вы можете настроить виджет Sortable с помощью многочисленных параметров, описанных далее.

6. **Добавьте параметры в функцию `sortable()`:**

```
$(document).ready(function() {
$('#playlist').sortable({
opacity : 0.5,
placeholder : 'ui-state-highlight'
});
}); // окончание ready
```

О различных параметрах данного виджета вы узнаете чуть позже, а пока кратко разберемся в том, что делает приведенный код. Во-первых, параметр `opacity` заставляет плагин jQuery UI уменьшить степень непрозрачности элемента в процессе его перетаскивания. Второй параметр, `placeholder`, заставляет плагин jQuery UI применить стиль к пустому месту, на которое может быть брошен перетаскиваемый элемент.

ПРИМЕЧАНИЕ

Вы опробуете виджет Sortable в процессе создания веб-приложения в главе 14.

Параметры виджета Sortable

Виджет jQuery UI Sortable предусматривает множество параметров. Вы можете контролировать направление и расстояние перемещаемого сортируемого элемента, его поведение при перемещении, а также определить часть элемента, за которую вы можете перетащить его. Как и в случае с другими виджетами плагина jQuery UI, вы задаете параметры, передавая объект функции `sortable()`. Например, чтобы указатель мыши принял вид указательного пальца, а элемент `h2` внутри объекта стал маркером для перетаскивания, вы передаете объект со следующими двумя параметрами:

```
$('#playlist).sortable({
cursor : 'pointer',
handle : 'h2'
});
```

Далее описаны некоторые из наиболее часто используемых параметров:

- **`axis`**. С помощью свойства `axis` вы можете ограничить перемещение сортируемого элемента только направлением слева направо или сверху вниз. Например, вы создали горизонтальную группу элементов `div`, которую пользователь может переупорядочить в процессе игры (см. рис. 12.6). Вы можете сделать так, чтобы эти элементы перемещались только слева направо. Используйте значение `'x'` (чтобы задать направление слева направо) или `'y'` (чтобы задать направление сверху вниз):

  ```
  axis : 'x'
  ```

- **cancel**. Данный параметр позволяет предотвратить перетаскивание объекта при щелчке по конкретному элементу. Например, у вас есть список воспроизведения (неупорядоченный список). Рядом с названием каждой песни есть значок в виде мусорной корзины, по которому пользователи могут щелкнуть, чтобы удалить песню из списка. Однако если пользователь захватит этот значок, то он сможет перетащить песню в новое место в списке вместо ее удаления. Чтобы элемент внутри сортируемого виджета не мог использоваться в качестве маркера перемещения, передайте селектор параметру `cancel`:

  ```
  cancel : '.trashcan'
  ```

  Теперь, если пользователь щелкнет по значку в виде мусорной корзины (при условии, что этот значок имеет класс `trashcan`), он не сможет инициировать перетаскивание. Вы также можете указать несколько элементов, разделив их запятыми:

  ```
  cancel : '.trashcan, .addToFavorites'
  ```

  Параметр `handle`, описанный далее, позволяет задать конкретный элемент в качестве маркера перетаскивания.

- **connectWith**. Позволяет указать селектор других сортируемых элементов (другого списка), в которые вы можете перетащить элемент. Например, у вас на странице есть два списка: список пожеланий, содержащий товары, которые вы хотели бы купить, и список товаров, помещенных в корзину, которые вы намерены приобрести. Вы можете решить переупорядочить список пожеланий, например, переместить самые желанные товары в верхнюю часть списка, и при этом иметь возможность перетащить элемент из списка пожеланий в список товаров, находящихся в корзине. В этом случае у вас будут два разных списка, к каждому из которых применена функция `sortable()`. Тем не менее с помощью параметра `connectWith` вы можете позволить пользователям перетащить элемент из одного списка в другой:

  ```
  $('#wishList').sortable({
  connectWith : '#shoppingCart'
  });
  ```

  Параметр `connectWith` работает только в одном направлении. Другими словами, приведенный выше код позволяет перетащить элементы из списка пожеланий в корзину, но не наоборот. Для соеди-

нения обоих списков вам нужно применить параметр `connectWith` к каждому из них:

```
$('#shoppingCart').sortable({
connectWith : '#wishList'
});
```

Значение, которое вы передаете параметру `connectWith`, должно представлять собой селектор, соответствующий элементу, к которому вы применили функцию `sortable()`.

- **`containment`**. Вы можете предотвратить возможность перетаскивания элемента за пределы элемента-контейнера. Этот параметр работает аналогично одноименному параметру виджета Draggable (см. раздел «Параметры виджета Draggable» ранее в главе). На самом деле сортируемый элемент является просто перетаскиваемым объектом, который вы можете поместить в список элементов. Параметр `containment` может принимать несколько значений:

- **Селектор.** Если вы укажете имя селектора, то плагин jQuery UI будет удерживать пункт списка в границах соответствующего элемента. Например, если у вас на странице есть элемент `div` с идентификатором `mainContent`, то вы можете удерживать перетаскиваемый элемент внутри этого элемента `div`, настроив параметр `containment` следующим образом:

```
containment : '#mainContent'
```

- **`parent`, `document` или `window`**. Чтобы удерживать пункт списка внутри его родительского элемента, используйте в качестве значения `parent`. Например, если вы хотите, чтобы элемент неупорядоченного списка можно было перетаскивать только в пределах этого списка, используйте следующий код:

```
containment : 'parent'
```

Значения `document` и `window` почти одинаковы, за исключением того, что значение `document` удерживает всю перетаскиваемую область в пределах области документа, а значение `window` позволяет частично перетащить элемент за пределы окна (это выглядит не очень хорошо, так что подумайте дважды, прежде чем использовать значение `window`).

- **`cursor`**. Соответствует параметру `cursor` для перетаскиваемых элементов (см. раздел «Параметры виджета Draggable» ранее в главе).

- **cursorAt**. Соответствует параметру `cursorAt` для перетаскиваемых элементов (см. раздел «Параметры виджета Draggable» ранее в главе).
- **delay**. Продолжительность задержки при перетаскивании сортируемого элемента (в миллисекундах). Такая задержка может пригодиться, если вы считаете, что при перемещении указателя мыши по странице можно случайно перетащить элемент списка. Значение данного параметра определяет, сколько времени посетитель должен удерживать нажатой кнопку мыши, прежде чем у него появится возможность перетащить элемент:

```
delay : 100
```

- **distance**. Расстояние в пикселах, на которое необходимо переместить элемент списка до начала сортировки. Этот параметр может пригодиться в тех случаях, когда посетителям необходимо щелкнуть по сортируемому элементу с целью, отличной от перетаскивания. Например, если сортируемый элемент предусматривает кнопку для его удаления, то вы можете установить параметр `distance`, чтобы предотвратить перетаскивание элемента, когда посетитель случайно перемещает указатель мыши после щелчка по кнопке для удаления. Используйте небольшое численное значение для данного параметра (в противном случае посетителям придется перетащить элемент на достаточно большое расстояние, прежде чем произойдет сортировка):

```
distance : 10
```

- **grid**. Позволяет привязать элемент списка к сетке. Данный параметр хорошо работает, если у вас есть группа одинаковых по размеру фотографий или элементов `div`: вы можете сделать так, чтобы при перетаскивании элемента списка он перемещался на расстояние, точно соответствующее ширине остальных элементов списка. Данный параметр работает аналогично одноименному параметру для виджета Draggable (см. раздел «Параметры виджета Draggable» ранее в главе).
- **handle**. Определяет часть элемента, по которой пользователь должен щелкнуть, чтобы перетащить элемент на другое место в списке. Как правило, пользователи могут перетаскивать объекты, щелкнув в любой точке сортируемого элемента. Тем не менее вы можете решить, что пользователи должны щелкнуть по определенной части элемента, например, по вкладке. Этот параметр работает аналогично

одноименному параметру виджета Draggable (см. раздел «Параметры виджета Draggable» ранее в главе).

- **items.** Вы можете определить, какие элементы сортируемой группы могут быть отсортированы. Например, вы применили функцию `sortable()` к многоуровнему списку, то есть к такому, который содержит подсписки (рис. 12.7). Вам не нужно, чтобы список верхнего уровня был сортируемым (на рис. 12.7 элементами списка верхнего уровня являются Папка А и Папка Б). Вам нужно, чтобы сортируемыми элементами оказались пункты вложенного списка, что позволит поместить их в другие вложенные списки или переупорядочить в текущем вложенном списке.

  В этом случае вы можете использовать параметр `items` со значением `'li li'`, которое обеспечивает возможность сортировки только пунктов списка, находящегося в другом списке (то есть пунктов вложенного списка). Списки верхнего уровня не являются вложенными, поэтому они не могут быть отсортированы:

  ```
  items : 'li li'
  ```

- **opacity**. При перетаскивании элемента вы можете изменить степень его прозрачности. Например, вы можете сделать элемент на 50% непрозрачным, чтобы придать ему «призрачный» вид. Изменение степени непрозрачности является распространенным способом показать, что элемент перемещается с одного места на другое. Используйте значение от 0 (полностью невидимый) до 1 (полностью непрозрачный). Этот параметр работает аналогично свойству CSS `opacity`. Например, чтобы сделать перетаскиваемый элемент на 50% прозрачным, установите значение `0.5`:

  ```
  opacity : 0.5
  ```

- **placeholder**. Имя класса, который плагин jQuery UI применяет к пустому месту, на которое может быть перемещен пункт списка. Вы можете использовать один из собственных классов jQuery UI (см. раздел «Как плагин jQuery UI стилизует виджеты» главы 11), чтобы выделить это место:

  ```
  placeholder : 'ui-state-highlight'
  ```

ПРИМЕЧАНИЕ

О других параметрах виджета Sortable вы можете прочитать на странице: **api.jqueryui.com/sortable/**.

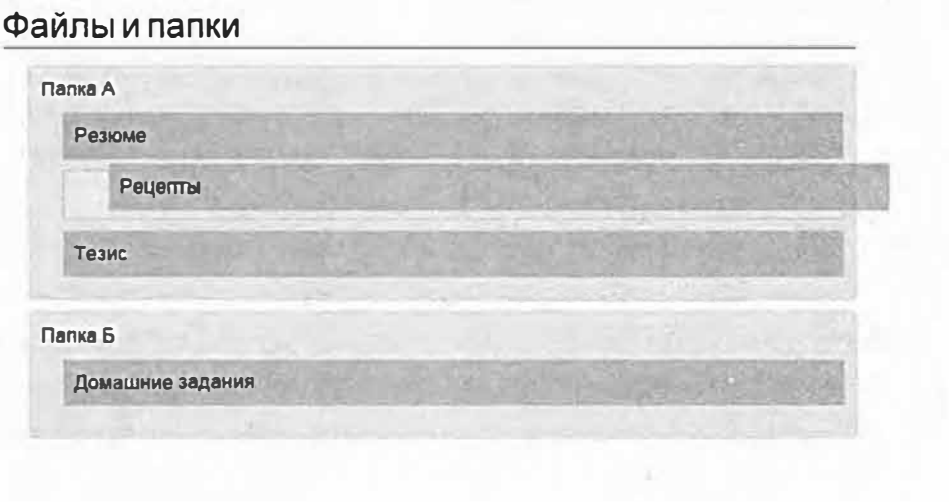

Рис. 12.7. Вы можете создавать вложенные списки (то есть списки внутри списков) в коде HTML путем добавления элемента неупорядоченного (`ul`) или упорядоченного (`ol`) списка внутри пункта списка: `<li>Папка A<ul><li>Подсписок</li></ul></li>`. Плагин jQuery UI позволяет определить, какие элементы внутри виджета Sortable могут быть перемещены и отсортированы, поэтому вы можете сделать так, чтобы сортировались только элементы внутри вложенных списков

События виджета Sortable

Во время взаимодействия посетителя с сортируемым списком плагин jQuery UI генерирует всевозможные события, например, когда посетитель начинает и заканчивает процесс перетаскивания элемента списка. Плагин jQuery UI предусматривает более десятка различных событий виджета Sortable, на каждое из которых вы можете отреагировать с помощью собственного программного кода. Некоторые из этих событий генерируются так близко друг от друга, что их практически невозможно различить.

Как и в случае с виджетами Draggable и Droppable, вы назначаете функции событиям виджета Sortable с помощью объекта с параметрами, переданного функции `sortable()`. Например, для отображения окна с оповещением после перемещения пункта списка вы можете написать следующий код:

```
$('#playList').sortable({
stop : function (event, ui) {
alert('Отличная работа, мой друг!');
}
});
```

Некоторые события применяются ко всем спискам, а два применяются только тогда, когда на одной странице находятся два (или более) сортируемых списка, а также в процессе перетаскивания элементов между списками. Предположим, вы начинаете с событий, которые применяются ко всем сортируемым виджетам. Эти события срабатывают в определенном порядке, так что вы будете перечислять их следующим образом:

- **`create`**. Это событие генерируется каждый раз, когда вы используете функцию `sortable()` для создания нового сортируемого списка. Вы можете использовать это событие для отображения диалогового окна с инструкцией: «Перетащите композиции, чтобы переупорядочить свой список воспроизведения». Это событие срабатывает только один раз — при создании виджета Sortable.

ПРИМЕЧАНИЕ

В отличие от остальных событий виджета Sortable, событие `create` не принимает объект `ui` (см. раздел «Событие `create`» ранее в этой главе).

- **`start`**. Генерируется, когда пользователь начинает перетаскивать элемент списка. Функция, которую вы назначаете этому событию, принимает два аргумента: объект `event` (см. раздел «Объект события» главы 5) и объект `ui`, который содержит информацию о виджете. Объект `ui` содержит семь других объектов, каждый из которых предоставляет важную информацию о виджете.
- **`ui.helper`**. Объект `helper` является объектом jQuery, который представляет собой элемент, перемещаемый пользователем. Плагин jQuery UI создает клон исходного элемента, поэтому при перетаскивании сортируемого объекта задействуются два различных элемента HTML-кода: вспомогательный клон и фактический элемент, который является частью списка. По окончании перемещения вспомогательный элемент удаляется из документа. Поскольку `helper` является обычным объектом jQuery, вы можете применить к нему такие методы jQuery, как `css()`,`animate()` или `find()`.
- **`ui.item`**. Представляет фактический сортируемый элемент, по которому посетитель щелкнул, чтобы перетащить, например, элемент `li`. Это реальный элемент HTML-кода, и он будут помещен в надлежащее место в списке, как только пользователь закончит перетаскивать вспомогательный элемент. Этот элемент также является

объектом jQuery, поэтому к нему применимы все обычные методы этой библиотеки.

- **ui.position** предоставляет x и y координаты левого верхнего угла вспомогательного элемента (элемента, который визуально перетаскивается по экрану) относительно его ближайшего позиционированного предка. Если виджет Sortable находится внутри относительно или абсолютно позиционированного элемента, то свойство ui.position предоставит координаты его верхнего левого угла относительно верхнего левого угла позиционированного элемента.

  Вы можете получить доступ к этим значениям, используя фрагменты кода: ui.position.top и ui.position.left.

- **ui.originalPosition**. Исходное положение сортируемого элемента, то есть место, где он находился, прежде чем пользователь начал его перемещать. Это такой же объект, как ui.position, с двумя свойствами: top и left.

- **ui.offset** — это объект с двумя свойствами: top и left. Однако в данном случае позиция определяется относительно левого верхнего угла окна браузера. Свойство ui.offset.top указывает, как далеко верхняя часть вспомогательного элемента находится от верхней части окна.

  Свойство ui.offset.left определяет, на сколько пикселов перетаскиваемый объект отстоит от левого края окна браузера.

- **ui.sender**. Данное свойство применяется только при перетаскивании элемента из одного сортируемого виджета в другой. Свойство ui.sender содержит объект jQuery для сортируемого виджета, из которого перетаскивается элемент.

- **placeholder**. Объект jQuery, представляющий пустое место заполнителя, создаваемое при перетаскивании сортируемого элемента.

  Объект ui доступен для всех других обработчиков событий виджета Sortable — activate, over, sort и т. д., кроме события create.

- **activate**. Срабатывает непосредственно после событие start, но практически одновременно с ним. Вы можете использовать это событие, чтобы добавить вторую функцию, которую вы хотите запустить после функции, назначенной событию start.

- **sort**. Событие sort срабатывает при каждом перемещении указателя мыши в процессе перетаскивания сортируемого элемента.

Другими словами, это событие срабатывает постоянно, поэтому не назначайте для него задач, требующих много времени или вычислительных ресурсов. В противном случае многочисленные вызовы функции приведут к снижению отзывчивости страницы и, вероятно, усложнят процесс перетаскивания сортируемого виджета для ваших посетителей.

- **change**. Данное событие срабатывает, как только сортируемые элементы перемещаются на новое место. Когда элемент перетаскивается с первой позиции списка вниз, а второй пункт перескакивает на его место, срабатывает событие `change`.

  Вы можете использовать это событие, например, для выделения двух пунктов списка, которые поменялись местами.

- **beforestop**. Событие `beforestop` срабатывает после того как пользователь отпускает сортируемый элемент. Это последнее событие, имеющее доступ к объекту `ui.helper`. После срабатывания данного события объект `ui.helper` (клон перетаскиваемого элемента) удаляется.

- **update**. После размещения сортируемых элементов по местам и обновления DOM срабатывает событие `update`.

- **deactivate**. Когда сортировка закончена, после события `update` срабатывает событие `deactivate`.

- **stop**. Когда сортируемый элемент будет помещен на новую позицию, сработает событие `stop`. Это событие является последним и всегда следует после событий `stop` и `deactivate`. Назначьте этому событию функцию, которая соответствует последнему действию, которое должно быть выполнено над элементом, после того, как вы расположите его на новом месте. Например, вы можете использовать событие `stop`, чтобы проверить, соответствует ли состояние списка некоторому заранее определенному порядку (см. рис. 12.7).

  Описанные выше события срабатывают в порядке, в котором они перечислены. Тем не менее существует несколько событий, которые запускаются в другое время и для других типов виджетов Sortable.

- **out**. Событие `out` срабатывает, когда сортируемый элемент перемещается за пределы виджета Sortable, например, если вы перетаскиваете элемент из соответствующего виджета Sortable на пустое место на странице. Данное событие также срабатывает при перемещении объекта из одного виджета в другой.

- **over**. Срабатывает при перемещении сортируемого элемента над связанным списком. Например, если у вас есть два связанных списка, и вы перетаскиваете элемент из первого списка во второй, то на втором списке срабатывает событие `over`. Это событие также срабатывает, когда вы перемещаете элемент из группы сортируемых элементов, а затем обратно.
- **receive**. Используйте событие `receive` при использовании нескольких соединенных виджетов (см. описание параметра `connectWith` в разделе «Параметры виджета Sortable» ранее в главе). Когда виджет Sortable получает элемент из другого виджета, срабатывается событие `receive`. Вы можете использовать это событие в сочетании с объектом `ui.sender` (см. выше), чтобы определить, откуда взялся новый элемент.
- **remove**. Это событие срабатывает, когда элемент удаляется из текущего сортируемого виджета. Например, у вас есть два виджета: список пожеланий и корзина. Если покупатель перетащил элемент из списка пожеланий в корзину, то на списке пожеланий сработает событие `remove` (а также событие `out`). (В этом сценарии события `over` и `receive` также сработают на корзине.)

Методы виджета Sortable

Виджет Sortable предусматривает несколько различных методов или функций, которые можно к нему применить. Вы не будете вызывать эти методы, как обычную jQuery-функцию, например, для выбора элемента страницы и применения к нему метода: `$('body').hide()`. Вместо этого вы предоставляете функции `sortable()` имя метода в виде строки. Допустим, вы используете метод `destroy`, который полностью удаляет виджет Sortable — он превращает сортируемый набор элементов списка в обычный список элементов, которые нельзя перетаскивать. Код для вызова метода `destroy` выглядит следующим образом:

```
$('#sortableItems').sortable('destroy');
```

Селектор, в данном примере `#sortableItems`, должен соответствовать селектору присутствующего на странице виджета Sortable. Вы часто будете использовать эти методы в ответ на срабатывание одного из событий виджета Sortable, описанных в предыдущем разделе. Например, если у вас есть игра, которая требует от игрока поместить набор блоков в определенном порядке, то вы могли бы уничтожить сортируе-

мый виджет после того, как игрок завершит игру, чтобы он не мог снова двигать блоки.

Вы можете найти полный список методов, применяемых к виджету Sortable на странице: **api.jqueryui.com/sortable/**, а далее описаны некоторые из наиболее полезных функций:

- **`cancel`**. Этот метод отменяет любые изменения в текущем порядке сортируемого списка. Другими словами, он прекращает процесс переупорядочивания этого списка. Вы можете использовать эту функцию совместно с методом `receive` (см. выше), чтобы отклонить элемент списка, перемещенный в другой виджет Sortable. Кроме того, в качестве события `stop` (см. выше) вы можете создать функцию, которая проверяет местоположение добавленного элемента и отвергает изменение, если этот элемент не отвечает определенным условиям. Например, в приложении для создания списка дел вы можете сделать так, чтобы определенные задачи зависели от других задач. Например, если пользователь пытается перетащить задачу в верхнюю часть списка, но первой должна быть выполнена другая задача, то вы можете отменить действие этого пользователя и отобразить диалоговое окно с объяснением причины отмены.
- **`destroy`**. Чтобы полностью удалить возможность сортировки виджета, используйте метод `destroy`.
- **`disable`**. Для временного отключения возможности сортировки виджета вызовите метод `disable`. Используйте его, если вам нужно запретить сортировку до возникновения некоторого условия. Вы можете включить возможность сортировки, используя метод `enable` (описан далее).
- **`enable`**. Включите возможность сортировки виджета, к которому ранее был применен метод `disable`.
- **`serialize`**. Этот метод полезен для отправки данных о порядке следования элементов в списке обратно на веб-сервер с помощью технологии Ajax или передачи формы. Вам может потребоваться это сделать при необходимости хранения этой информации на сервере для последующего использования. Например, если менеджер упорядочивает список задач для сотрудников, то информация об их порядке может быть отправлена на сервер, храниться в базе данных и извлекаться, когда сотрудник зайдет в систему, чтобы увидеть, какие задачи (и в каком порядке) должны быть выполнены. Вам необходимо особым образом отформатировать сортируемые элементы:

- Каждый элемент списка должен иметь идентификатор.
- Каждый идентификатор должен начинаться со слова, используемого в качестве идентификатора группы для списка, за которым следует символ нижнего подчеркивания (_).
- За символом _ в имени идентификатора должен следовать уникальный идентификатор для элемента списка.

Например, у вас есть список воспроизведения. HTML-код может выглядеть следующим образом:

```
<ul id="playlist">
<li id="song_1">My Way -- Frank Sinatra</li>
<li id="song_2">Like a Rolling Stone -- Bob Dylan</li>
<li id="song_3">Respect -- Aretha Franklin</li> </ul>
```

Обратите внимание на то, что идентификатор для каждого элемента списка начинается с фрагмента `"song_"` и заканчивается уникальным числом. Это необязательно должно быть число, это может быть слово или уникальный идентификатор из базы данных. Идея состоит в том, чтобы обеспечить способ передачи серверу данных о порядке следования элементов списка.

Вы можете использовать метод `serialize` для сохранения текущего порядка элементов списка:

```
var listOrder = $("#playlist").sortable('serialize');
```

Метод `serialize` возвращает строку, которая выглядит примерно так:

```
song[]=2&song[]=3&song[]=1
```

Эта строка сообщает порядок следования элементов в списке. В данном случае числа 2, 3 и 1 указывают на то, что первый элемент списка был перемещен в его конец. Вы можете пометить строковое значение, возвращенное методом `serialize` URL-адресу, как часть запроса Ajax (глава 13).

ПРИМЕЧАНИЕ

Метод `serialize` позволяет предоставить параметры, определяющие формат строки, возвращаемой данным методом. Об этих параметрах вы можете прочитать на странице: **api.jqueryui.com/sortable/#method-serialize**.

- **toArray**. Подобно методу `serialize`, метод `toArray` является способом извлечения упорядоченного списка элементов виджета Sortable.

  Чтобы его использовать, необходимо добавить идентификатор для каждого пункта сортируемого списка. Однако в отличие от метода `serialize`, он не требует использования какого-либо специального формата. Этот метод просто возвращает массив с перечнем идентификаторов всех элементов виджета Sortable в порядке их следования в списке.

  Например, у вас есть виджет Sortable с идентификатором `colorList`. Внутри него находятся три элемента списка с разными цветами:

```
<ul id="colorList">
<li id="red">Красный</li>
<li id="green">Зеленый</li>
<li id="blue">Синий</li>
</ul>
```

Допустим, пользователь изменил порядок следования цветов, поэтому «Синий» теперь идет первым, «Красный» — вторым, а «Зеленый» — последним. В этом случае вы можете использовать метод `toArray` следующим образом:

```
var colors = $('#colorList').sortable('toArray');
```

Переменная `colors` будет содержать такой массив:

```
['blue', 'red', 'green']
```

Результатом является простой массив JavaScript (см. раздел «Массивы» главы 2). Вы можете применить к нему любой из методов, описанных в разделе, посвященном массивам.

Метод `toArray` может применяться, например, для проверки соответствия порядка элементов в списке некоторому предопределенному порядку. Например, вы создали игру, которая отображает список цветных блоков в случайном порядке.

Игрок должен расположить блоки в порядке цветов радуги. Вы можете назначить функцию событию `stop` (см. выше), которое извлекает текущий порядок следования блоков с помощью метода `toArray` и затем сравнивает его с другим массивом, содержащим ответ (рис. 12.8).

ПРИМЕЧАНИЕ

Плагин jQuery UI включает в себя еще два взаимодействия. Виджет Resizable используется виджетом Dialog (см. раздел «Создание диалоговых окон с сообщениями» главы 9), чтобы пользователи могли изменять размер диалогового окна, перетаскивая маркер масштабирования в углу. Вы можете использовать этот виджет для создания масштабируемых плавающих окон. Подробное описание этого виджета приведено на сайте: **api.jqueryui.com/resizable/**.

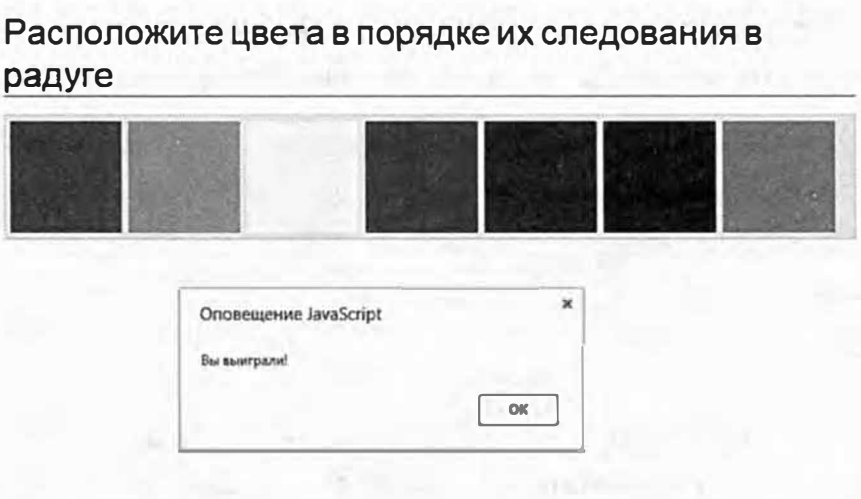

Рис. 12.8. Используйте метод `toArray` виджета Sortable для извлечения массива с идентификаторами элементов в том порядке, в котором они в настоящее время располагаются в списке. Вы можете использовать эту информацию для сравнения текущего порядка с некоторым предопределенным результатом. В этой игре при каждом перемещении блока порядок их следования сравнивается с правильным ответом. Если этот порядок совпадает с ответом, игрок выигрывает. Вы найдете готовый пример в файле *радуга.html* в папке *глава 12*

ПРИМЕЧАНИЕ

Виджет Selectable позволяет посетителю выбирать элементы (выделять их), щелкнув по ним кнопкой мыши. Вы можете использовать его на странице загрузки фотографий «Выберите фотографии, которые вы хотели бы загрузить». Подробное описание данного виджета вы найдете по адресу: **api.jqueryui.com/selectable/**.

Эффекты jQuery UI

Плагин jQuery UI включает набор визуальных анимационных эффектов, которые можно использовать для оживления веб-приложений. Например, эффект взрыва, который вы использовали в руководстве

«Drag-and-Drop на практике» ранее в этой главе, заставляет элемент разваливаться и исчезать из вида. Далее вы подробно узнаете об этих эффектах, однако сначала вам нужно знать, как их применять.

Эффекты предназначены для того, чтобы отобразить, скрыть или выделить элемент, например, заставить его загораться или дрожать. Для применения эффектов вы используете либо jQuery-функции, как раньше, либо функцию jQuery UI `effect()`. Например, чтобы использовать эффект `drop` плагина jQuery UI, при котором элемент как бы «падает» на страницу, вы можете применить метод `show()` с именем эффекта и значением продолжительности:

```
$('#pageElement').show('drop', 1000);
```

Этот код заставляет элемент «падать» на страницу в течение 1 секунды (1000 миллисекунд). Метод `show()` должен быть вам знаком (см. раздел «Основы отображения и сокрытия» главы 6). Это jQuery-функция, которая отображает скрытый на странице элемент. Тем не менее плагин jQuery UI добавляет некоторые новые возможности в jQuery-функцию `show()`, а также в две другие jQuery-функции. Плагин jQuery UI предусматривает четыре разные функции для добавления эффектов к элементам:

- **`show()`**.Плагин jQuery UI расширяет возможности функции `show()`, позволяя использовать один из 15 различных эффектов для отображения скрытого элемента. Вы должны убедиться в том, что элемент скрыт (см. описание метода `hide()` в разделе «Основы отображения и сокрытия» главы 6), прежде чем использовать функцию `show()`. В противном случае, элемент быстро исчезнет, а затем вновь отобразится.
- **`hide()`**.Чтобы применить один из эффектов jQuery UI при сокрытии элемента, используйте метод `hide()`. Он работает подобно одноименной jQuery-функции (см. раздел «Основы отображения и сокрытия» главы 6), то есть заставляет элемент исчезать со страницы, но с применением специального эффекта. Прежде чем использовать эту функцию, убедитесь в том, что элемент отображен на странице.
- **`toggle()`**.Метод `toggle()` отображает или скрывает элемент. Если элемент скрыт, то метод `toggle()` заставляет его появиться с использованием выбранного эффекта. Если элемент видим, то метод `toggle()` скрывает его.
- **`effect()`**.Большинство эффектов плагина jQuery UI создано с целью обеспечения зрелищного отображения или сокрытия элемента. Некоторые эффекты, например, `bounce`, `highlight`, `pulsate`

и shake могут быть использованы для выделения уже видимого элемента, не заставляя его исчезать. Эффект highlight, например, на мгновение выделяет элемент с помощью яркого цвета, что позволяет привлечь внимание посетителя к определенному месту на странице. Функция effect() является уникальной для плагина jQuery UI и не является стандартной функцией библиотеки jQuery.

ПРИМЕЧАНИЕ

Вы примените функцию effect() в веб-приложении, которое вы будете создавать в главе 14.

Эффекты

Эффекты плагина jQuery UI позволяют добавить мощные анимационные инструменты в ваш инструментарий веб-дизайнера. Вы можете передать каждый из этих эффектов в качестве аргумента перечисленным выше функциям. Каждый эффект создает особый визуальный результат. Базовая структура использования эффекта с одной из указанных выше функций выглядит следующим образом:

```
$('#element').hide('effectName', { optionName : ↵
optionValue }, duration, callBackFunction);
```

Например, если вы хотите скрыть находящийся на странице элемент с идентификатором deleteThis, заставив его разорваться на 16 фрагментов за полсекунды, а затем отобразить окно оповещения со словом «Бах!», вы можете использовать следующий код:

```
$('#deleteThis').hide('explode', { pieces : 16 },↵
500, function () {
alert('Бах!');
});
```

Большая часть эффектов предусматривает возможность передачи одного или нескольких дополнительных параметров, контролирующих принцип работы эффекта. Эти параметры передаются в качестве объекта, состоящего из имен и значений. В предыдущем примере параметром является количество фрагментов, на которые должен разделить элемент эффект explode:

```
{ pieces : 16 }
```

Некоторые эффекты имеют по нескольку параметров, а некоторые — вообще их не имеют. Плагин jQuery UI предусматривает множество эффектов:

- **blind**. Эффект `blind` отображает или скрывает элемент, имитируя поднятие или опускания шторки. Данный эффект принимает один параметр — `direction`, указывающий направление движения шторки: `up` (вверх), `down` (вниз), `left` (влево) или `right` (вправо). Вы передаете этот параметр в качестве объекта — имени параметра, а значение — в качестве второго аргумента функции. Например, чтобы заставить определенный элемент исчезнуть с экрана, вы можете использовать следующий код:

```
$('#element').hide('blind', { direction : 'left'},↵
1000);
```

- **bounce**. Эффект `bounce` можно применить при сокрытии или отображении элемента. Вы также можете использовать эффект `bounce` с функцией `effect()`, чтобы видимый элемент просто отскакивал вверх и вниз, как бы говоря: «Посмотрите на меня!!!» Данный эффект предусматривает два параметра:
  - **distance**. Максимальное расстояние (в пикселах), на которое перемещается элемент при отскоке. Чем больше число, тем дальше он отскакивает и тем больше внимание привлекает.
  - **times**. Число отскоков, совершаемых элементом.

  Например, чтобы элемент совершал 20 отскоков на расстояние 100 пикселов при каждом щелчке по нему, вы можете использовать следующий код:

```
$('#theElement').click(function () {
$(this).effect('bounce', {
distance : 100,
times : 20
},
1000
);
});
```

- **clip**. Эффект `clip` применяет атрибут CSS `clip` к элементу, чтобы он отображался вертикально или горизонтально. Этот эффект при-

нимает только один параметр — `direction` со значением `vertical` или `horizontal`:

```
{ direction : 'horizontal' }
```

- **drop**. Эффект `drop` постепенно отображает или скрывает элемент, сдвигая его вверх, вниз, влево или вправо. Он предусматривает единственный параметр — `direction`, который принимает одно из следующих четырех значений: `up`, `down`, `left` или `right`.

- **explode**. Эффект `explode` разделяет элемент на несколько фрагментов, разлетающихся в разные стороны и исчезающих из вида. При использовании с функцией `show()` взрыв происходит в обратном направлении, то есть элемент собирается из отдельных фрагментов. Этот эффект предусматривает только один параметр — `pieces`, определяющий количество фрагментов, на которые должен быть разделен элемент. Это количество должно являться квадратом целого числа: 1, 4, 9, 16, 25 и т. д. (однако не следует задавать число больше 25, чтобы не замедлять работу эффекта):

```
{ pieces : 16 }
```

- **fade**. Работает подобно функциям jQuery `fadeIn()` и `fadeOut()`. Другими словами, это не самый впечатляющий эффект.

- **fold**. Эффект `fold` отображает или скрывает элемент, сворачивая его. Для этого интригующего визуального эффекта предусмотрено два параметра. Параметр `size` — это размер в пикселах (или процентах), до которого должен уменьшиться элемент перед сворачиванием. Параметр `horizFirst` принимает логическое значение, которое определяет, как элемент должен сворачиваться сначала — по вертикали (по умолчанию) или по горизонтали:

```
{ size : '50%', horizFirst : true }
```

- **highlight**. Эффект `highlight` моментально меняет цвет фона элемента, чтобы привлечь к нему внимание. Это еще один эффект, который вы можете применить к видимому элементу, не скрывая его. Вы можете изменить цвет фона с помощью параметра `color`. Например, чтобы выделить элемент, на 1 секунду изменив цвет фона на красный, вы можете применить метод `effect()` следующим образом:

```
$('#element').effect('highlight', { color :↵
'#ff0000' }, 1000);
```

- **puff**. Этот эффект заставляет элемент изменяться в размере и исчезать из вида (или проявляться). Он предусматривает единственный параметр `percent`, который определяет размер элемента при его отображении или сокрытии:

```
{ percent : 200 }
```

- **pulsate**. Еще один эффект, который вы можете использовать, чтобы привлечь к элементу внимание, не скрывая его. Этот эффект поочередно скрывает и отображает элемент. Вы можете указать, сколько раз элемент должен пульсировать, используя параметр `times`:

```
$('#element').effect('pulsate', {times : 20}, 2000);
```

- **scale**. Используйте этот эффект для масштабирования элемента. Вы можете применить этот эффект с методом `hide()` или `show()`. Обычно, если вы используете метод `show()`, чтобы отобразить скрытый элемент, то этот элемент увеличивается с размера небольшого пятна на странице до своего нормального размера. Если элемент является видимым, и вы используете метод `hide()`, то элемент уменьшится и исчезнет.

- **shake**. Этот эффект можно применить к элементу, отображенному на странице, используя функцию `effect()` (описана ранее в этом разделе). Этот эффект заставляет элемент трястись определенное количество раз в конкретном направлении и на определенном расстоянии. Это еще один из эффектов, позволяющих привлечь внимание к элементу, однако он может служить и более важной цели, например, указывать на ошибку: этот эффект может обеспечить дрожание диалогового окна, если посетитель не установил в форме регистрации флажок «С условиями использования ознакомлен». Этот эффект предусматривает три параметра. Параметр `direction` задает направление, в котором начинается дрожание элемента: `up`, `down`, `left` или `right`. Установите параметр `distance`, чтобы указать количество пикселов, на которое должен перемещаться элемент. Чем больше число, тем более заметно дрожание. Наконец, параметр `times` определяет, сколько раз должен дрожать элемент. Например, чтобы элемент дрожал 10 раз, перемещаясь на 50 пикселов влево (и продолжал дрожать в горизонтальном направлении), вы можете передать функции `effect()` объект, содержащий следующие параметры:

```
{ direction : 'left' , distance : 50, times : 10 }
```

- **size**. Этот эффект изменяет размер элемента до заданной ширины и высоты. Он принимает три параметра: `to` — объект со свойствами `width` и `height`, которые определяют ширину и высоту элемента при сокрытии или отображении элемента.

  Свойство `origin` определяет точку исчезновения при использовании эффекта `size` для скрывания элемента. Это массив, содержащий два числа: первое соответствует позиции `top`, а второе — позиции `left`.

  Свойство `scale` используется для указания того, что должно быть отмасштабировано. Допустимыми значениями являются `both`, `box` и `content`. Например, чтобы увеличить или уменьшить только размер внешней рамки (граница, фон, ширина и высота) используйте значение `box`. При этом содержимое элемента (например, текст) не масштабируется.

- **slide**. Этот эффект заставляет элемент скользить из области просмотра (при сокрытии) или в область просмотра (при отображении). Передайте параметр `direction` — `left`, `right`, `up` или `down`, чтобы задать направление скольжения:

```
$('#element').show('slide', { direction : 'right' },↵
1000 );
```

ПРИМЕЧАНИЕ

Чтобы подробнее узнать об эффектах jQueryUI, посетите страницу: **api.jqueryui.com/category/effects/**.

Параметр `easing`

Плагин jQuery UI предусматривает набор параметров `easing`, которые контролируют скорость изменений в процессе анимации. Параметр `easing` не изменяет общую продолжительность анимации, но изменяет скорость, с которой происходят ее отдельные фрагменты. Например, если вы используете эффект `bounce`, вы можете применить параметр `easing`, который обеспечивает медленное начало и дальнейшее ускорение отскока.

ПРИМЕЧАНИЕ

Посетите страницу jQuery UI Easings, чтобы ознакомиться с параметрами `easing`: **api.jqueryui.com/easings/**.

Для использования параметра `easing` вместе с эффектом вам необходимо передать его в качестве свойства объекта `options`. Названием свойства является `easing`, а значением является одна из возможных функций — `easeInOutQuart`, `easeInSine` и т. д. (полный список параметров `easing` можно найти на сайте **api.jqueryui.com/easings/**).

Например, вы хотите использовать эффект `bounce` (описан ранее в этом разделе), чтобы элемент отскакивал каждый раз, когда по нему щелкают кнопкой мыши. Вам нужно, чтобы элемент отскакивал 20 раз на максимальное расстояние в 100 пикселов (это два параметра эффекта `bounce`). Вы также хотите, чтобы эффект `easing` выглядел более реалистично, то есть чтобы передвижение элемента замедлялось по мере истощения энергии. В этом случае вы можете добавить параметр `easing` к объекту `options` следующим образом:

```
$('#theElement').click(function () {
$(this).effect('bounce', {
distance : 100,
times : 20,
easing : 'easeOutBounce'
},
1000
);
});
```

Вам необязательно использовать параметры `easing` только с эффектами jQuery UI. Как говорилось в разделе «Управление скоростью анимации» главы 6, вы можете использовать эти параметры также с анимацией jQuery UI.

Анимация изменения класса

Плагин jQuery UI включает несколько функций, которые позволяют анимировать изменения значений свойств CSS элемента при добавлении или удалении класса. Эти функции фактически являются продолжением существующих jQuery-функций: `addClass()`, `toggleClass()` и `removeClass()` (см. раздел «Установка и чтение атрибутов элемента» главы 4). Плагин jQuery UI просто анимирует изменения, поэтому при добавлении или удалении класса jQuery UI анимирует визуальные изменения, которые происходят из-за добавления или удаления свойств CSS.

ПРИМЕЧАНИЕ

Анимация изменения класса jQuery UI может походить на переходы CSS. Они на самом деле очень похожи: в то время как переходы CSS зависят от встроенного в браузер движка для рендеринга CSS, jQuery использует для анимации изменений JavaScript. Переходы CSS являются лучшим вариантом, поскольку они работают более плавно и не требуют какой-либо работы от интерпретатора JavaScript. Тем не менее переходы CSS не поддерживаются браузером Internet Explorer 9 или более ранней версией, поэтому вам может понадобиться использовать функцию анимации изменения класса jQuery UI, чтобы обеспечить тот же эффект в этих браузерах. Однако если вам не нужно поддерживать устаревшие версии Internet Explorer, используйте переходы CSS вместо этих функций jQuery UI. Чтобы подробнее узнать о переходах CSS, посетите страницу: **developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_transitions**.

Каждая из функций анимации изменения класса jQuery UI принимает до четырех аргументов, контролирующих принцип их работы. Первый аргумент — это имя класса, который вы хотите добавить или удалить. Следующие три включают длительность, параметр `easing` и функцию завершения. Это те же аргументы, которые вы передаете методу `effect()` (см. раздел «Эффекты jQuery UI» ранее в данной главе). Например, чтобы выбрать элемент с идентификатором `feature`, добавить имя класса `highlight` и анимировать визуальные изменения, вызванные добавлением этого класса, вы можете использовать следующий код:

```
$('#feature').addClass('highlight', 1000,↵
'easeOutBack', function () {
alert('Анимация завершена');
});
```

Этот фрагмент кода добавляет класс `highlight` к выбранному элементу и анимирует визуальное изменение элемента на протяжении 1 секунды (1000 миллисекунд), используя метод `easeOutBack` (описан в предыдущем разделе). После того как плагин jQuery UI закончит анимировать изменение класса, запускается функция обратного вызова, открывающая окно оповещения со словами «Анимация завершена». Для обеспечения этого анимационного эффекта необходимы только имя класса и значение продолжительности.

Плагин jQuery UI предусматривает четыре функции для анимации изменения класса:

- **Метод `addClass()`** позволяет добавить класс к одному или нескольким элементам и анимировать визуальные изменения в течение определенного периода времени.

ПРИМЕЧАНИЕ

Если вы не передадите аргумент, соответствующий значению продолжительности, какому-либо из этих методов, то плагин jQuery UI просто использует стандартный вид jQuery-функции без анимации. Например:

```
$('#feature').addClass('highlight');
```

Этот код назначает элементу класс `highlight`, но не анимирует изменение.

- **Метод `removeClass()`** работает так же, как метод `addClass()`. Вы передаете ему имя класса, продолжительность, дополнительное значение параметра `easing` и дополнительную функцию обратного вызова. Тем не менее эта функция удаляет указанный класс и анимирует любые визуальные изменения, вызванные удалением этого класса.
- **Метод `toggleClass()`** удаляет класс, назначенный элементу, или добавляет указанный класс, если он отсутствует. Вы можете использовать эту функцию вместе с событием `click` (см. раздел «События мыши» главы 5), чтобы добавить класс при первом щелчке по элементу, а затем удалить этот класс при следующем щелчке. Таким образом, вы можете создать элемент-переключатель, который либо включен, либо выключен. Разумеется, эта функция также анимирует изменения, вызванные добавлением или удалением класса.

ПРИМЕЧАНИЕ

Метод `toggleClass()` плагина jQuery UI также предусматривает расширенные параметры. Вы можете ознакомиться с ними на сайте: **api.jqueryui.com/toggleClass/**.

- **Метод `switchClass()`** является единственным методом jQuery UI, который не имеет эквивалента jQuery. Он принимает два имени класса: первый соответствует классу, который плагин jQuery UI уда-

ляет из выбранных элементов, а второй представляет собой класс, добавляемый плагином jQuery UI. Это подобно объединению методов `removeClass()` и `addClass()`, но позволяет вам анимировать замену удаляемого класса добавляемым. Например, если вы удаляете класс с красным фоном и добавляете класс с зеленым фоном, то плагин jQuery UI анимирует изменение цвета фона с красного на зеленый:

```
$('#feature').switchClass('defaultStyles', ↵
'highlight',1000);
```

ПРИМЕЧАНИЕ

Чтобы узнать о методах jQueryUI подробнее, посетите страницу Effects Core на сайте: **api.jqueryui.com/category/effects-core/**.

Часть IV

РАСШИРЕННЫЕ СПОСОБЫ ИСПОЛЬЗОВАНИЯ JQUERY И JAVASCRIPT

Глава 13. Введение в технологию Ajax

Глава 14. Создание приложения «Список дел»

Глава 13
ВВЕДЕНИЕ В ТЕХНОЛОГИЮ AJAX

Язык JavaScript — это мощное, но не всесильное средство. Если вы хотите представить информацию из базы данных, отослать электронное письмо с результатами заполнения формы или просто загрузить дополнительный HTML-код, вам необходимо связаться с веб-сервером. Для этого обычно нужно загрузить новую веб-страницу. Например, при поиске в базе данных вы обычно покидаете страницу поиска и переходите к странице с результатами.

Разумеется, ожидание загрузки новых страниц занимает какое-то время. От сайтов люди ждут быстроты и интерактивности, как от компьютерных программ. Сайты, подобные Facebook, Twitter, Google Docs и Gmail, фактически стирают грань между сайтами и прикладными программами. Технология программирования, делающая возможным появление этого нового поколения веб-приложений, носит название Ajax.

Технология Ajax позволяет веб-странице запрашивать данные и получать ответ от веб-сервера, после чего обновляться без необходимости в загрузке новой страницы. В результате сайт становится более интерактивным. Например, на сервисе Google Maps (рис. 13.1) вы можете увеличивать карту, перемещать ее во всех направлениях и даже перетаскивать, не загружая новую страницу.

Что такое Ajax

Термин «Ajax» появился в 2005 г. с целью охарактеризовать сущность новых сайтов, разработанных компанией Google: Google Maps (**maps.google.com**), Gmail (**www.gmail.com**). Ajax — это аббревиатура от Asynchronous JavaScript и XML (Асинхронные JavaScript и XML). Ajax не является официальной технологией вроде HTML, JavaScript или CSS. Под этим термином следует понимать сочетание ряда технологий — языка JavaScript, браузера и веб-сервера — для получения и отображения обновленной информации без загрузки новой страницы.

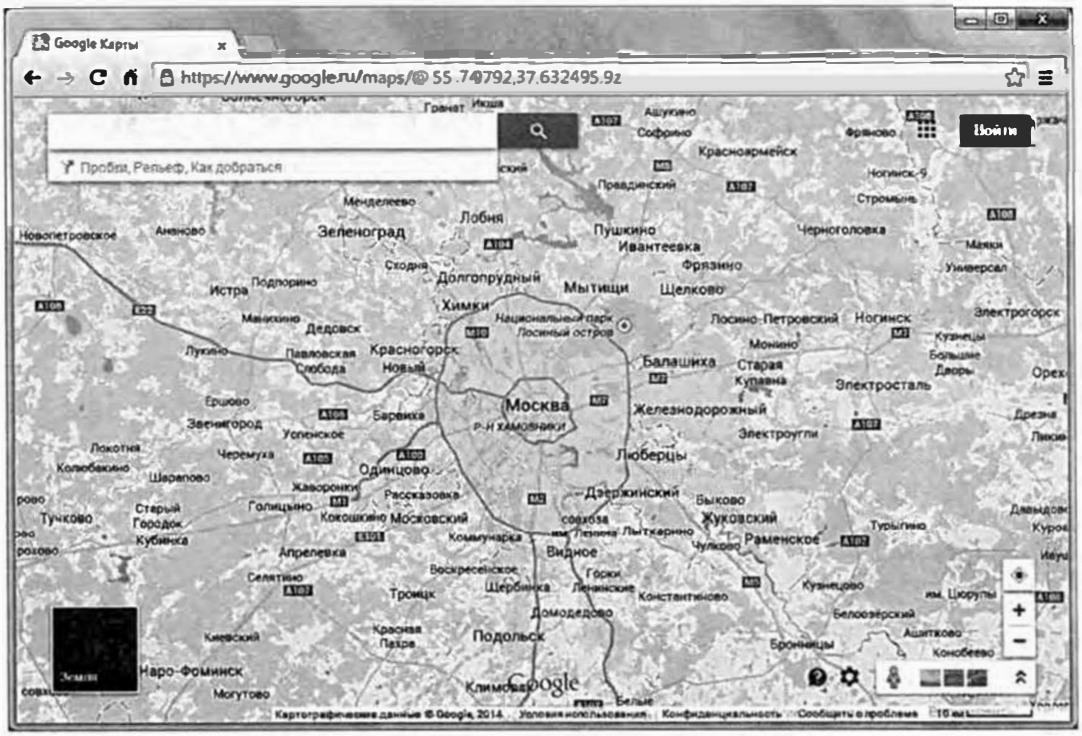

Рис. 13.1. Сервис Google Maps (**maps.google.com**) стал одним из первых крупных сайтов, применяющих технологию Ajax для обновления содержимого страницы без загрузки новых страниц. Это возможно благодаря тому, что обновлению подлежат лишь данные карты, другие части страницы (окно поиска, панель результатов поиска и кнопки управления картой) остаются неизменными

Говоря коротко, современные браузеры позволяют с помощью кода JavaScript посылать запрос на веб-сервер, который в ответ присылает браузеру некоторые данные. Программа JavaScript берет эти сведения и что-нибудь с ними делает. Например, если на странице сервиса Google Maps вы щелкнете по кнопке со стрелкой, указывающей на «север», то код JavaScript этой страницы запросит новые данные карты с сервера Google. Система Google пришлет в ответ новую информацию, которая будет использоваться для отображения нужного участка карты.

Хотя вы, возможно, и не собираетесь создавать сервис подобный Google Maps, есть много простых вещей, которые позволяет делать технология Ajax:

- **Отображение нового HTML-контента без перезагрузки страницы.** Например, на странице, где перечисляются заголовки новостей и отображаются статьи, когда посетитель щелкает по заголовку, вы можете избавить его от ожидания, пока загрузится новая страница. Вместо этого, новостная статья может появиться на той же странице без необходимости в перезагрузке остальных ее элементов. Далее в главе вы узнаете, как этого добиться.

- **Отправка формы и немедленный показ результата.** Например, возьмем форму для подписки на рассылку новостей. После заполнения и отправки она исчезает, зато немедленно появляется сообщение «Вы подписаны на рассылку новостей». О том, как создавать такие формы с помощью технологии Ajax, вы также узнаете в этой главе.
- **Авторизация без ухода со страницы.** Еще одним примером использования языка JavaScript является страница с небольшой формой авторизации. Заполнив ее, вы щелкаете по кнопке **Login** и оказываетесь авторизованным, страница трансформируется, показывая ваш статус, имя пользователя и, возможно, другую информацию, касающуюся именно вас.
- **Назначение рейтинга.** На сайтах со списками книг, фильмов и других товаров вы часто можете видеть линейки рейтингов (обычно включающих от одной до пяти звезд), иллюстрирующих оценку качества товаров посетителями. Эта рейтинговая система позволяет вам выразить свое мнение, выбрав желаемое количество звезд. С помощью технологии Ajax вы можете предоставить вашим посетителям возможность отдать предпочтение без необходимости покидать страницу: все, что они должны сделать, — это выбрать количество звезд. Существует хороший плагин библиотеки jQuery, делающий как раз это: **www.wbotelhos.com/raty/**.
- **Обзор информации базы данных.** Магазин Ozon.ru представляет собой типичный пример сетевой базы данных, которую вы можете просматривать. Когда вы ищете в этом магазине литературу, скажем, по теме JavaScript, то получаете список книг по JavaScript, продаваемых на этом сайте. Обычно позиций бывает так много, что все они не умещаются на странице, что вынуждает переходить от страницы к странице, чтобы увидеть «следующие 10 позиций». С помощью технологии Ajax вы сможете просматривать базу данных без необходимости переходить на другую страницу. Вот как используется технология Ajax на сервисе Twitter: при просмотре своей страницы вы видите несколько твитов пользователей, за чьими сообщениями вы следите. Если вы до конца прокрутите страницу, то сервис загрузит больше твитов. Прокрутите снова — появится еще больше сообщений. Похоже, что страница никогда не закончится.

Среди приведенных выше задач нет ничего революционного, за исключением упоминания об отсутствии необходимости загружать новую страницу. Тех же результатов можно достичь с помощью обычного HTML-кода и серверного программирования (например, для сбора дан-

ных форм или доступа к информации баз данных). Однако технология Ajax повышает интерактивность веб-страниц и улучшает впечатление посетителя от использования сайта.

Технология Ajax: Основы

Рассматриваемые в совокупности технологии, которые лежат в основе Ajax, достаточно сложны. Это язык JavaScript, серверное программирование и браузер, функционирующие совместно. Тем не менее базовую концепцию легко понять, если иметь ясное представление обо всех компонентах в отдельности. На рис. 13.2 показана разница между тем, как общаются с веб-сервером традиционные HTML-страницы, и как это делают страницы, использующие технологию Ajax.

Части мозаики

Ajax нельзя рассматривать как самостоятельную технологию, так как это смесь ряда технологий, функционирующих совместно для повышения эффективности работы пользователя. Как вы уже знаете, Ajax сводит вместе три разных компонента.

- **Браузер** — очевидно, что он необходим для просмотра веб-страниц и применения кода JavaScript. Есть тайный «ингредиент», встроенный в большинство современных обозревателей и делающий использование Ajax возможным, — объект `XMLHttpRequest`. За этим странно звучащим термином стоит как раз то, что позволяет коду JavaScript общаться с веб-сервером и получать в ответ информацию.

  Объект `XMLHttpRequest` был введен в структуру браузера Internet Explorer 5 много лет назад и постепенно «проник» во все основные браузеры. Подробнее об этом вы узнаете в следующем разделе.

- **Код JavaScript** выполняет главные функции Ajax: посылает запрос на веб-сервер, ожидает ответ, обрабатывает его и (как правило) обновляет страницу, добавляя новые данные или изменяя облик страницы определенным образом. В зависимости от того, что вы хотите от своей программы, код JavaScript может посылать данные заполненной формы, запрашивать дополнительные записи из базы данных или просто отправлять фрагмент информации (вроде рейтинга, только что присвоенного книге посетителем). После отправки дан-

ных на сервер программа JavaScript будет готова принять ответ, например, дополнительные записи из базы данных или просто сообщение вроде «Ваш голос учтен».

Рис. 13.2. Традиционный способ общения браузера с веб-сервером (вверху) заключается в запрашивании файла с сервера и получении в ответ веб-страницы. При этом происходят постоянные загрузка и перезагрузка веб-страниц. При использовании технологии Ajax браузер запрашивает только новую информацию. Сервер возвращает запрошенные данные, в результате чего с помощью кода JavaScript обновляются содержимое и вид веб-страницы (внизу)

Получив информацию, код JavaScript обновляет страницу — представляет новые записи базы данных или, например, сообщает посетителю, что он успешно авторизован. Обновление веб-страницы

включает манипулирование ее объектной моделью (объектная модель документа, DOM, описана в разделе «Объектная модель документа (DOM)» главы 4) для добавления, изменения и удаления HTML-элементов и контента. По существу, это как раз то, чем вы занимались, изучая большую часть настоящей книги: изменяли содержимое и облик страницы средствами языка JavaScript.

- **Веб-сервер** получает запросы от браузера и отправляет ему информацию. Сервер может просто вернуть определенный HTML-код или простой текст, а может — документ XML (см. врезку «Получение документа XML с сервера» далее в этой главе) или данные JSON (см. раздел «Формат JSON» этой главы). Например, если сервер получает информацию из формы, он может добавить ее в базу данных и отослать обратно сообщение, подтверждающее, что данные добавлены. Программа JavaScript также может послать запрос на следующие 10 записей базы данных, и веб-сервер в ответ пришлет информацию по этим 10 записям.

С серверной «частью уравнения» могут возникнуть некоторые сложности. Она обычно задействует несколько различных технологий, включая веб-сервер, сервер приложения и сервер базы данных. Веб-сервер представляет собой нечто вроде картотеки: он хранит документы и отсылает их по запросу браузера. Для выполнения более сложных задач, таких как передача сведений из формы в базу данных, вам понадобятся *сервер приложения* и *сервер базы данных*. Первый понимает языки серверного программирования — PHP, Java, C#, Ruby или Cold Fusion Markup — и позволяет выполнять задачи, которые нельзя решить только при помощи HTML-страницы (вроде отправки электронной почты, проверки цен на книги на сайте Ozon или хранения информации в базе данных). Второй дает возможность хранить такую информацию, как имена и адреса покупателей, описание товаров или архив ваших любимых рецептов. К числу распространенных серверов баз данных относятся MySQL, PostgreSQL и SQL Server.

ПРИМЕЧАНИЕ

Термин «сервер» можно отнести как к оборудованию, так и к программному обеспечению. В данной книге понятия «сервер приложения», «веб-сервер» и «сервер базы данных» имеют отношение к различным компонентам программного обеспечения, которые могут выполняться на одном и том же компьютере (как часто и бывает).

Существует множество разных сочетаний веб-серверов, серверов приложений и серверов баз данных. Например, вы можете использовать веб-сервер Microsoft IIS с ASP.NET (сервер приложения) и SQL Server (сервер базы данных). Еще одна комбинация: Apache (веб-сервер), PHP (сервер приложения) и MySQL (база данных).

ПРИМЕЧАНИЕ

Широко применяется сочетание бесплатных серверов Apache, PHP и MySQL (часто называемое AMP). Вы обнаружите, что большинство хостинговых компаний предоставляют именно эти серверы. В примерах для этой книги также используется комбинация AMP (см. врезку далее).

УСКОРЯЕМ РАБОТУ

Установка веб-сервера

Ajax работает с веб-сервером. Его главная задача — позволить коду JavaScript посылать информацию и получать ее с сервера. Хотя все (кроме одного) руководства, рассмотренные в этой и последующих главах, выполняются на вашем локальном компьютере без веб-сервера, вам понадобится доступ к серверу, если вы захотите продолжить изучение мира Ajax. Если вы уже создали свой сайт в Интернете, вы можете протестировать программы Ajax путем перемещения ваших файлов на веб-сервер. К сожалению, этот способ является трудоемким: вам придется создавать веб-страницы на компьютере и затем перемещать их на сервер с помощью программы FTP просто, чтобы проверить, функционируют они или нет.

Лучше создать сервер разработки, что предусматривает установку веб-сервера на ваш компьютер, это позволит программировать и тестировать ваши Ajax-страницы прямо на рабочем месте. Подобная задача может казаться устрашающей, но есть большое количество бесплатных программ, делающих установку всех необходимых компонентов такой же простой процедурой, как двойной щелчок кнопкой мыши по значку.

Пользователи операционной системы Windows могут установить компоненты Apache, PHP и MySQL с помощью бесплатной программы WAMP (**www.wampserver.com/ru/**), которая инсталлирует все элементы, необходимые для симуляции реального сайта, размещенного в Интернете.

Для поклонников операционной системы OS X также есть бесплатный инсталлятор MAMP (**www.mamp.info/en**), который предоставляет легкую в применении программу, включающую Apache, PHP и MySQL. (Существует также версия MAMP для операционной системы Windows.)

Последнее руководство данной главы требует наличия веб-сервера. Так что если вы собираетесь выполнить его, установите веб-сервер на ваш компьютер с помощью одной из указанных программ. Если у вас уже есть сайт, использующий другой веб-сервер (например, IIS от корпорации Microsoft), вам, вероятно, придется установить его на ваш компьютер, раз вы планируете создавать приложения Ajax для применения на реальном сайте. IIS уже установлен в ОС Windows 8, вам нужно всего лишь включить его. Вы можете узнать о том, как это сделать, посмотрев видео: **www.youtube.com/watch?v=mRm9-Xddt2w**.

Взаимодействие с веб-сервером

Главный компонент программы Ajax — это объект `XMLHttpRequest`, иногда называемый просто *XHR*. Он встроен в большинство современных браузеров и обеспечивает коду JavaScript возможность посылать информацию на веб-сервер и получать данные в ответ. Этот процесс состоит из пяти этапов, и все они выполняются с помощью кода JavaScript.

1. **Создайте экземпляр объекта `XMLHttpRequest`.**

   Этот первый шаг всего лишь сообщает браузеру: «Эй, я собираюсь послать информацию на сервер, так что приготовься». В общем виде создание объекта `XMLHttpRequest` на языке JavaScript можно записать так:

   ```
   var newXHR = new XMLHttpRequest();
   ```

   Несмотря на простоту приведенного фрагмента кода, создание программ Ajax исключительно с помощью JavaScript сопряжено с определенными сложностями. К счастью, библиотека jQuery предусматривает более простой способ создания запросов Ajax. Вы узнаете о нем в следующем разделе.

2. **Примените XHR-метод `open()` для определения способа отправки данных и места, куда они отправляются.**

   Посылать данные можно двумя способами: `GET` или `POST`, они аналогичны тем, которые использовались в HTML-формах. Метод `GET` посылает информацию на веб-сервер как часть URL-адреса, допу-

стим, `shop.php?productID=34`. В рассматриваемом примере данные отправляются в качестве «строки запроса», которая представляет собой информацию, следующую за символом «?» (в данном случае `productID=34`). Это пара имя/значение, где `productID` является именем, а `34` — значением. Именем может быть имя поля формы, а значением — данные, которые посетитель вводит в это поле.

Метод `POST` посылает данные отдельно от URL-адреса. Если метод `GET` обычно используется для получения данных с сервера, то метод `POST` служит для обновления информации на сервере (например, чтобы добавить, обновить или удалить запись в базе данных). Вы освоите оба метода в разделе «Функции `get()` и `post()`» далее в этой главе.

Вы также используете метод `open()` для выбора страницы на сервере, куда направляются данные. Обычно это та страница, которая использует серверный язык сценариев (например, PHP) для извлечения сведений из базы данных или выполнения иной программной задачи; вы указываете на нее с помощью URL-адреса. Например, следующий код сообщает объекту XHR, какой метод использовать (GET) и какую страницу на сервере запрашивать:

```
newXHR.open('GET', 'shop.php?productID=34');
```

ПРИМЕЧАНИЕ

URL-адрес, который вы задали для метода `open()`, должен находиться на том же сайте, что и страница, посылающая запрос. Из соображений безопасности браузер не разрешит посылать Ajax-запросы в другие домены. Формат данных JSONP позволяет обойти эту проблему. Вы узнаете об этом формате подробнее в разделе «Введение в JSONP» далее в главе.

3. Создайте функцию для обработки результатов.

Когда веб-сервер возвращает результат (новая информация из базы данных, подтверждение о передаче формы или просто текстовое сообщение), вам обычно нужно с ним что-то сделать. Можно просто написать сообщение «Форма успешно передана». Иногда необходимо заменить всю таблицу записей базы данных на новую. В любом случае вам придется написать функцию JavaScript для обработки результата (называемую *функцией обратного вызова*), которая часто является сутью вашей программы.

Как правило, эта функция манипулирует содержимым страницы (то есть изменяет ее DOM) путем удаления (например, удаляя форму, посланную с помощью Ajax), добавления (отображая сообщение «Форма успешно передана» или новую HTML-таблицу записей базы данных) или изменения элементов (например, выделяя количество звезд, которое посетитель только что выбрал при присвоении рейтинга товару).

Есть еще несколько этапов процесса, но, поскольку вы используете библиотеку jQuery для упрощения рутины, единственное, что вам необходимо понять о функции обратного вызова, это то, что она является кодом JavaScript для обработки ответа сервера.

4. **Пошлите запрос.**

   Для отправки запроса на сервер вы применяете метод объекта XHR `send()`. Все, что мы делали до этого момента, было просто подготовкой, а данный шаг сообщает серверу: «Мы готовы... посылай запрос!». Если вы пользуетесь методом `GET`, то код выглядит следующим образом:

   ```
   xhr.send();
   ```

   Метод `send()` может принимать аргумент — данные, которые вы посылаете на сервер. При использовании метода `GET` данные посылаются как часть URL-адреса, например, `search.php?q=javascript`, где данные — это фрагмент `q=javascript`. Используя метод `POST`, вы должны предоставить данные наряду с методом `send()`:

   ```
   xhr.send('q=javascript');
   ```

   Опять же, не задумывайтесь над деталями, вы увидите, как библиотека jQuery упрощает этот процесс, в следующем разделе.

   После того как запрос послан, ваша программа JavaScript не обязательно должна остановиться. Буква «*A*» в слове Ajax, как известно, означает *асинхронный*, а это подразумевает, что после отправки запроса код JavaScript может продолжать заниматься чем-то еще. Браузер «не сидит без дела», просто ожидая ответа от сервера.

5. **Примите ответ.**

   После того как сервер обработал запрос, он посылает ответ браузеру. Функция обратного вызова, созданная в пункте 3, обрабатывает ответ, и объект XHR получает при этом часть информации, включая *статус* запроса, *текст* ответа и, возможно, ответ в формате *XML*.

Статус ответа — это число, указывающее на характер ответа сервера: вам, вероятно, знаком код ошибки 404, означающий, что запрошенный файл не найден. Если все прошло по плану, вы получите статус 200 или, быть может, 304. В случае ошибки при обработке страницы вы получите статус 500 и сообщение «Внутренняя ошибка сервера», а если запрашиваемый файл защищен паролем, вы получите ошибку 403: «Доступ запрещен». Полный список кодов ошибок вы можете найти по адресу: **www.w3.org/Protocols/rfc2616/rfc2616-sec10.html**.

Чаще всего вы получаете текстовой ответ, который хранится в свойстве объекта XHR `responseText`. Это может быть фрагмент HTML-кода, простое текстовое сообщение или сложный набор данных JSON (см. раздел «Формат JSON» данной главы). Наконец, если сервер посылает в ответ XML-файл, он хранится в свойстве объекта XHR `responseText`. Хотя формат XML по-прежнему используется, более распространенными формами ответа для страниц сервера являются все же текстовые, HTML- и JSON-данные, так что вам, может, и вовсе никогда не придется обрабатывать XML-ответ.

Какие бы данные сервер ни возвращал, все они обрабатываются функцией обратного вызова для обновления веб-страницы. По завершении работы функции обратного вызова заканчивается полный цикл Ajax (однако вы можете отправлять несколько Ajax-запросов одновременно).

Работа с Ajax с помощью средств jQuery

Несмотря на то, что базовый процесс `XMLHttpRequest` не слишком сложен, всякий раз при создании запроса XHR необходимо выполнить много шагов. К счастью, библиотека jQuery предоставляет несколько функций, весьма упрощающих весь процесс. В конце концов, если вы посмотрите на пять шагов, которые мы изучили в предыдущем разделе, то увидите, что по-настоящему интересное действие — а именно программный код, который обрабатывает ответ сервера, — происходит только в пункте 3. Все остальные этапы упрощены с помощью библиотеки jQuery, так что вы можете сосредоточиться на интересном процессе программирования.

Использование метода `load()`

Простейшим из методов Ajax, предлагаемых библиотекой jQuery, является `load()`. Он загружает HTML-файл в указанный элемент на стра-

нице. Например, на вашей странице отведено место для списка заголовков новостей. При его загрузке отображаются пять последних новостных сообщений. Вы можете добавить ссылки, которые дали бы возможность посетителям выбирать самостоятельно, какие заголовки должны быть в данном месте: вчерашние события, местные новости, новости спорта и т. д. Конечно, это можно сделать с помощью ссылок на различные страницы, содержащие нужную информацию (и не использовать Ajax вовсе!), но тогда посетителям придется переходить на другую веб-страницу.

Альтернативным способом является загрузка выбранных новостных сюжетов в рамку с заголовками новостей на странице. Другими словами, каждый раз, когда посетитель щелкает по другой категории новостей, браузер запрашивает новый HTML-файл с сервера, а затем помещает этот файл в рамку с заголовками, не покидая текущую страницу (рис. 13.3).

УСКОРЯЕМ РАБОТУ

Способы серверного программирования

Если вы не используете jQuery-функцию `load()` (обсуждалась ранее) для вставки HTML-контента со страницы на веб-сервере на страницу в браузере, вам понадобится серверное программирование для использования технологии Ajax. Главный смысл Ajax заключается в том, чтобы позволить коду JavaScript общаться с сервером и получать от него информацию. В большинстве случаев это подразумевает наличие дополнительного сценария, выполняемого на веб-сервере и решающего задачи, которые не под силу коду JavaScript, например считывание информации из базы данных, отправка электронных сообщений или авторизация пользователя.

В настоящей книге вопросы серверного программирования не освещаются, так что вам следует изучить такие серверные технологии, как PHP, Ruby on Rails, .NET или JSP (или найти того, кто выполнит эту часть работы за вас). Если до сих пор вы не имели дело с языком серверного программирования, хорошо было бы начать с PHP (одного из самых популярных). Он бесплатный, и почти каждая компания веб-хостинга использует его в числе других на своих серверах. Это мощный язык и к тому же *относительно* простой для освоения. Если вы решили начать изучение серверного программирования с языка PHP, то вам следует познакомиться с книгами: «Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript» (издательства «Питер») или «Изучаем PHP и MySQL» (издательства «Эксмо»).

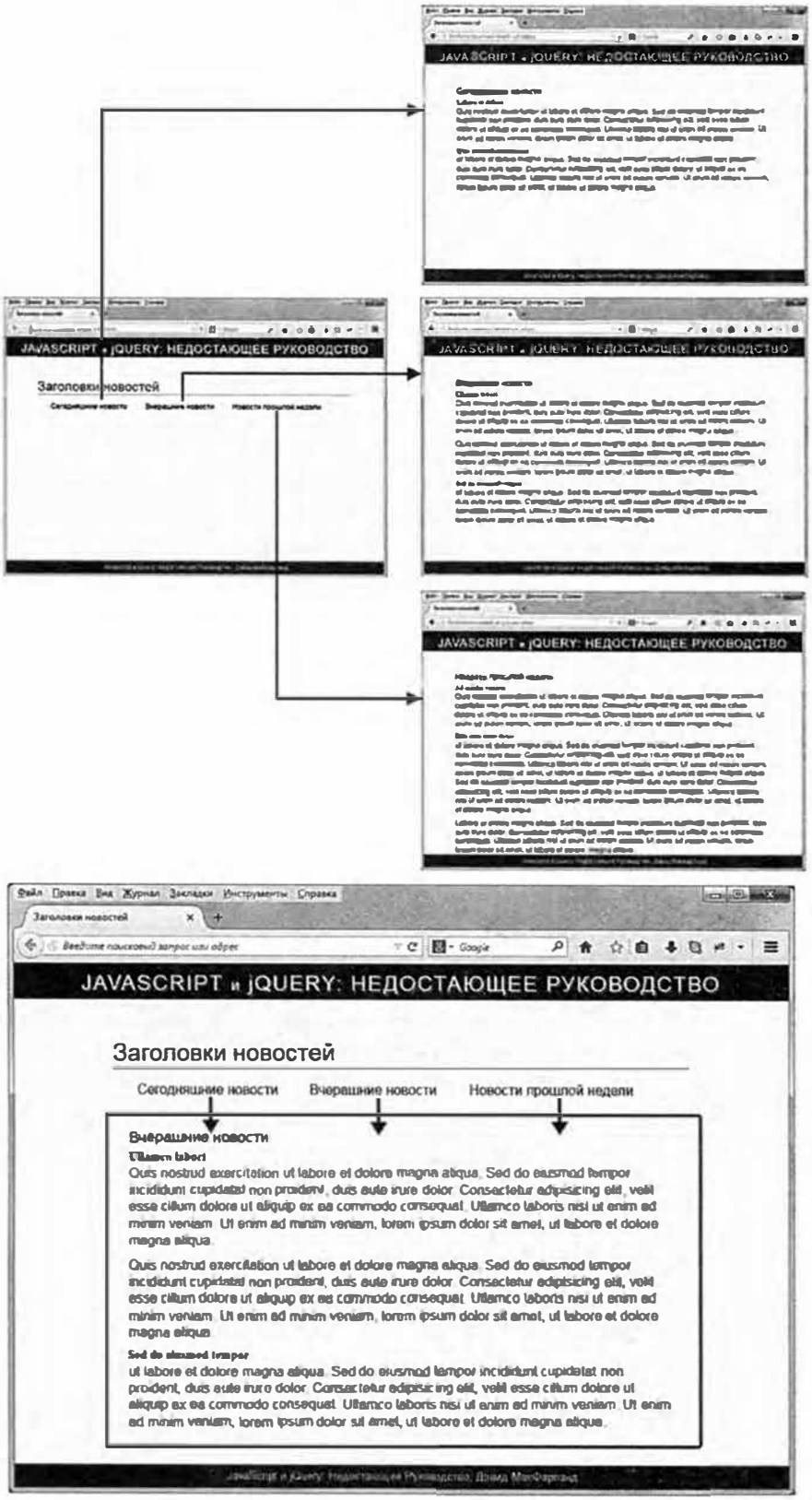

Рис. 13.3. Верхний набор изображений иллюстрирует работу ссылок — обычный способ доступа к дополнительному HTML-контенту. Перейдя по ссылке на странице (слева), вы загрузите совершенно новую страницу (справа). Однако с помощью технологии Ajax и jQuery-функции `load()` вы можете получить тот же самый HTML-контент, не покидая текущей страницы (внизу). Щелчок по ссылке загружает HTML-содержимое в элемент `div` (выделен на рисунке)

Также существует множество бесплатных онлайн-ресурсов для изучения языка PHP. В частности, PHP 101 от Zend, одной из ведущих компаний, развивающих технологию PHP, содержит много базовой и дополнительной информации (**devzone.zend.com/6/php-101-php-for-the-absolute-beginner/**). Сайт W3Schools также располагает разнообразными данными для начинающих программистов на языке PHP (**www.w3schools.com/PHP**).

Чтобы применить функцию `load()`, сначала воспользуйтесь селектором jQuery для идентификации элемента на странице, где должен быть размещен запрашиваемый HTML-контент; затем вызовите функцию `load()` и передайте URL-адрес страницы, которую нужно получить. Например, у вас есть элемент `div` с идентификатором `headlines` и вы хотите загрузить HTML из файла *todays_news.html* в данный элемент. Сделать это можно следующим образом:

```
$('#headlines').load ('todays_news.html');
```

В процессе выполнения кода браузер запрашивает файл *todays_news.html* у веб-сервера. После его загрузки браузер заменяет текущее содержимое элемента `div` с идентификатором `headlines` данными из нового файла. HTML-файл может представлять собой полную веб-страницу (включая элементы `html`, `head` и `body`) или только ее фрагмент, например, запрашиваемый файл может содержать всего лишь элемент `h1` и абзац текста. В этом случае этот HTML-фрагмент и вставляется на текущую (полную) страницу.

ПРИМЕЧАНИЕ

Загружать HTML-файлы можно только с этого же сайта. Например, вы не можете загрузить домашнюю страницу сервиса Google в элемент `div` на странице вашего сайта с помощью функции `load()`.

При использовании функции `load()` вам следует быть очень внимательными к путям файлов. Во-первых, URL-адрес, который вы передаете функции `load()`, должен быть связан с текущей страницей. Другими словами, вы используете такой же путь, как если бы ссылались из текущей страницы на HTML-файл, который нужно загрузить. Кроме того, пути к файлам в HTML-коде не переписываются при загрузке этого HTML в документ, так что если загруженный файл имеет ссылку или изображения, их URL-адреса должны быть связаны со страницей, использующей функцию `load()`. Иначе говоря, если вы используете пути

относительно документа (см. врезку «Типы URL-адресов» главы 1), а загруженный HTML-файл находится в другой папке на вашем сайте, то изображения и ссылки могут оказаться нерабочими после загрузки HTML-контента на текущую страницу. Вот простое решение: используйте ссылки относительно корневого каталога, или удостоверьтесь, что загружаемый файл находится в той же папке, что и страница, использующая функцию `load()`.

Кроме того, функция `load()` позволяет указать, какую часть загруженного HTML-файла вы хотите добавить на страницу. Например, запрашиваемая страница является обычной страницей с сайта: она включает такие привычные элементы, как баннер, панель навигации и нижний колонтитул. Допустим, вас интересует лишь часть содержимого этой страницы, находящаяся в отдельно взятом элементе `div`. Чтобы выбрать ее, вставьте пробел и селектор jQuery после URL-адреса. В нашем примере нам нужно вставить содержимое элемента `div` с идентификатором `news` в файле *todays_news.html*. Для этого используем следующий код:

```
$('headlines').load('todays_news.html #news');
```

В данном случае браузер загружает страницу *todays_news.html*, но не вставляет все ее содержимое в элемент `div` с идентификатором `headlines`, а извлекает только элемент `div` (и все, что там есть) с идентификатором `news`. Вы увидите этот метод в действии в следующем руководстве.

Функция `load()` на практике

С помощью библиотеки jQuery мы попробуем заменить традиционный способ доступа («нажал и загрузил») к HTML-контенту (рис. 13.3 вверху) методом, который просто замещает содержимое текущей страницы новым HTML-кодом (рис. 13.3 внизу).

Обзор

Чтобы понять, что следует делать в этом руководстве, сначала вам необходимо разобраться в HTML-коде страницы, которую предстоит «Ajax-нуть». Взгляните на рис. 13.4: на странице расположен маркированный список со ссылками, каждая из которых указывает на отдельную страницу, содержащую определенные заголовки новостей. Эле-

мент ul, использованный для создания списка, имеет идентификатор newslinks. Кроме того, есть пустой элемент div под заголовком «Заголовки новостей». Он имеет идентификатор headlines и в данный момент является пустым местом для подстановки. При использовании jQuery-функции load() нажатие на одну из ссылок загрузит новостные статьи в элемент div.

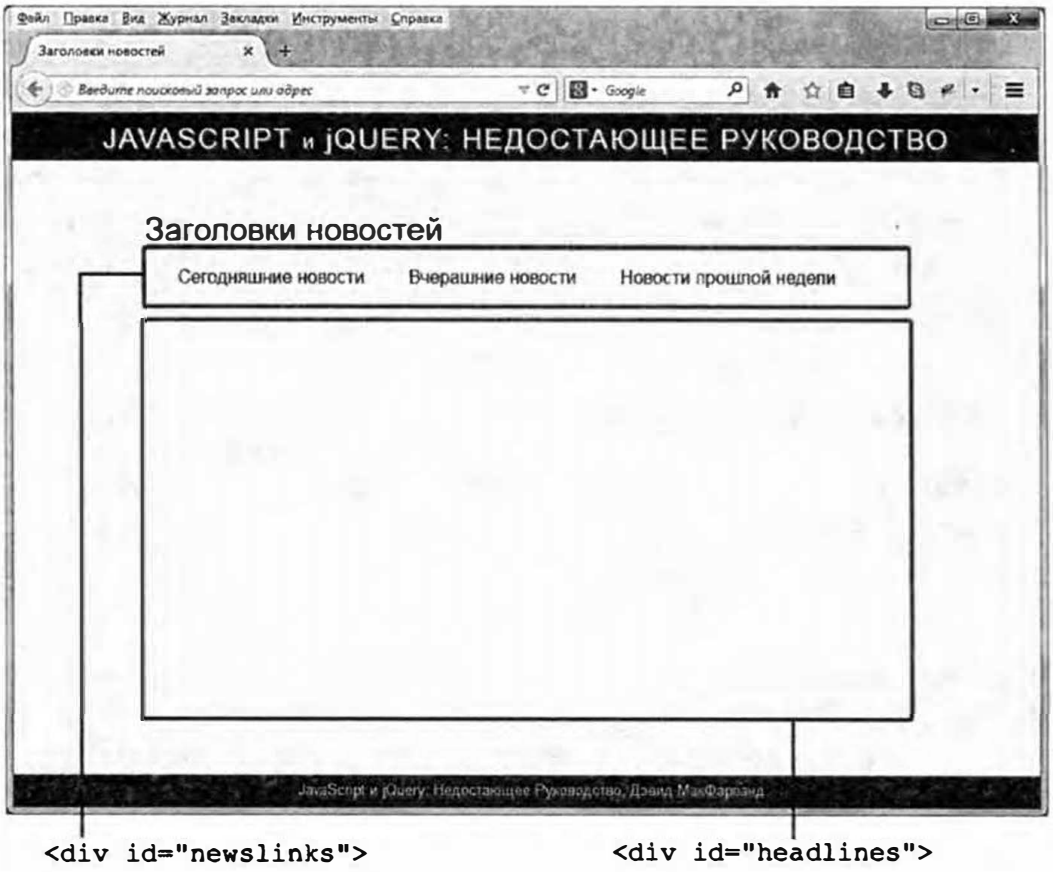

Рис. 13.4. Когда вы хотите использовать код JavaScript для добавления содержимого на страницу, следует вставить пустой элемент div с идентификатором. Когда понадобится, вы можете выбрать этот элемент div и вставить содержимое. Например, данная страница содержит пустой элемент div ( ) в правой боковой панели. С помощью технологии Ajax несложно заполнить этот элемент div содержимым любой из трех связанных страниц, перечисленных в центре

Сейчас щелчок по ссылке просто открывает веб-страницу с новостями. Другими словами, данная страница работает стандартным для языка HTML способом — она имеет ссылки, указывающие на другие страницы. В сущности, страница работает отлично и без кода JavaScript, предлагая любому посетителю ту новость, которая ему нужна. И это хорошо, поскольку не у всех пользователей в браузере активирован JavaScript. Кроме того, если единственный путь получения новостей предусматри-

вал бы использование кода JavaScript, то поисковые системы пропускали бы эту ценную информацию.

ПРИМЕЧАНИЕ

Вы можете использовать метод `load()` непосредственно с вашего жесткого диска без участия веб-сервера, так что вам необязательно устанавливать его на компьютер (см. врезку «Установка веб-сервера» в начале данной главы). На момент написания этой книги браузер Safari в операционной системе OS X не позволял использовать метод `load()` без веб-сервера.

Данное руководство представляет собой пример *прогрессивного улучшения* — страница прекрасно работает и без кода JavaScript, но с ним — еще лучше. Иначе говоря, каждый может получить доступ к информации, и никто не будет забыт.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Верстка кода

Для осуществления принципа прогрессивного улучшения вы добавите код JavaScript с целью «захватить» обычную функцию ссылки, затем получить URL-адрес этой ссылки, после чего загрузить ссылку на страницу и поместить ее содержимое в пустой элемент `div`. Вот так просто.

1. **В текстовом редакторе откройте файл *13_01.html* из каталога *глава13*.**

   Начнем с присвоения события `click` каждой из ссылок маркированного списка в главной части страницы. Маркированный список (элемент `ul`) имеет идентификатор `newslinks`, так что с помощью библиотеки jQuery вы легко можете выбрать каждую ссылку и присвоить ей функцию `click()`.

   Этот файл уже содержит ссылку на библиотеку jQuery и элемент `script` с функцией `$(document).ready()`.

2. **Щелкните мышью в пустой строке внутри функции `$(document).ready()` и введите код:**

```
$('#newslinks a').click(function() {

});
```

Код `$('#newslinks a')` — это способ выбора каждой ссылки, используемый библиотекой jQuery, а функция `.click()` присваивает функцию (обработчик событий) событию `click` (для получения сведений о событиях см. раздел «Использование событий: способ jQuery» главы 5).

Следующий шаг — извлечение URL-адреса из каждой ссылки.

3. **Внутри функции `click()` (пустая строка в пункте 2) введите код `var url=$(this).attr('href');` и нажмите клавишу Enter для создания пустой строки.**

   Эта строка кода создает новую переменную (`url`) и присваивает ей значение атрибута ссылки `href`. Как вы помните из раздела «Понимание выборок jQuery» главы 4, при присоединении функции (типа `click()`) к выборке jQuery (`$('#newslinks a')` в нашем случае) библиотека сканирует все элементы выборки (каждую ссылку) и применяет к каждому из них функцию. Код `$(this)` — это способ обращения к выбранному в данный момент элементу. Иначе говоря, ключевое слово `$(this)` будет относиться к разным ссылкам при проработке коллекции элементов в цикле. Функция `attr()` (обсуждалась в разделе «Чтение, установка и удаление атрибутов HTML» главы 4) может извлекать или задавать определенный элемент для тега; в данном случае функция извлекает свойство `href` для получения URL-адреса страницы, на которую указывает ссылка. Следующим шагом используем этот URL-адрес наряду с функцией `load()` для извлечения содержимого страницы и его вставки в элемент `div` на странице.

4. **Введите фрагмент `$('#headlines').load(url);` так, чтобы код выглядел следующим образом:**

```
$('#newslinks a').click(function() {
var url=$(this).attr('href');
$('#headlines').load(url);
});
```

   Вспомните, что пустой элемент `div` на странице (куда будет помещен загруженный HTML-код) имеет идентификатор `headlines`, поэтому код `$('#headlines')` выбирает этот элемент `div`. Функ-

ция `load()` затем загружает HTML-контент, соответствующий URL-адресу, который извлекла предыдущая строка кода, и помещает это содержимое в элемент `div`. Да, безусловно, «под капотом» происходит еще *много* чего, однако благодаря библиотеке jQuery вам не нужно об этом беспокоиться.

Страница еще не завершена. Если вы сейчас сохраните файл и откроете его в браузере, то заметите, что щелчок по ссылке не загружает новой информации на страницу, а вместо этого загружает новую страницу! Что случилось с Ajax? Все в порядке, просто браузер по-прежнему следует своей обычной привычке загружать новую страницу при щелчке по ссылке. Чтобы не допустить этого, вам нужно предотвратить переход по ссылке.

5. **Добавьте пустую строку после кода, введенного на предыдущем этапе, и напишите `return false;`. Таким образом, сценарий примет следующий вид:**

```
$('#newslinks a').click(function() {

var url=$(this).attr('href');

$('#headlines').load(url);

return false;

});
```

Подобным образом код сообщает браузеру: «Эй, браузер, не следуй по данной ссылке». Это один из способов предотвратить нормальное поведение браузера в ответ на событие. Для достижения того же эффекта вы также можете применить и jQuery-функцию `preventDefault()`, описанную в разделе «Отмена обычного поведения событий» главы 5.

6. **Сохраните файл.**

Если на вашем компьютере установлен веб-сервер, вы можете просмотреть эту страницу в браузере. Щелкните по какой-нибудь ссылке.

Теперь возникла другая проблема, как можно увидеть на рис. 13.5. Функция `load()` работает, но страница выглядит странно. Функция `load()` загрузила всю страницу и вставила ее в элемент `div`. Это привело к тому, что баннер «JavaScript и jQuery» дублирован в верхней части страницы. На деле же вам необходима лишь часть этой страницы — область с новостными сюжетами. К счастью, функция `load()` может помочь и здесь.

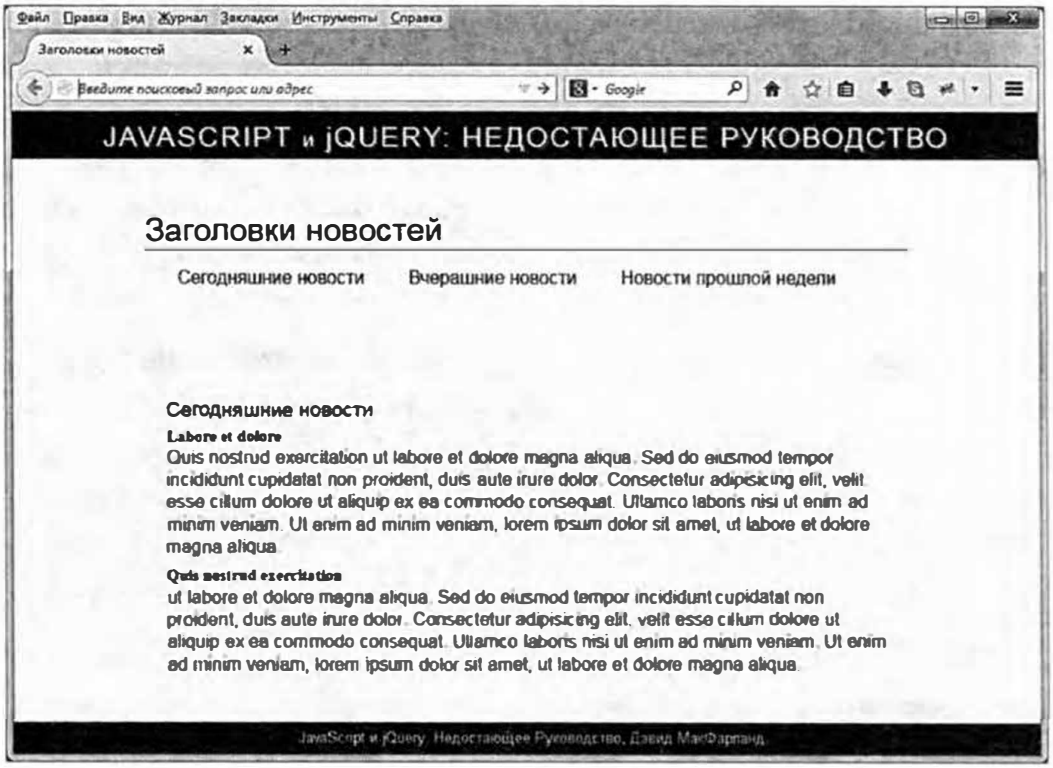

Рис. 13.5. Функция библиотеки jQuery `load()` загружает весь HTML-код выбранного файла и помещает его в элемент на текущей странице. Если загруженный файл включает ненужные части HTML вроде дублированных баннеров, боковых панелей и нижних колонтитулов, то результат будет похож на страницу в странице. В этом случае над заголовком «Сегодняшние новости» окажется много пустого места и баннер в верхней части страницы будет смещен. Фактически это дублированный баннер, помещенный поверх старого

7. **Найдите строку с функцией `load()` и добавьте фрагмент `+ ' #newsItems'` после `url`. Окончательный код должен выглядеть так:**

```
$('#newslinks a').click(function() {
var url=$(this).attr('href');
$('#headlines').load(url + ' #newsItems');
return false;
});
```

Как было сказано в предыдущем разделе, вы можете выбрать нужную часть загружаемого файла, которую функция `load()` добавит на страницу. Для этого вставьте пробел после URL-адреса, за которым введите селектор, идентифицирующий эту самую часть загружаемой страницы.

Вот как работает код: во-первых, на каждой из связанных страниц есть элемент `div` с идентификатором `newsItems`. Он содержит

нужный вам фрагмент HTML-контента — новостные статьи. Поэтому вы можете приказать функции `load()` вставить только данную часть загружаемого HTML-документа путем добавления пробела и фрагмента кода `#newsItems` к URL-адресу, переданному функции `load()`. Например, если вы хотите загрузить файл *today.html* и поместить в элемент `headlines` только HTML-контент, находящийся внутри элемента `newsItems`, используйте такой код:

```
$('#headlines').load('today.html #newsItems');
```

В данном случае вам нужно объединить две строки — содержимое переменной `url` и фрагмент `'#newsItems'` — для получения корректного кода, поэтому вы используете операцию конкатенации строк JavaScript (символ +): `load(url + ' #newsItems')`. (См. раздел «Объединение строк» главы 2 для получения сведений о том, как объединить две строки.)

8. **Сохраните файл.**

   Если у вас на компьютере установлен веб-сервер, вы можете просмотреть страницу в браузере. Проверьте ссылки. Теперь новостные статьи — и только они — из каждой связанной страницы должны появляться в середине страницы. Технология Ajax в нескольких строках кода! (Полную версию руководства *готовый_13_01.html* можно найти в папке *глава13*.)

Методы `get()` и `post()`

Метод `load()`, описанный в начале данной главы, представляет собой быстрый способ получения HTML-контента от веб-сервера и вставки его на страницу. Но сервер не всегда способен возвращать непосредственно HTML-код — ответом могут быть сообщения, номера кодов или данные, требующие дальнейшей обработки средствами JavaScript. Например, если вы используете технологию Ajax для получения определенных записей из базы данных, сервер может возвратить XML-файл, содержащий эти записи (см. врезку «Получение XML-файла с сервера» далее в главе), или объект JSON (см. раздел «Формат JSON» этой главы). У вас не получится вставить эти данные на страницу — их придется обработать для получения HTML-кода.

JQuery-функции `get()` и `post()` являются простыми инструментами для отправки и получения данных с веб-сервера. Как отмечалось в пункте 2 в разделе «Взаимодействие с веб-сервером», обработка объ-

екта XMLHttpRequest методами GET или POST имеет несколько различий. Однако библиотека jQuery учитывает данные различия, так что функции get() и post() работают одинаково. (Так какую же функцию использовать? Читайте врезку «GET или POST?»)

Базовая структура этих функций такова:

```
$.get(url, data, callback);
```

или:

```
$.post(url, data, callback);
```

В отличие от большинства других функций библиотеки jQuery, вам не нужно добавлять get() или post() в селектор jQuery, другими словами, вам никогда не придется использовать код, подобный этому: $('#mainContent').get('products.php'). Обе функции не связаны с какими-либо элементами на странице, так что вы просто используете символ $, после которого следует точка и слово get или post: $.get().

Функции get() и post() принимают три аргумента. Аргумент url — это строка, содержащая путь к серверному сценарию обработки данных (например, 'processForm.php'). Аргумент data является либо строкой, либо литералом объекта JavaScript, содержащим данные, которые вы хотите послать на сервер (как это сделать, вы узнаете в следующем разделе). Наконец, аргумент callback является функцией обратного вызова, которая обрабатывает информацию, возвращаемую сервером (см. раздел «Выполнение действия после завершения эффекта» главы 6 для получения сведений о деталях написания функции).

При выполнении функции get() или post() браузер посылает данные по указанному URL-адресу. Когда сервер возвращает ответные данные, браузер передает их функции обратного вызова, которая обрабатывает эту информацию и обновляет страницу. Вы увидите эту процедуру в действии в следующем руководстве.

Форматирование данных, посылаемых на сервер

Как правило, при написании программ JavaScript, использующих технологию Ajax, вы посылаете какую-то информацию на сервер. Например, если вам нужны сведения о конкретном товаре из базы данных, то вы можете просто послать номер, идентифицирующий этот товар.

Когда сервер получает его в запросе XHR, он ищет в базе данных товар с совпадающим номером, извлекает информацию о товаре и посылает ее обратно браузеру. Вы также можете применить технологию Ajax для отправки заполненной формы как части онлайн-заказа или формы «Подпишитесь на нашу рассылку новостей».

В любом случае посылаемые данные необходимо отформатировать способом, понятным для функций `get()` и `post()`. Их второй аргумент содержит данные, посылаемые на сервер, вы можете форматировать их как строку запроса или как литерал объекта JavaScript, о чем говорится в следующих двух разделах.

Строка запроса

Вы могли видеть строки запросов ранее: они часто появляются в конце URL-адреса, следуя после знака «?». Например, в этом URL-адресе `http://www.chia-vet.com/products.php?prodID=18&sessID=1234` строкой запроса является: `prodID=18&sessID=1234`. Она содержит две пары имя/значение — `prodID=18` и `sessID=1234` — и выполняет в принципе ту же задачу, что и создание двух переменных, `prodID` и `sessID` и сохранение в них двух значений. Строка запроса является обычным методом передачи информации в URL-адресе.

Форматировать данные, посылаемые на сервер, можно и с помощью технологии Ajax. Например, вы создали веб-страницу, на которой посетители имеют возможность присваивать рейтинг фильмам, выбирая определенное количество звезд. Щелчок по пяти звездам означает отправку на сервер информации о рейтинге 5. В этом случае данные, посланные на сервер, могут выглядеть так: `rating=5`. Допустим, страница, обрабатывающая эти рейтинги, называется `rateMovie.php`, тогда код для отправки на сервер данных с помощью Ajax мог бы выглядеть так для метода GET:

```
$.get('rateMovie.php','rating=5');
```

а для метода POST — так:

```
$.post('rateMovie.php','rating=5');
```

ПРИМЕЧАНИЕ

JQuery-функции `get()` и `post()` не требуют от вас задавания данных или функции обратного вызова. Вам нужно лишь сообщить

URL-адрес серверной страницы. Однако почти всегда вы будете предоставлять и данные. Например, в коде `$.get('rankMovie.php','rating=5');` посылаются только URL-адрес и данные, никакая функция обратного вызова не задана. В данном случае посетитель просто отправляет рейтинг, и серверу нет необходимости отвечать, а функции обратного вызова что-либо делать.

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

GET или POST?

Методы отправки данных на веб-сервер GET и POST мало чем отличаются. Что же мне выбрать?

Ответ действительно неоднозначен. В некоторых случаях у вас нет выбора. Например, вы посылаете информацию серверному сценарию, который уже выполняется на вашем сервере. Иначе говоря, серверное программирование завершено, и вам просто нужно воспользоваться кодом JavaScript для налаживания коммуникации. В этом случае вы используете тот метод, который предусмотрен в сценарии. Вероятнее всего, программист настроил сценарий на прием либо GET-, либо POST-данных. Таким образом, вы можете либо поговорить с программистом, либо заглянуть в сценарий, чтобы увидеть, какой метод им предусмотрен, после чего выбрать нужную функцию библиотеки jQuery — `get()` и `post()`.

Если вы (или другой программист) еще не написали серверного сценария, с которым ваша программа JavaScript будет общаться, тогда вы должны сами выбрать метод. Метод GET подходит для запросов, которые не влияют на структуру базы данных или файлов на сервере. Другими словами, используйте этот метод, если вам нужно *получить* информацию, например, для запроса цены определенного товара или списка самых популярных продуктов. Метод POST удобен для отправки данных, изменяющих информацию на сервере, подобно запросу на удаление файла, обновление базы данных или добавление в нее новых записей.

На деле вы можете использовать любой метод, нередко программисты применяют метод GET для удаления информации из базы, а метод POST — для извлечения данных с сервера. Однако существует ситуация, однозначно требующая метода POST. Скажем, если вы отправляете на сервер форму, содержащую большой объем информации, например, пост в блоге, который может состоять из сотен слов, в этом случае пользуйтесь методом POST. Метод GET имеет

встроенный лимит на объем данных, которые можно посылать (варьируется для разных браузеров, но в среднем составляет несколько тысяч знаков). Как правило, веб-дизайнеры пользуются методом POST для форм, содержащих достаточно много элементов.

Если вам нужно послать на сервер более одной пары имя/значение, вставьте между ними знак `&`:

```
$.post('rateMovie.php','rating=5&user=Вова');
```

Вам нужно аккуратно применять этот метод, поскольку некоторые знаки принимают специальное значение после вставки в строку запроса. Например, знак `&` служит для включения дополнительных пар имя/значение; знак = присваивает значение имени. Следующая строка запроса является некорректной:

```
'favFood=Mac & Cheese' // неправильно
```

Знак `&` здесь является частью значения `Mac & Cheese`, но в составе строки запроса символ `&` означает вторую пару имя/значение. Если вы хотите использовать специальные знаки, то вам следует их *кодировать* или *экранировать* во избежание конфликтов с альтернативными значениями. Например, знак пробела кодируется как `%20`, знак `&` — как `%26`, а знак = — как `%3D`. Таким образом, предыдущий код следует переписать так:

```
'favFood=Mac%20%26%20Cheese' // правильно
```

Язык JavaScript предоставляет для этих целей метод `encodeURIComponent()`. Примените функцию `endcodeURI Component()` к строке, и она ее правильно зашифрует. Например:

```
var queryString = 'favFood=' + encodeURIComponent ↵
('Mac & Cheese');
$.post('foodChoice.php', queryString);
```

Литерал объекта

Для коротких и простых фрагментов данных (не содержащих знаки пунктуации) метод строки запроса работает хорошо. Но более надежным способом, поддерживаемым функциями jQuery `get()` и `post()`, является использование литерала объекта для хранения данных. Как вы помните из раздела «Одновременное изменение нескольких свойств CSS» главы 4, литерал объекта (объектная константа) — это метод

JavaScript для хранения пар имя/значение. Базовая структура литерала объекта такова:

```
{
name1: 'value1',
name2: 'value2'
}
```

Вы можете передавать литерал объекта непосредственно функциям `get()` или `post()`. Например, следующий код использует метод строки запроса:

```
$.post('rankMovie.php','rating=5');
```

Для использования литерала объекта перепишите код так:

```
$.post('rankMovie.php', { rating: 5 });
```

Вы можете либо передать литерал объекта непосредственно функции `get()` или `post()`, либо сначала сохранить его в переменной, а затем передать ее функции `get()` или `post()`:

```
var data = { rating: 5 };
$.post('rankMovie.php', data);
```

Разумеется, вы можете включить любое количество пар имя/значение в объект, передаваемый функции `get()` или `post()`:

```
var data = {
rating: 5,
user: 'Вова'
}
$.post('rankMovie.php', data);
```

Или, если вы напрямую передаете объектную константу методу `$.post()`:

```
$.post('rankMovie.php',
{
rating: 5,
user: 'Вова'
}
); // конец post
```

JQuery-функция `serialize()`

Создание строки запроса или литерала объекта для всех пар имя/значение формы может оказаться настоящим испытанием. Вы должны будете извлечь имя и значение для каждого элемента формы, а затем объединить их, чтобы составить единую длинную строку запроса или большой литерал объекта JavaScript. К счастью, библиотека jQuery предусматривает функцию, облегчающую процедуру конвертации данных формы в информацию, которую могут использовать функции `get()` и `post()`.

Функция `serialize()` может применяться к любой форме (или даже выбранным элементам формы). Для ее применения сначала создайте выборку jQuery, включающую форму, затем присоедините функцию `serialize()`. Например, у вас есть форма с идентификатором `login`. Если вы хотите создать для нее строку запроса, сделайте это следующим образом:

```
var formData = $('#login').serialize();
```

Часть `var formData` просто создает новую переменную; код `$('#login')` — это выборка jQuery, содержащая форму; наконец, фрагмент `.serialize()` собирает все имена элементов формы и текущие значения каждого элемента и создает единую строку запроса.

Чтобы использовать эту строку с функциями `get()` или `post()`, просто передайте преобразованные функцией `serialize()` результаты в качестве второго аргумента после URL-адреса. Например, вы хотите послать содержимое формы `login` на страницу *login.php*. Для этого используйте такой код:

```
var formData = $('#login').serialize();
$.get('login.php',formData,loginResults);
```

Данный код посылает данные из формы в файл *login.php*, используя метод `GET`. Последний аргумент функции `get()` — `loginResults` — представляет собой функцию обратного вызова, которая берет данные, возвращаемые с сервера, и обрабатывает их. Далее вы узнаете, как создается функция обратного вызова.

Обработка данных с сервера

Технология Ajax — это улица с двусторонним движением. Программа JavaScript посылает данные на сервер, а тот, в свою очередь, возвращает ответные сведения программе JavaScript, которая может использовать

их для обновления страницы. Ранее вы изучили, как форматировать данные и посылать их на сервер, используя функции `get()` и `post()`. Сейчас вы узнаете, как получать и обрабатывать ответ сервера.

Браузер посылает запрос на сервер с помощью объекта `XMLHttpRequest` и ожидает ответ. Когда он приходит, функция обратного вызова обрабатывает его. Ей передаются несколько аргументов, первый из которых (самый важный) и определяет данные, возвращаемые сервером.

Вы можете форматировать сведения, возвращаемые сервером, любыми способами. Серверный сценарий возвращает число, слово, абзац текста или целую страницу. В случаях, когда сервер присылает большой объем информации (подобно группе записей базы данных), он часто использует форматы XML или JSON (см. врезку «Получение документа XML с сервера» и раздел «Формат JSON» в этой главе).

Второй аргумент функции обратного вызова представляет собой строку, указывающую статус ответа. Чаще всего статус `success` означает, что сервер успешно обработал запрос и возвратил данные. Однако иногда обработка запроса не удается, например, был запрошен файл, который уже не существует, или произошла ошибка в серверной программе. В таких случаях функция обратного вызова получает сообщение со статусом `error`.

Функция обратного вызова определенным образом обрабатывает данные и, как правило, обновляет веб-страницу, заменяя отосланную форму результатами, полученными с сервера, или, например, просто отображая на странице сообщение «request successful». Процесс обновления содержимого страницы облегчается функциями jQuery `html()` и `text()`, описанными в разделе «Добавление содержимого на страницу» главы 4. Другие методы манипулирования DOM страницы обсуждались в главе 4.

Чтобы разобраться с полным циклом запрос/ответ, обратимся к примеру с присвоением рейтингов фильмам (рис. 13.6). Посетитель может выбрать рейтинг, щелкнув по одной из пяти ссылок, каждая из которых указывает на соответствующий рейтинг. После щелчка по ссылке рейтинг и идентификатор фильма посылаются на сервер, программа которого вносит эту информацию в базу данных, а затем возвращает среднее значение рейтинга для данного фильма. Это среднее значение отображается на веб-странице.

Рис. 13.6. На этой странице посетитель выбирает ссылку для присвоения рейтинга фильму (вверху). Применив технологию Ajax, вы можете отослать рейтинг на сервер, не покидая страницу, а используя ответ с сервера — обновить содержимое страницы (внизу)

Чтобы на данной странице обойтись без кода JavaScript, каждая ссылка указывает на динамическую серверную страницу, которая может обрабатывать рейтинг, присвоенный посетителем. Например, ссылка пятизвездочного (высшего) рейтинга (см. рис. 13.6) может иметь вид: `rate.php?rate=5&movie=123`. Имя серверного файла, обрабатывающего рейтинги, — `rate.php`, тогда как строка `?rate=5&movie=123` включает два фрагмента информации для сервера: рейтинг (`rate=5`) и номер, который идентифицирует фильм (`movie=123`). Вы можете использовать код JavaScript для перехвата нажатий на ссылки и преобразования их в Ajax-вызовы на сервер:

```
1 $('#message a').click(function() {
2     var href=$(this).attr('href');
3     var querystring=href.slice(href.indexOf('?')+1);
4     $.get('rate.php', querystring, processResponse);
5     return false; // блокировать переход по ссылке
6 });
```

Строка 1 выбирает каждую ссылку (элемент `a`) внутри другого элемента с идентификатором `message` (в данном примере каждая ссылка, используемая для присвоения рейтинга фильму, находится внутри элемента `div` с идентификатором `message`). Затем к событию щелчка для каждой их этих ссылок применяется функция.

Строка 2 извлекает атрибут ссылки HREF, так, например, переменная `href` может содержать URL-адрес типа `rate.php?rate=5&movie=123`. Строка 3 извлекает часть кода после знака «?» в URL-адресе, используя метод `slice()` (см. раздел «Извлечение строки методом `slice()`» главы 16) для получения части строки и метод `indexOf()` (см. раздел «Поиск в строке: техника `indexOf()`» главы 16) для определения положения знака «?» (эта информация используется функцией `slice()` для определения начала разрезания).

Строка 4 представляет собой Ajax-запрос. Используя метод `GET`, запрос, содержащий строку запроса для ссылки, направляется серверному файлу `rate.php` (рис. 13.7). Результаты передаются функции обратного вызова `processResponse`. Строка 5 предотвращает стандартное поведение ссылки и не дает браузеру загрузить связанную с ней страницу.

ПРИМЕЧАНИЕ

За сведениями о создании функции и ее работе обратитесь к разделу «Функции: многократное использование кода» главы 3 .

Наконец пришла пора создать функцию обратного вызова. Она принимает данные и строку со статусом ответа (`'success'`, если сервер возвратил данные). Помните, что имя функции обратного вызова используется в запросе (см. код в строке 4 на предыдущей странице). В данном случае имя функции — `processResponse`. Код для обработки ответа с сервера может выглядеть так:

```
1 function processResponse(data) {
2     var newHTML;
3     newHTML = '<h2>Ваш голос учтен</h2>';
4     newHTML += '<p>Средний рейтинг для этого ↵
      фильма составляет ';
5     newHTML += data + '.</p>';
6     $('#message').html(newHTML);
7 }
```

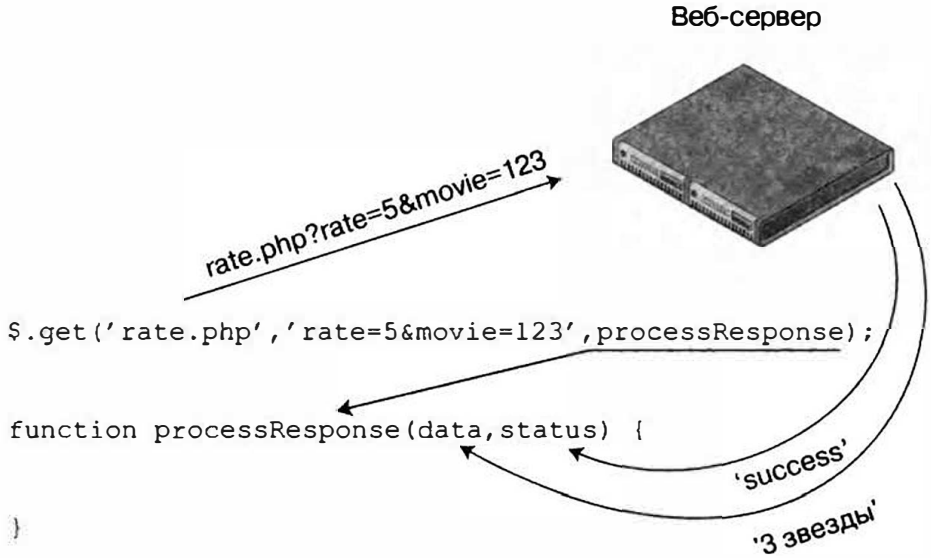

Рис. 13.7. На этой диаграмме вы можете видеть взаимодействие программы JavaScript с веб-сервером. Функция `get()` посылает информацию на сервер, тогда как функция обратного вызова `processResponse()` обрабатывает данные, возвращаемые сервером

Функция принимает в качестве аргументов данные, которые являются информацией, возвращенной сервером. Они могут быть текстом, документами формата HTML, XML или JSON. Строка 2 создает новую переменную, содержащую HTML-контент, который будет отображен на странице (например, «Ваш голос учтен»). В строках 3 и 4 в переменной `newHTML` сохраняется некоторый HTML-код, включая элементы `h2` и `p`. Ответ сервера не появляется вплоть до строки 5 — здесь ответ сервера (хранящийся в переменной `data`) добавляется в переменную `newHTML`. В нашем случае сервер возвращает строку со средним рейтингом фильма, например, `'3 звезды'`.

Наконец, строка 6 изменяет HTML-код на веб-странице с помощью jQuery-функции `html()` (см. раздел «Добавление содержимого на стра-

ницу» главы 4) путем замены содержимого элемента `div` с идентификатором `message` новым HTML-кодом. Результат похож на нижнюю часть рис. 13.6.

ПРИМЕЧАНИЕ

Если вы хотите установить рейтинговую систему на ваш сайт, рекомендуем отличный плагин библиотеки jQuery, который доступен по адресу: **www.wbotelhos.com/raty/**.

В данном примере функция обратного вызова была определена за пределами функции `$.get()`; однако вы можете воспользоваться анонимной функцией (см. раздел «Анонимные функции» главы 4), если хотите держать весь код Ajax в одном месте:

```
$.get('file.php', data, function(data,status) {
// код функции обратного вызова помещается сюда
});
```

Например, вы могли бы переписать строку 4 в первом коде данного раздела, чтобы использовать анонимную функцию:

```
$.get('rate.php', querystring, function(data) {
var newHTML;
newHTML = '<h2>Ваш голос учтен</h2>';
newHTML += '<p>Средний рейтинг для этого фильма ↵
составляет ';
newHTML += data + '.</p>';
$('#message').html(newHTML);
}); // конец get
```

Обработка ошибок

К сожалению, дела не всегда идут по плану. При использовании технологии Ajax для общения с сервером, что-то может пойти не так. Возможно, сервер временно недоступен или посетитель внезапно отключился от сети. Если подобное происходит, то функции `$.get()` и `$.post()` не выполнят своих задач, а посетитель об этом даже не узнает. Хотя такие проблемы возникают довольно редко, лучше подготовить-

ся и сообщить посетителям о том, что пошло не так, чтобы они смогли предпринять какие-нибудь действия (например, перезагрузить страницу, попробовать еще раз или вернуться позже).

Чтобы отреагировать на ошибку, вы просто добавляете функцию `.error()` в конец функции `$.get()` или `$.post()`. Базовая структура выглядит так:

```
$.get(url, data, successFunction).error(errorFunction);
```

Например, вы можете переписать строку 4 в первом коде предыдущего раздела так:

```
$.get('rate.php', querystring, processResponse) ↵
.error(errorResponse);
```

Затем создайте новую функцию `errorResponse`, которая информирует посетителя о возникновении проблемы. Например:

```
function errorResponse() {
var errorMsg = "Ваш голос не может быть обработан ↵
в данный момент.";
errorMsg += "Повторите попытку позже.";
$('#message').html(errorMsg);
}
```

В данном случае функция `errorResponse` выполняется только при возникновении ошибки, связанной с сервером или соединением.

ПРИМЕЧАНИЕ

Метод `.error()` не работает с функцией jQuery `.load()`, а также с запросом, посылаемым на другой сайт с использованием формата JSONP (см. раздел «Формат JSON» далее в этой главе).

Использование метода `get()` на практике

В этом руководстве вы воспользуетесь возможностями Ajax для отправки информации из формы авторизации. Когда посетитель вводит действительные имя пользователя и пароль, появляется сообщение о том, что он авторизован. Если же введенные данные неверны, то появится сообщение об ошибке — без загрузки новой страницы.

ПРИМЕЧАНИЕ

Для успешного выполнения этого руководства вам нужен сервер AMP (Apache, MySQL и PHP) для тестирования страниц. Обратитесь к тексту в рамке «Установка веб-сервера» в начале данной главы для получения информации о том, как установить тестовый сервер на ваш компьютер.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Получение XML-документа с сервера

XML является распространенным форматом обмена данными между компьютерами. Подобно HTML, XML позволяет использовать HTML-элементы для идентификации информации. Но, в отличие от HTML, вы вольны придумывать имена элементов, точно отражающие содержимое ваших данных. Например, простой XML-файл может выглядеть так:

```
<?xml version="1.0" ?>
<message id="234">
<from>Вова</from>
<to>Жанна</to>
<subject>Привет, Жанна</subject>
<content>Жанна, давай вместе пообедаем.</content>
</message>
```

Как видите, здесь есть главный элемент (называемый *корневым элементом*) `<message>` — эквивалент HTML-элемента `html` — и ряд других элементов, которые определяют значение каждого фрагмента данных.

При использовании технологии Ajax, вы можете работать с серверной программой, возвращающей XML-файл. Библиотека jQuery без проблем считывает и извлекает данные из XML-файла. При использовании функций `get()` или `post()`, в случае если сервер возвращает XML-файл, аргумент данных, передаваемый функции обратного вызова, будет содержать DOM XML-файла. Другими словами, библиотека jQuery станет трактовать XML-файл как еще один документ. Вы можете использовать инструменты селектора jQuery для доступа к данным внутри XML-файла.

Допустим, серверный XML-файл.*php* возвратил XML-файл, приведенный выше, и вы хотите извлечь текст, находящийся внутри элемента `content`. XML-файл становится возвращенной информацией,

так что функция обратного вызова может обрабатывать его. Вы можете использовать jQuery-функцию `find()` для поиска внутри XML-файла определенного элемента CSS, используя любой селектор jQuery. Например, найти элемент, класс, идентификатор, дочерний селектор (см. раздел «Специальные селекторы» главы 4) или фильтры jQuery (см. раздел «Фильтры jQuery» главы 4).

Например:

```
$.get('xml.php','id=234',processXML);
function processXML(data) {
var messageContent=$(data).
find('content').text();
}
```

Ключевым фрагментом здесь является код `$(data).find ('content')`, который поручает библиотеке jQuery выбрать каждый элемент `content` в переменной `data`. Поскольку в данном случае переменная `data` содержит возвращенный XML-файл, этот код приказывает jQuery искать элемент `content` внутри этого XML-файла.

Чтобы узнать больше о формате XML, посетите страницу **www.learn-xml-tutorial. com/xml-basics.cfm**. Если вы хотите найти сведения о jQuery-функции `find()`, то обратитесь к странице **api.jquery.com/find**.

Обзор

Начнем с формы, представленной на рис. 13.8. Она содержит поля для ввода имени пользователя и пароля. При передаче формы сервер проверяет правильность введенных данных. Если информация корректна, сервер авторизует посетителя.

Вы добавите Ajax к форме путем передачи информации с помощью объекта `XMLHttpRequest`. Сервер пошлет сообщение функции обратного вызова, которая удалит форму и выведет сообщение об успешном входе в систему в случае правильности введенных данных или сообщение об ошибке, если данные неверны.

Верстка кода

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров. Начальный файл содержит HTML-форму, готовую для добавления программного кода jQuery и Ajax.

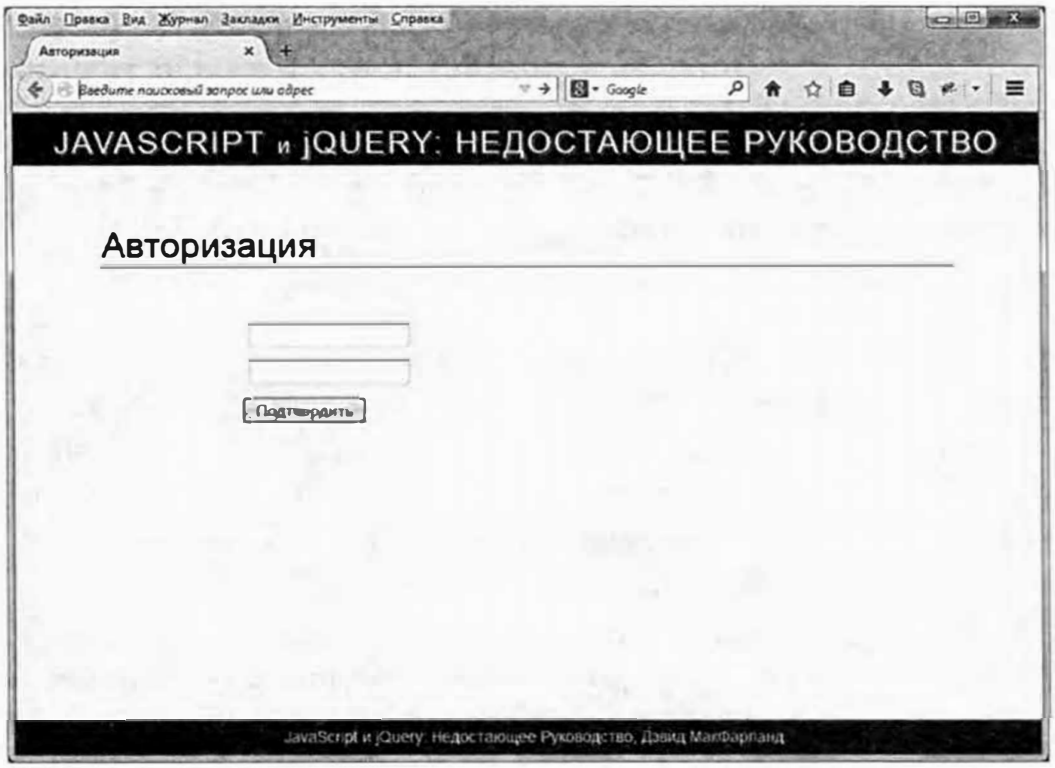

Рис. 13.8. Стандартная страница авторизации включает всего пару полей и кнопку передачи. Тем не менее нет смысла покидать ее после того, как посетитель вошел в систему. С помощью технологии Ajax вы сможете передать данные пользователя и сообщить ему, успешно ли он авторизовался

1. **В текстовом редакторе откройте файл *13_02.html* из каталога *глава13*.**

   Ссылка на файл библиотеки jQuery и функция `$(document).ready()` уже присутствует. Начнем с выбора формы и добавления к ней события `submit`.

2. **Щелкните мышью в пустой строке внутри функции `$(document).ready()` и введите следующий код:**

```
$('#login').submit(function() {

}); // конец submit
```

   Элемент `form` имеет идентификатор `login`, и, таким образом, селектор jQuery (`$('#login')`) выбирает данную форму, тогда как функция `submit()` добавляет обработчик события `submit`. Другими словами, когда посетитель попытается отправить форму, начнет выполняться функция, которую вы создадите.

   Следующий шаг заключается в сборе информации из формы и форматировании ее как строки запроса для отправки на сервер. Сделать это можно, находя каждый элемент формы, извлекая значение, вве-

денное посетителем, и конструируя строку запроса, соединяя вместе данные фрагменты информации. К счастью, jQuery-функция `serialize()` позаботится обо всех этих деталях.

3. **Нажмите клавишу Enter, чтобы создать новую пустую строку, и введите следующий код:**

```
var formData = $(this).serialize();
```

Эта строка начинается с создания новой переменной, содержащей данные формы, а затем применяет к форме функцию `serialize()`. Вспомните, что ключевое слово `$(this)` указывает на текущий элемент, то есть в данном случае на форму авторизации, что аналогично коду `$('#login')` (см. раздел «Ключевые слова `this` и `$(this)`» главы 4 для получения сведений о том, как работает ключевое слово `$(this)`). Функция `serialize()` (см. раздел «JQuery-функция `serialize()`») берет форму, извлекает имена и значения элементов формы и преобразует их в формат, пригодный для отправки на сервер.

А сейчас с помощью функции `$.get()` создадим объект `XMLHttpRequest`.

4. **Нажмите клавишу Enter, чтобы создать еще одну пустую строку, и введите следующий код:**

```
$.get('login.php',formData,processData);
```

Этот код передает три аргумента функции `$.get()`. Первый из них (`'login.php'`) представляет собой строку, определяющую, куда следует послать данные, в данном случае это серверный файл *login.php*. Второй аргумент является строкой запроса, содержащей данные, посылаемые на сервер, — данные для авторизации. Наконец, аргумент `processData` указывает на функцию обратного вызова, обрабатывающую ответ сервера. Сейчас мы создадим эту функцию.

5. **Добавьте еще одну пустую строку и введите:**

```
1 function processData(data) {
2
3 } // конец processData
```

Эти строки формируют оболочку функции обратного вызова; никакого программного кода внутри еще нет. Заметьте, что для функции предусмотрен только один аргумент (`data`), представляющий собой

ответ сервера. Серверная страница запрограммирована таким образом, чтобы возвращать слово `pass` при успешной авторизации или `fail` при отказе.

Иначе говоря, в зависимости от ответа сервера сценарий либо выдаст сообщение о том, что посетитель авторизовался, либо нет — это подходящее место для управляющей инструкции.

ПРИМЕЧАНИЕ

Серверная страница, используемая в данном руководстве, не является законченным сценарием входа в систему. Она дает ответ, когда получает правильные данные, однако это не то, что могло бы позволить вам действительно защитить сайт от несанкционированного доступа. Есть много способов сделать это, но многие из них требуют создания базы данных или различных настроек конфигурации для веб-сервера — данная тема находится за рамками настоящего руководства. Для знакомства с настоящим сценарием входа в систему на базе языка PHP, который также применяет библиотеку jQuery, посетите страницу: **www.wikihow.com/Create-a-Secure-Login-Script-in-PHP-and-MySQL**.

6. **Внутри функции `processData()` введите код, выделенный полужирным шрифтом:**

```
1 function processData(data) {
2 if (data === 'pass') {
3  $('.main').html('<p>Вы авторизованы!</p>');
4 }
5} // конец processData
```

Здесь строка 1 проверяет, возвратил ли сервер строку `'pass'`. Если да, то вход в систему осуществлен и появляется подтверждающее сообщение (строка 2). Форма находится внутри элемента `div` с классом `main`, так что код `$('.main').html('<p>Вы авторизованы!</p>')` заменит все, что находится в этом элементе `div`, новым содержимым. Иначе говоря, форма исчезнет, а вместо нее появится сообщение.

Для завершения добавим инструкцию `else`, чтобы дать посетителю возможность узнать о своей ошибке в случае предоставления неверной информации.

7. **Добавьте инструкцию `else` в функцию `processData()`, чтобы она выглядела так (дополнения выделены полужирным шрифтом):**

```
1 function processData(data) {
2 if (data=='pass') {
3 $('.main').html('<p>Вы авторизованы!</p>');
4 } else {
5 $('# formwrapper').prepend('<p id="fail"> ↵
Некорректная информация. Попробуйте еще раз</p>');
6 }
7 } // конец processData
```

Строка 5 содержит сообщение о том, что введены неверные данные. Обратите внимание на функцию `prepend()`. Как говорилось в разделе «Добавление содержимого на страницу» главы функция `prepend()` позволяет добавлять содержимое в начало элемента. При этом то, что уже находится там, не удаляется, а добавляется новое содержимое. В данном случае мы хотим оставить форму на месте, чтобы посетитель попробовал повторно войти в систему.

8. **Сохраните файл и откройте его в браузере.**

Вы должны использовать URL-адрес, вроде *http://localhost/глава13/13_02.html*, чтобы это руководство работало. См. врезку «Установка веб-сервера» в начале данной главы для получения дополнительных сведений по установке веб-сервера.

9. **Попробуйте авторизоваться на сайте.**

«Но постойте, вы ведь не сообщили мне имя пользователя и пароль!» — вероятно, подумали вы. В этом и смысл — посмотреть, что произойдет, если вы не авторизуетесь. Попробуйте войти в систему повторно: вы увидите, что сообщение «Некорректная информация» появилось вновь (рис. 13.9). Причина в том, что функция `prepend()` не удаляет первого сообщения, а просто добавляет его вторично. Непорядок!

У вас есть несколько способов решения данной проблемы. Вы можете вставить пустой элемент `div` под формой (`<div id="failMessage">`). Затем просто заменить HTML-код при неудавшейся авторизации. Однако в данном случае на странице нет пустого элемента `div`. Вместо этого вы используете простую управляющую инструкцию для проверки присутствия на странице сообщения об ошибке. Если оно присутствует, то нет необходимости добавлять его повторно.

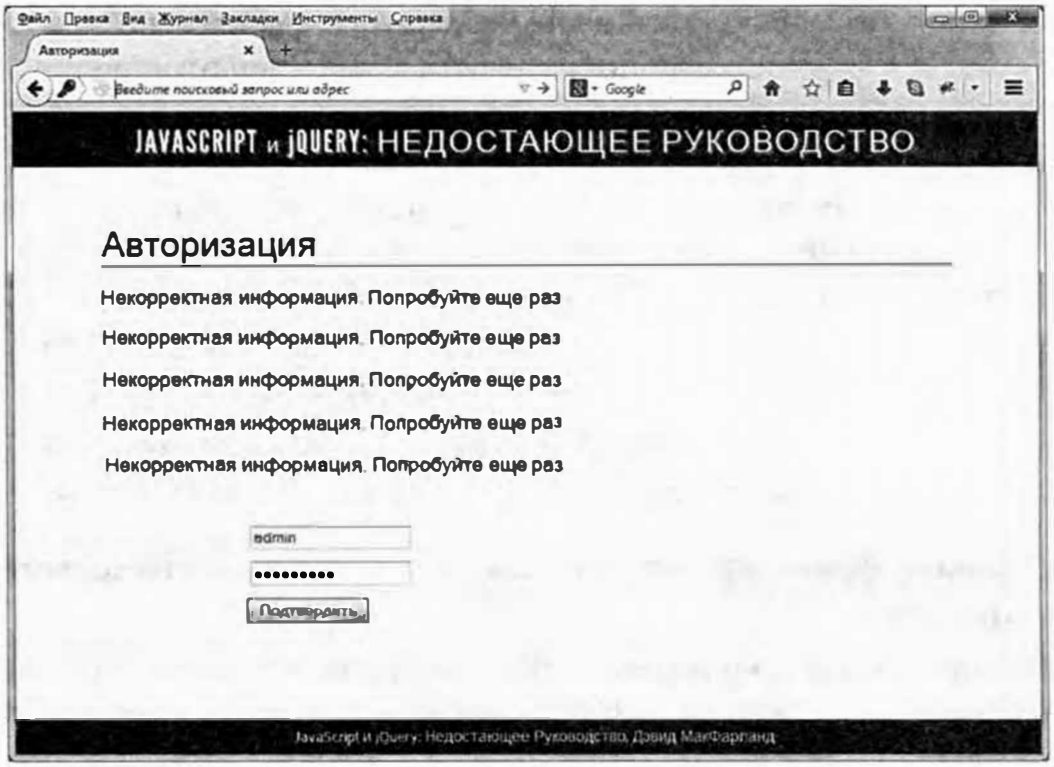

Рис. 13.9. Функция библиотеки jQuery `prepend()` добавляет HTML-код в уже существующий элемент. Она ничего не удаляет, поэтому в случае невнимательности вы рискуете получать одно и то же сообщение вновь и вновь

10. Добавьте еще одну управляющую инструкцию (строки 5 и 7):

```
1 function processData(data) {
2  if (data=='pass') {
3  $('.main').html('<p>Вы авторизованы!</p>');
4  } else {
5  if ($('#fail').length==0) {
6  $('#formwrapper).prepend('<p id="fail">Некорректная
7  информация. Попробуйте еще раз</p>');
8  }
9  }
10 } // конец processData
```

Заметьте, что текст сообщения об ошибке имеет идентификатор `fail`, так что вы можете использовать библиотеку jQuery для проверки наличия этого идентификатора на странице. Если такового нет, программа поместит сообщение на странице. Один из способов подобной проверки заключается в выборе искомого элемента сред-

ствами jQuery. Далее вы можете проверить атрибут `length` результирующей строки. Если совпадений не найдено, атрибут `length` имеет значение `0`. Другими словами, код `$('#fail')` пытается найти элемент с идентификатором `fail`. Если таковой не найден, то есть сообщения об ошибке на странице нет, указанный атрибут имеет значение `0`, управляющая инструкция оказывается истинной и программа выводит сообщение об ошибке. Как только оно оказалось на странице, управляющая инструкция становится ложной и сообщение об ошибке повторно не появляется.

Наконец, вам нужно сообщить браузеру, чтобы он не отправлял данные формы, — вы уже делали это с помощью технологии Ajax.

11. Добавьте фрагмент `return false;` в конец события `submit` (строка 15).

В окончательном виде сценарий должен выглядеть следующим образом:

```
1 $(document).ready(function() {
2 $('#login').submit(function() {
3  var formData = $(this).serialize();
4  $.post('login.php',formData,processData);
5  function processData(data) {
6  if (data=='pass') {
7  $('.main').html('<p>Вы авторизованы!</p>');
8  } else {
9  if ($('#fail').length === 0) {
10 $('#formwrapper').prepend('<p id="fail">Некорректная
11 информация. Попробуйте еще раз </p>');
12 }
13 }
14 } // конец processData
15 return false;
16 }); // конец submit
17 }); // конец ready
```

12. Сохраните файл и откройте его в браузере.

Попытайтесь войти в систему: имя пользователя — `007`, пароль — `secret`.

Окончательная версия этого руководства *готовый_13_02.html* находится в папке *глава13*.

ПРИМЕЧАНИЕ

Как указывалось в разделе «Функции `get()` и `post()`» данной главы, jQuery-функции `post()` и `get()` работают одинаково, даже несмотря на то, что библиотеке jQuery приходится выполнять два различных набора действий, чтобы Ajax-запрос обрабатывался корректно. Вы можете проверить это самостоятельно, просто поменяв в сценарии слово `post` на `get` (см. строку 4 в пункте 11). Серверный сценарий для данного руководства запрограммирован на принятие как GET-, так и POST-запросов.

Формат JSON

Еще один популярный формат для отправки данных с сервера — JSON (JavaScript Object Notation — представление объектов JavaScript). Он написан на языке JavaScript и похож на формат XML (см. врезку «Получение документа XML с сервера») тем, что является методом обмена данными. Данные в формате JSON — это строка, форматированная как объект JavaScript. Браузеры могут быстро и легко преобразовать данные в формате JSON в код JavaScript. С другой стороны, документ XML должен разбираться кодом JavaScript, что замедляет его работу и требует более длительного программирования. Поэтому формат JSON чаще применяется для обмена данными с помощью технологии Ajax.

Как создавать данные JSON, вы уже знаете. В сущности, JSON представляет собой литерал объекта JavaScript, или коллекцию пар имя/значение. Вот пример данных JSON:

```
{
"firstName" : "Федор",
"lastName" : "Семенов",
"phone" : "495-333-1212"
}
```

Знак { отмечает начало объекта JSON, тогда как } — его конец. Внутри скобок находятся пары имя/значение, например, `"firstName" : "Федор"`. Пары отделяются друг от друга запятой. В отличие от обычного объекта JavaScript имена свойств JSON и все строковые значения

должны заключаться в двойные кавычки. Другими словами, следующий код некорректен:

```
firstName : 'Федор'
```

А правильным является код:

```
"firstName" : "Федор"
```

Как и объекты JavaScript, пары имя/значение отделяются друг от друга запятой. Однако не следует ставить запятую после последней пары (иначе некоторые версии браузера Internet Explorer выдадут ошибку).

Пару имя/значение можно представить как переменную: имя соответствует имени переменной, а значение — ее содержимому. В предыдущем примере `"lastName"` действует как переменная, хранящая строку `"Семенов"` внутри себя.

Когда веб-сервер отвечает на Ajax-запрос, он может возвратить строку в формате JSON. Сервер не посылает JavaScript-код: он посылает текст, отформатированный как объект JSON. Использовать код JavaScript невозможно до тех пор, пока строка не конвертирована в объект JSON. К счастью, библиотека jQuery предусматривает специальную функцию `$.getJSON()`, которая заботится обо всех деталях. Функция `$.getJSON()` во многом похожа на функции `$.get()` и `$.post()`. Базовый код для нее выглядит так:

```
$.getJSON(url, data, callback);
```

Три аргумента функции `$.getJSON()` аналогичны аргументам функций `$.post()` и `$.get()` — URL-адрес серверной страницы, данные, посылаемые на серверную страницу, и имя функции обратного вызова. Разница в том, что функция `$.getJSON()` обрабатывает ответ сервера (который является просто строкой) и преобразует его (с помощью кода JavaScript) в объект JavaScript.

Другими словами, функция `$.getJSON()` работает как функции `$.post()` или `$.get()`, но данные, передаваемые функции обратного вызова, представляют собой объект JSON. Таким образом, чтобы применить функцию `$.getJSON()`, вам нужно узнать, как обрабатывать объект JSON функцией обратного вызова. Например, вы хотите сделать с помощью Ajax запрос по поводу какого-то контакта, содержащегося в серверном файле *contacts.php;* этот файл возвращает данные контакта в формате JSON (как в примере ранее). Базовый запрос выглядит так:

```
$.getJSON('contacts.php','contact=123',processContacts);
```

Данный код посылает строку запроса `contact=123` в файл *contacts.php*, который на основе этой информации находит запрашиваемый контакт в базе данных и извлекает необходимые сведения. Результат отсылается браузеру и передается функции обратного вызова `processContacts`. Базовая структура обратного вызова выглядит так:

```
function processContacts(data) {
}
```

Функция `processContacts()` имеет один аргумент `data`, который содержит объект JSON, полученный с сервера. Давайте посмотрим, как функция обратного вызова осуществляет доступ к информации объекта JSON.

Доступ к данным JSON

Есть два способа доступа к данным объекта JSON: *dot syntax* и *array notation*. *Dot syntax*, или *точечный синтаксис* (см. раздел «Небольшой урок, посвященный объектам» главы 2), указывает на определенное свойство объекта, что достигается путем вставки точки между именем объекта и названием свойства, доступ к которому нужно получить. Вы уже не раз наблюдали этот процесс при работе со свойствами различных объектов JavaScript вроде строк и массивов. Например, код `'abc'.length` предоставляет доступ к свойству `length` строки и возвращает число букв в строке `'abc'`, то есть `3`.

Предположим, вы создали переменную и храните в ней литерал объекта:

```
var bdate = {
"person" : "Павел",
"date" : "10/27/1980"
};
```

В этом случае переменная `bdate` содержит литерал объекта, так что если вы хотите получить имя человека, содержащееся в объекте, используйте следующий точечный синтаксис:

```
bdate.person // "Павел"
```

Для даты рождения:

```
bdate.date // "10/27/1980"
```

То же самое справедливо для объекта JSON, возвращаемого веб-сервером. Например, возьмем следующий экземпляр функции `$.getJSON()` и функцию обратного вызова:

```
$.getJSON('contacts.php','contact=123',processContacts);
function processContacts(data) {
}
```

Допуская, что сервер возвратил экземпляр объекта JSON (см. предыдущий раздел), этот объект JSON присваивается переменной `data` (аргументу функции обратного вызова `processContacts()`) так же, как если бы был выполнен следующий код:

```
var data = {
"firstName" : "Федор",
"lastName" : "Семенов",
"phone" : "495-333-1212"
};
```

Теперь внутри функции обратного вызова вы получаете доступ к значению `firstName`:

```
data.firstName // "Федор"
```

А теперь извлечем фамилию контакта:

```
data.lastName // "Семенов"
```

Представим, что вся суть этой маленькой программы Ajax заключается в том, чтобы извлечь контактную информацию и показать ее внутри элемента `div` с идентификатором `info`. Весь программный код может выглядеть так:

```
$.getJSON('contacts.php','contact=123',processContact
s); function processContacts(data) {
var infoHTML='<p>Контакт: ' + data.firstName;
infoHTML+=' ' + data.lastName + '<br>';
infoHTML+='Телефон: ' + data.phone + '</p>';
$('#info').html(infoHTML);
}
```

На выходе мы получили бы строки, добавленные на страницу, которые выглядят следующим образом:

```
Контакт: Федор Семенов
Телефон: 495-333-1212
```

Сложные объекты JSON

Вы можете создавать и более сложные коллекции данных, используя литералы объектов в качестве значений внутри объекта JSON, другими словами, литералы объектов, вложенные в другие литералы объектов. Вот пример: вам нужно, чтобы сервер послал контактную информацию для нескольких лиц, используя формат JSON. Вы посылаете запрос в файл *contacts.php* в виде строки, в которой указывается, сколько контактов требуется возвратить. Этот код может выглядеть так:

```
$.getJSON('contacts.php','limit=2',processContacts);
```

Код `limit=2` — это информация, посылаемая на сервер и указывающая число контактов, которое нужно возвратить. Веб-сервер возвратит два контакта. Допустим, контактная информация для первого лица совпадает с данными из предыдущего примера (`Федор Семенов`), а контактная информация для второго лица содержится еще в одном объекте JSON:

```
{
"firstName" : "Елена",
"lastName" : "Иванова",
"phone" : "495-555-5235"
}
```

Веб-сервер возвращает строку, представляющую единый объект JSON, который объединяет два объекта:

```
{
"contact1:" {
"firstName" : "Федор",
"lastName" : "Семенов",
"phone" : "495-333-1212"
```

```
},
"contact2:" {
"firstName" : "Елена",
"lastName" : "Иванова",
"phone" : "495-555-5235"
}
}
```

Предположим, функция обратного вызова принимает единственный аргумент `data` (например, `function processContacts(data)`). Тогда переменной `data` был бы присвоен объединенный объект JSON так же, как если бы был выполнен следующий код:

```
var data = {
"contact1:" {
"firstName" : "Федор",
"lastName" : "Семенов",
"phone" : "495-333-1212"
},
"contact2:" {
"firstName" : "Елена",
"lastName" : "Иванова",
"phone" : "495-555-5235"
}
}
```

Теперь получим доступ к первому объекту внутри функции обратного вызова:

```
data.contact1
```

Извлечем имя первого контакта:

```
data.contact1.firstName
```

Поскольку в данном случае вы хотите обработать несколько контактов, библиотека jQuery предлагает функцию `$.each()`, позволяющую проработать в цикле данные из каждого элемента объекта JSON. Базовая структура этой функции такова:

```
$.each(JSON,function(name,value) {
});
```

Вы передаете объект JSON и анонимную функцию функции `$.each()`. Анонимная функция получает имя и значение каждого элемента в объекте JSON. Вот как это выглядит в действии:

```
1 $.getJSON('contacts.php','limit=2',processContacts);
2 function processContacts(data) {
3     // создаем переменную с пустой строкой
4     var infoHTML='';
5
6     // обрабатываем каждый объект данных JSON
7     $.each(data,function(contact, contactInfo) {
8     infoHTML+='<p>Контакт: ' + contactInfo.firstName;
9     infoHTML+=' ' + contactInfo.lastName + '<br>';
10    infoHTML+='Телефон: ' + contactInfo.phone + '</p>';
11    }); // конец each()
12
13    // добавляем сгенерированный HTML-код на страницу
14    $('#info').html(infoHTML);
15 }
```

Проанализируем элементы кода.

1. **Строка 1 создает Ajax-запрос с данными (`limit=2`) и назначает функцию обратного вызова (`processContacts`).**
2. **Строка 2 создает функцию обратного вызова, которая принимает объект JSON, посланный сервером, и хранит его в переменной (`data`).**
3. **Строка 4 создает пустую строку. Ее заполнит HTML-код, который в итоге будет добавлен на страницу.**
4. **Строка 7 представляет собой метод `$.each()`, который просматривает объекты данных JSON.**

Метод `$.each()` принимает объект JSON в качестве первого аргумента (`data`) и анонимную функцию в качестве второго. Процесс

показан на рис. 13.10. По сути, для каждого из основных объектов (в данном примере `contact1` и `contact2`) анонимная функция получает имя объекта как строку (аргумент `contact` в строке 7) и его значение (аргумент `contactInfo`). В этом случае переменная `contactInfo` получит литерал объекта, содержащий информацию о контакте.

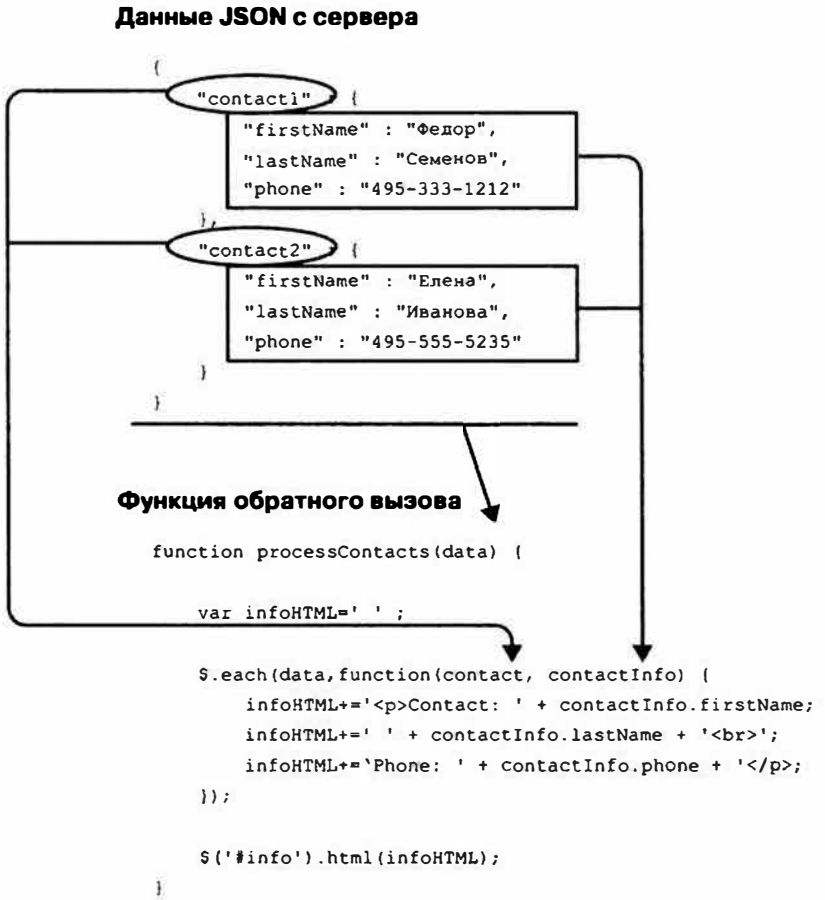

Рис. 13.10. Вы можете использовать jQuery-функцию `$.each()` для просмотра и обработки вложенных объектов JSON. С ее же помощью можно обрабатывать массивы. Чтобы больше узнать об этой полезной функции, посетите страницу: **docs.jquery.com/Utilities/jQuery.each#objectcallback**

5. **Строки 8—10 извлекают информацию из одного контакта.**

   Помните, что функция `$.each()` представляет собой цикл, так что строки 8—10 будут выполнены дважды — по разу для каждого контакта.

6. **Строка 14 обновляет страницу путем добавления нового HTML-кода.**

В результате получится следующий HTML-код:

```
<p>Контакт: Федор Семенов <br>
Телефон: 495-333-1212</p>
<p>Контакт: Елена Иванова<br>
Телефон: 495-555-5235</p>
```

Введение в JSONP

Из соображений безопасности запросы Ajax ограничены тем же доменом. То есть страница, посылающая запрос Ajax, должна находиться на том же сервере, что и страница, на него отвечающая. Это политика браузера для предотвращения несанкционированной связи одного сайта с другим (например, с сайтом вашего банка). Однако это ограничение можно обойти. Хотя веб-браузер не может послать XMLHTTP запрос на другой сайт, он может загружать информацию с других сайтов, включая изображения, таблицы стилей и внешние файлы JavaScript.

Формат JSONP (JSON with padding) предоставляет возможность получать информацию с другого сайта. По сути, вместо создания запроса Ajax на другой сайт, вы загружаете сценарий, содержащий код JSON. Другими словами, это похоже на создание ссылки на внешний файл JavaScript, находящийся на другом сайте.

Однако вы не можете запросить любую информацию с другого сайта. Чтобы код JSONP сработал, другой сайт должен быть настроен на ответ в формате JSONP. Большинство сайтов не настроены на отправку информации таким способом, но многие крупные сервисы, например, Google Maps, Twitter, Flickr, Facebook, Netflix и YouTube предлагают интерфейс программирования приложений (API, Application Programming Interface), позволяющий вам запрашивать данные, вроде карты, фотографии, рецензии на фильм и т. д., используя функцию библиотеки jQuery `$.getJSON()` (рис. 13.11).

Добавление фида сервиса Flickr на ваш сайт

Сервис Flickr — это популярный сайт для обмена фотографиями. Он существует уже несколько лет и содержит миллионы фотографий.

Многие сайты содержат фотографии, либо созданные владельцем сайта, либо взятые из группы Flickr (группа — это коллекция фотографий, добавленных разными людьми, но посвященных одной теме, вроде веб-дизайна, пейзажных фотографий и т. д.).

Рис. 13.11. Хотя технология Ajax ограничена запросами информации только из того же домена, обходной путь JSONP позволяет извлекать данные JSON с другого сайта, запрашивая с этого сайта файл JavaScript. Данная техника позволяет получить доступ к сообщениям сервиса Twitter, картам системы Google и фотографиям с сайта Flickr (на рисунке) и внедрять их прямо в ваши веб-страницы

Сайт Flickr предоставляет вам два способа получения фотографий и относящейся к ним информации. Самым мощным, но и самым сложным, методом является использование API сервиса Flickr для поиска фотографий. Этот метод требует от вас регистрации в системе Flickr и получения специального API *ключа* (строки чисел и символов, которые вас идентифицируют). Также этот метод требует сложного программного кода. Проще всего использовать сервис Flickr Feed. Фид — это способ держать людей в курсе относительно изменений на сайте. Например, вы, вероятно, видели сайты с RSS-лентами, которые позволяют вам просмотреть последние новости и информацию с этого сайта. Сайт Flickr предлагает подобный сервис для своих фотографий — вы можете получить 20 последних фотографий от конкретного пользователя или группы.

В данном разделе вы будете использовать фиды для получения коллекции фотографий с сервиса Flickr и отображения их на своей вебстранице, а также учиться применять jQuery-метод `$.getJSON()` для извлечения данных JSONP с другого сайта.

Построение URL-адреса

Сервис Flickr предоставляет несколько различных URL-адресов для получения доступа к фидам для разных типов фотографий (за полным списком обратитесь на сайт: **www.flickr.com/services/feeds/**). Например, вы используете адрес: *http://api.flickr.com/services/feeds/photos_public.gne* для получения доступа к фотографиям с конкретных аккаунтов (например, вашего, если он у вас есть) и ссылку для извлечения фотографий из конкретной группы (например, группа «Веб-дизайн», которая содержит фотографии и изображения для вдохновения).

Как только вы определили, какой тип фида вам нужен, и URL-адрес, который ему соответствует, вам нужно добавить дополнительную информацию для извлечения интересующих вас фотографий. Для этого вы добавляете строку запроса с несколькими фрагментами информации. (Как вы помните из раздела «Строка запроса» данной главы, строка запроса представляет собой знак «?» в конце URL-адреса, за которым следуют одна или несколько пар имя/значение: *http://api.flickr.com/services/feeds/groups_pool.gne?id= 25053835@N03&&format=json*, например.)

- **Добавьте один или несколько идентификаторов.** Для выбора фотографий из конкретной группы или пользовательского аккаунта, вы добавляете символы `id=`, за которыми следует номер аккаунта пользователя или группы. Например, для доступа к фиду группы «Веб-дизайн» вы используете фид группы и ее идентификатор так:

  ```
  http://api.flickr.com/services/feeds/groups_pool. ↵
  gne?id=37996591093@N01
  ```

  Для пользовательских фидов Flickr вы используете фид общедоступных фотографий и один или несколько идентификаторов. Например, для извлечения фотографий из Смитсоновского института (у которого есть собственный аккаунт Flickr) и Библиотеки Конгресса вы можете использовать идентификатор и фид общедоступных фотографий так:

  ```
  http://api.flickr.com/services/feeds/photos_public. ↵
  gne?ids=8623220@N02,25053835@N03
  ```

Чтобы использовать несколько идентификаторов, отделяйте их запятой. Заметьте, что вы можете использовать несколько идентификаторов только для доступа к индивидуальным аккаунтам: вы не можете использовать фид группы и извлекать фотографии из нескольких групп.

> **СОВЕТ**
>
> Если вы знаете чье-то имя пользователя в системе Flickr, то вы можете узнать его идентификатор, используя сайт **idgettr.com**.

- **Добавьте формат JSON.** Сервис фидов Flickr очень гибок и может возвращать информацию о фотографии во многих форматах от RSS до Atom, CSV и JSON. Чтобы сообщить сервису Flickr, что вы хотите получить данные в формате JSON, вам необходимо добавить в строку запроса фрагмент `&format=json`. Например, чтобы получить фид Смитсоновского института в формате JSON, вы используете такой URL-адрес:

  ```
  http://api.flickr.com/services/feeds/photos_public. ↵
  gne?ids=25053835@N03&format=json
  ```

  Попробуйте ввести этот адрес в адресную строку браузера (вы можете просто скопировать и вставить его из файла *flickr_json.txt*, находящегося в папке *глава13*). Вы увидите большое количество данных, точнее литерал объекта, содержащий большой объем информации. Вот, что вы получаете от сервиса Fickr, когда используете функцию `$.getJSON()` (описанную в разделе «Формат JSON» ранее в главе). Вам ужен код JavaScript, чтобы разделить этот объект, а затем использовать его для построения небольшой галереи изображений. (Структура JSON-фида сервиса Flickr рассматривается в следующем разделе, а разделение фида для его использования показано в первом руководстве данной главы.)

- **Добавьте к URL-адресу обратный вызов JSONP.** Наконец, чтобы страница вашего сайта успешно запросила данные в формате JSON с другого сайта, вы должны добавить последний фрагмент к URL-адресу: `&jsoncallback=?`. Помните, что по соображениям безопасности вы не можете просто послать XMLHTTP запрос в другой домен. Чтобы обойти это ограничение, часть `&jsoncallback=?` сообщает сервису Flickr, что вы хотите получать данные в формате JSONP и позволяет jQuery-функции `$.getJSON()` обработать этот запрос так, как если бы браузер просто запрашивал внешний файл

JavaScript. Другими словами, для получения последних фотографий Смитсоновского института с сервиса Flickr, вы должны передать функции `$.getJSON()` URL-адрес:

```
http://api.flickr.com/services/feeds/photos_public ↵
.gne?ids=25053835@N03&format=json&jsoncallback=?
```

Поиск конкретных фотографий

В системе Flickr разрешено присвоить любому фото *метку*, содержащую одно или более слов. Метка — это слово или короткая фраза, описывающая элемент фотографии. Например, вы можете отметить особенно яркую фотографию заката словом «закат». Любая фотография может иметь несколько меток, так что вы можете отметить фотографию заката словами «закат, оранжевый, пляж».

Сервис фидов Flickr предусматривает настройки, позволяющие вам искать конкретные метки:

- **`tags`.** Добавьте в URL-адрес фрагмент `tags` (метки) и одно или несколько разделенных запятыми ключевых слов, например, `&tags=fireworks,night`. Чтобы найти фотографии фейерверков в ночном небе, вы можете использовать следующий URL-адрес:

  ```
  https://api.flickr.com/services/feeds/photos_public ↵
  .gne?tags=fireworks,night&format=json&jsoncallback=?
  ```

- **`tagmode`.** Обычно, когда вы ищите набор меток, система Flickr извлекает фотографии, соответствующие всем этим меткам. Например, вы добавили в фид `?tags=chipmunks,baseball,winter`. Этот код находит только фотографии бурундуков, играющих в бейсбол зимой. Если вам нужны фотографии, связанные с бурундуками или бейсболом или зимой (другими словами, соответствующие как минимум одной из меток), добавьте в URL-адрес часть `&tagmode=any`. Например:

  ```
  https://api.flickr.com/services/feeds/photos_public ↵
  .gne?tags=chipmunk,baseball,winter&tagmode= ↵
  any&format=json&jsoncallback=?
  ```

Комбинирование параметров

Разрешается комбинировать параметры поиска, чтобы уточнить выбор фотографий. Например, вы можете объединить идентификатор

с меткой, чтобы найти фотографии конкретного автора и конкретной вещи. Например, вы и пара ваших друзей любите снимать бурундуков и публиковать фотографии на сервисе Flickr, и вы хотите получить фид с 20 последними фото бурундуков, которые сделали вы и ваши друзья. Это несложно сделать, отфильтровав фид с помощью одного или нескольких тегов:

```
https://api.flickr.com/services/feeds/photos_public. ↵
gne?ids=25053835@N03,8623220@N02&tags= ↵
chipmunk&format=json&jsoncallback=?
```

Использование метода `$.getJSON()`

Использование метода `$.getJSON()` для извлечения фотофида с сервиса Flickr работает так же, как и извлечение данных JSON с вашего собственного сайта. Базовая структура функции та же самая. Например, вот код, извлекающий фид Flickr Смитсоновского института:

```
1 var flickrURL = "https://api.flickr.com/services/ ↵
feeds/photos_public.gne?ids=25053835@N03& ↵
format=json&jsoncallback=?";
2 $.getJSON(flickrURL, function(data) {
3 // какие-либо действия над возвращенными данными JSON
4 }); // конец get
```

В данном примере строка 1 создает переменную `flickrURL` и сохраняет в ней URL-адрес (используя правила, описанные в предыдущем разделе). Строка 2 посылает URL-адресу запрос Ajax и подготавливает анонимную функцию для обработки данных. После отправки Ajax-запроса код извлекает данные с сервера — в нашем примере эти данные посылаются анонимной функции и хранятся в переменной `data`. Вы узнаете, как обрабатывать данные далее в главе, но сначала вы должны понять, как выглядят данные JSON сервиса Flickr.

JSON-фид сервиса Flickr

Как обсуждалось в разделе «Формат JSON» ранее в данной главе, данные JSON — это просто объектная константа JavaScript, которая может быть простой, как:

```
{
"firstName" : "Федор",
"lastName" : "Семенов"
}
```

В этом коде переменная `firstName` действует как ключ со значением `Федор` — простой строкой. Однако значением может быть другой объект (см. рис. 13.10 ранее в главе), так что у вас может получиться сложная вложенная структура, напоминающая матрешку. Вот, что представляет собой фид JSON сервиса Flickr. Вот небольшой фрагмент одного такого фида, который содержит информацию о двух фотографиях:

```
1 {
2 "title": "Uploads from Smithsonian Institution",
3 "link": "http://www.flickr.com/photos/smithsonian/",
4 "description": "",
5 "modified": "2011-08-11T13:16:37Z",
6 "generator": "http://www.flickr.com/",
7 "items": [
8 {
9     "title": "East Island, June 12, 1966.",
10    "link": "http://www.flickr.com/photos/ ↵
      smithsonian/5988083516/",
11    "media": {"m":"http://farm7.static.flickr. ↵
      com/6029/5988083516_bfc9f41286_m.jpg"},
12    "date_taken": "2011-07-29T11:45:50-08:00",
13    "description": "Short description here",
14    "published": "2011-08-11T13:16:37Z",
15    "author": "nobody@flickr.com (Smithsonian ↵
      Institution)",
16    "author_id": "25053835@N03",
17    "tags": "ocean birds redfootedbooby"
18 },
19 {
20    "title": "Phoenix Island, April 15, 1966.",
21    "link": "http://www.flickr.com/photos/ ↵
```

```
      smithsonian/5988083472/",
22    "media": {"m":"http://farm7.static.flickr. ↵
      com/6015/5988083472_c646ef2778_m.jpg"},
23    "date_taken": "2011-07-29T11:45:48-08:00",
24    "description": "Another short description",
25    "published": "2011-08-11T13:16:37Z",
26    "author": "nobody@flickr.com (Smithsonian ↵
      Institution)",
27    "author_id": "25053835@N03",
28    "tags": ""
29 }
30 ]
31 }
```

Объект JSON системы Flickr содержит общую информацию о данном фиде в начале: `"title"`, `"link"` и т. д. Элемент `"title"` (строка 2) — это название фида. В данном случае, `"Uploads from Smithsonian Institution"`, элемент `"link"` (строка 3) указывает на главную страницу Смитсоновского института на сервисе Flickr. Вы можете использовать эту информацию в качестве заголовка на странице с фотографиями.

Для доступа к этой информации, вы используете точечный синтаксис, описанный в разделе «Небольшой урок, посвященный объектам» главы 2. Допустим, вы использовали код из предыдущего раздела: анонимная функция, обработавшая данные, хранит ответ JSON в переменной `data` (строка 2). Чтобы получить доступ к названию фида, вы получаете доступ к свойству `"title"` объекта `"data"` так:

```
data.title
```

Самая важная часть фида — это свойство `"items"` (строка 7), содержащее дополнительные объекты, каждый из которых включает информацию о фотографии. Например, строки 8–18 предоставляют информацию об одной фотографии, а строки 19–29 — о другой. В рамках каждого объекта вы найдете другие свойства, например, название фотографии (строка 9), ссылку на страницу фотографии на сайте Flickr (строка 10), дату создания фотографии (строка 12), описание (`"Short description here"` (краткое описание) в строке 13 [кураторы Смитсоновского института явно поленились в тот день]) и т. д.

Другой важный элемент каждой фотографии — это объект "media". Например,

```
{
"m":"http://farm7.static.flickr.com/6029/5988083516_ ↵
bfc9f41286_m.jpg"
}
```

Часть "m" означает *medium*, и ее значением является URL-адрес фотографии. Фотографии на сервисе Fickr часто бывают доступны в разных размерах, например, *medium* (средний), *thumbnail* (эскиз) и *small* (небольшое квадратное изображение). Если вы хотите отобразить изображение на странице, то URL-адрес — это то, что вам нужно. Вы можете использовать его, чтобы вставить на страницу элемент img и указать на данную фотографию на сервере Flickr. Как это сделать, вы узнаете в следующем руководстве.

Добавление на сайт изображений сервиса Flickr на практике

В данном руководстве вы получите фотофид Смитсоновского института, отобразите эскизы фотографий и добавите ссылки на каждое изображение, чтобы посетитель смог щелкнуть по эскизу и перейти на страницу с фотографией на сайте Flickr.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *13_03.html* из каталога *глава13*.**

   Вы начнете с создания нескольких переменных и сохранения компонентов URL-адреса, необходимых для общения с сервисом Flickr.

2. **Щелкните мышью в пустой строке внутри функции `$(document).ready()` и введите следующий код:**

   ```
   var URL = "https://api.flickr.com/services/feeds/ ↵
   photos_public.gne?jsoncallback=?";
   ```

Этот URL-адрес указывает на публичный фид и включает фрагмент `?jsoncallback=?`, который сообщает сервису Flickr о необходимости отправить данные в формате JSONP. Другими словами, этот код заставляет сервис Flickr отправить JavaScript-файл с информацией о фотографии.

Далее вы создадите объект, содержащий дополнительную информацию для сервиса Flickr.

Каждая переменная является частью URL-адреса, как было сказано в разделе «Построение URL-адреса» ранее в данной главе. Разделение каждого фрагмента на отдельные переменные упрощает настройку данного кода. Например, если вы захотите получить фотофид от другого пользователя системы Flickr, просто поменяйте имя идентификатора.

СОВЕТ

Если у вас есть аккаунт на сервисе Flickr, то подставьте собственный идентификатор. Если вы не знаете его, посетите сайт: **idgettr.com**.

Далее вы соберете эти переменные вместе, чтобы построить полный URL-адрес.

3. **Добавьте еще одну строку кода после тех, которые вы только что ввели:**

```
var searchInfo = {

};
```

Эта строка представляет собой пустой литерал объекта JavaScript. Как вы узнали в разделе «Формат JSON» ранее в главе, вы можете передать дополнительную информацию вместе с запросом Ajax. Сервер сможет затем использовать эту информацию, чтобы изменить свой ответ. В данном случае вам нужно предоставить идентификатор пользователя Flickr, чтобы найти только сделанные им фотографии.

4. **Внутри объекта добавьте пару имя/значение (код выделен полужирным шрифтом):**

```
var searchInfo = {
id : "25053835@N03"
```

```
};
```

API-интерфейс сервиса Flickr ожидает получить идентификатор конкретного пользователя. В данном случае идентификатор принадлежит аккаунту Смитсоновского института. Обратите внимание на то, что вам необходимо указывать имя свойства в нижнем регистре — `id`. Если вы хотите получить фотофид другого пользователя, просто замените значение идентификатора (опять же, если у вас есть собственный аккаунт, вы можете использовать свой идентификатор).

Вам также необходимо дать сервису Flickr задание отправить обратно данные JSON.

> **СОВЕТ**
>
> Если вы хотите извлечь фотографии группы, например, «Вебдизайн», то измените URL-адрес в шаге 2 на *https://api.flickr.com/services/feeds/groups_pool.gne?jsoncallback=?*, и укажите значение идентификатора соответствующей группы.

5. **Добавьте запятую после строки с идентификатором, нажмите клавишу Enter и введите код `format : "json"`:**

```
var searchInfo = {
id : "25053835@N03",
format : "json"
};
```

Если вы не добавите этот фрагмент, то получите в ответ данные в формате XML.

Теперь пришло время приступить к части Ajax и использовать функцию `$.getJSON()`.

6. **После только что добавленной строки введите:**

```
$.getJSON(ajaxURL, searchInfo, function(data) {
});  // конец get JSON;
```

Здесь создается оболочка функции `$.getJSON()`, которая будет связываться с URL-адресом, созданным на шаге 2, посылать критерии поиска, указанные в шагах 3–5, и получать данные с сервиса Flickr. Эти данные передаются анонимной функции и хранятся

в переменной data. Вы сможете взять эти данные и использовать их на странице. Сначала вы получите название фида и замените им присутствующий в данный момент на странице элемент h1.

7. **Добавьте строку кода, выделенную полужирным шрифтом:**

```
$.getJSON(ajaxURL,function(data) {
$('h1').text(data.title);
}); // конец get JSON;
```

Это простой селектор jQuery $('h1') и функция .text(), с помощью которых присутствующий на странице в данный момент элемент h1 выделяется, а содержащийся в нем текст заменяется. Фид JSON хранится в переменной data. Для доступа к ее компонентам вы используете точечный синтаксис (см. раздел «Небольшой урок, посвященный объектам» главы 2), так что код data.title извлекает название фида. Если вы сейчас сохраните страницу и просмотрите ее в браузере, то увидите заголовок Uploads from Smithsonian Institution.

Далее вы создадите новую переменную, которая будет содержать строку с HTML-кодом, добавляемым на эту веб-страницу.

8. **Добавьте строку кода, выделенную полужирным шрифтом:**

```
$.getJSON(ajaxURL,function(data) {
$('h1').text(data.title);
var photoHTML = '';
}); // конец get JSON;
```

Сейчас эта строка пуста, но скоро вы добавите HTML-код, необходимый для отображения возвращенных с сервиса Flickr фотографий. Для создания этого HTML-кода вы с помощью цикла обрабатываете массив объектов items, возвращенный фидом Flickr. Для каждого фотофида вы добавите дополнительный HTML-код в переменную photoHTML.

9. **Добавьте строку кода, выделенную полужирным шрифтом:**

```
$.getJSON(ajaxURL,function(data) {
$('h1').text(data.title);
varphotoHTML='';$.each(data.items,function(i,photo)
{
```

```
}); // конец each
});  // конец get JSON;
```

Вы читали о функции `.each()` в разделе «Работа с каждым элементом выборки» главы 4. Эта функция используется для проработки выборки jQuery. Функция `$.each()` подобна ей. Это функция, которую вы можете использовать для циклической обработки либо массива (см. раздел «Массивы» главы 2), либо серии объектов. Вы передаете функции `$.each()` массив или объектную константу и анонимную функцию. Затем функция `$.each()` прорабатывает массив или объектную константу и применяет анонимную функцию по одному разу к каждому элементу. Эта анонимная функция получает два аргумента (`i` и `photo` в коде на данном шаге), которые являются переменными, содержащими индекс элемента и сам элемент. Индекс — это порядковый номер элемента в цикле, он работает подобно индексу в массиве (см. раздел «Доступ к элементам массива» главы 2). Например, первый элемент в цикле имеет индекс 0. Второй аргумент (в данном примере `photo`) — это конкретный объект фотографии, содержащий название, описание, URL-адрес и т. д., как описано в предыдущем разделе.

В случае с фидом сервиса Flickr часть `data.items` представляет собой объекты-фотографии в фиде JSON, так что функция `$.each()` передает объект для каждой фотографии анонимной функции в переменной `photo`. Другими словами, данный код обходит каждую фотографию фида, а затем совершает какие-нибудь действия. В данном случае вы просто создаете серию эскизов, которые ссылаются на страницы с фотографиями на сервисе Flickr. Целью будет создание простого HTML-кода для отображения каждого изображения и включения ссылки. Например,

```
<span class="image">
<a href="http://www.flickr.com/photos/ ↵
smithsonian/5988083516/">
<img src="http://farm7.static.flickr. ↵
com/6029/5988083516_bfc9f41286_s.jpg">
</a>
</span>
```

Чтобы создать этот код, требуются только два фрагмента информации: URL-адрес фотографии на сайте Flickr и путь к файлу. Вы просто составите длинную строку, похожую на вышеприведенный

HTML-код, заменив URL-адрес и путь к изображению для каждой фотографии.

10. **Внутри функции `$.each()` добавьте код, выделенный полужирным шрифтом:**

```
$.each(data.items,function(i,photo) {
photoHTML += '<span class="image">';
photoHTML += '<a href="' + photo.link + '">';
photoHTML += '<img src="' + photo.media.m + '"></a></span>';
}); // конец each
```

Этот код начинается с добавления открывающего тега `<span>` к переменной `photoHTML`. Каждая последующая строка добавляет больше данных в переменную (об операции += и принципе ее работы читайте в разделе «Изменение значений в переменных» главы 2). Ключевыми элементами здесь являются: `photo.link` и `photo.media.m`. Если вы посмотрите на код JSON в разделе «JSON-фид сервиса Flickr» данной главы, то увидите, что каждая фотография имеет различные свойства, вроде *title* (название фотографии) и *description* (описание). Свойство *link* указывает на страницу фотографии на сайте Flickr.com, а объект *media* имеет свойство *"m"*, которое содержит путь к изображению среднего размера. Этот код создает HTML, описанный на шаге 6. Теперь вам нужно просто добавить этот код на страницу.

11. **Добавьте код, выделенный полужирным шрифтом:**

```
$.each(data.items,function(i,photo) {
photoHTML = '<span class="image">';
photoHTML += '<a href="' + photo.link + '">';
photoHTML += '<img src="' + photo.media.m + '"></a></span>';
}); // конец each
$('#photos').append(photoHTML);
```

Обратите внимание на то, что эта строка находится за пределами цикла. Вам не нужно добавлять HTML-код на страницу до тех пор, пока вы завершите его создание внутри цикла. Часть `$('#photos')`

выбирает уже существующий на странице элемент `div`, а функция `append()` (см. раздел «Добавление содержимого на страницу» главы 4) добавляет HTML-код в конец этого элемента `div`.

12. Сохраните страницу и просмотрите ее в браузере.

Вы должны увидеть 20 фотографий, загруженных на страницу. (Если вы ничего не видите, проверьте ваш код и используйте консоль ошибок своего браузера (см. раздел «Отслеживание ошибок» главы 1) для поиска и устранения неполадок.) Проблема заключается в том, что фотографии отличаются по размеру и не выглядят структурированными на странице. Это потому, что фид сервиса Flickr предоставляет пути только к изображениям среднего размера.

Сервис Fickr также располагает симпатичными квадратными эскизами всех имеющихся фотографий. Отображение одинаковых по размеру фотографий на странице создает красивую упорядоченную презентацию. К счастью, запросить эти эскизы не сложно. Сервис Flickr использует следующие правила наименования фотографий: путь к изображению среднего размера выглядит так: *http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_m.jpg*, а путь к эскизу той же фотографии так: *http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_s.jpg*. Единственное отличие — в конце имени файла: *_m* указывает на средний размер, *_s* — на маленький квадратный эскиз (75×75 пикселов), *_t* — это небольшое изображение, самая длинная сторона которого не превышает 100 пикселов, *_o* указывает на оригинальный размер (изначальный размер изображения при загрузке на сайт Flickr), и *_b* — крупное изображение (максимум 1024 пиксела в высоту или длину). Просто изменив имя файла (заменив *_m* на *_s*, например), вы можете отобразить файл другого размера. К счастью, язык JavaScript предоставляет удобный способ быстрой замены символов в строке.

13. Измените часть кода `photo.media.m` на `photo.media.m.replace('_m','_s')`. Итоговый код должен выглядеть так:

```
$(document).ready(function() {

var URL = "https://api.flickr.com/services/feeds/ ↵
photos_public. gne?jsoncallback=?";

var searchInfo = {

id : "25053835@N03",
```

```
format : "json"
};
$.getJSON(URL,searchInfo,function(data) {
$('h1').text(data.title);
var photoHTML = '';
$.each(data.items,function(i,photo) {
photoHTML += '<span class="image">'; photoHTML += ↵
<a href="' + photo.link + '">'; photoHTML += '<img ↵
src="' + photo.media.m.replace('_m','_s') + '"> </a></
span>';
}); // конец each
$('#photos').append(photoHTML);
}); // конец get JSON
}); // конец ready
```

Функция JavaScript `replace()` (обсуждается в разделе «Замена текста» главы 16) работает со строками и принимает два аргумента — строку, которую нужно найти (в данном случае `'_m'`) и строку, которой нужно заменить (`'_s'`).

14. Сохраните страницу и просмотрите ее в браузере.

Теперь вы должны увидеть 20 аккуратно выровненных квадратных эскиза (см. рис. 13.11). Щелкните по эскизу, чтобы увидеть крупное изображение на сайте Flickr. Итоговая версия данного руководства — файл *готовый_ 13_03.html* — находится в каталоге *глава13*.

ПРИМЕЧАНИЕ

Фид сервиса Flickr предоставляет максимум 20 изображений. Чтобы узнать, как отобразить 10 изображений, обратитесь к файлу *готовый_13_03_02.html* в папке *глава13*.

Глава 14
СОЗДАНИЕ ПРИЛОЖЕНИЯ «СПИСОК ДЕЛ»

Библиотека jQuery и плагин jQuery UI предусматривают инструменты, позволяющие всего за несколько шагов создать профессионально выглядящее веб-приложение. Библиотека jQuery заботится о деталях, касающихся выделения элементов страницы, добавления новых элементов и обновления DOM. Плагин jQuery UI предоставляет великолепные виджеты, взаимодействия и эффекты, которые решают многие часто возникающие проблемы, связанные с дизайном пользовательского интерфейса. Благодаря этим двум библиотекам вы можете пропустить этап трудоемкого программирования и перейти непосредственно к кодированию динамического интерактивного приложения. В этой главе мы подробно рассмотрим процесс создания простого менеджера задач.

Обзор приложения

Ваше приложение для создания списка дел должно давать пользователям возможность:

- **Добавлять новую задачу в список дел.** Вы можете обеспечить такую возможность, добавив кнопку jQuery UI (№ 1 на рис. 14.1), щелчок по которой приводит к открытию диалогового окна jQuery UI.
- **Помечать задачу как выполненную.** Каждый элемент списка дел будет иметь флажок слева от названия задачи (№ 2 на рис. 14.1). Установка флажка автоматически удалит элемент из списка «Предстоящие задачи» и переместит его в список «Выполненные задачи».
- **Перетаскивать элементы из списка «Предстоящие задачи» в список «Выполненные задачи» и наоборот** (№ 3 на рис. 14.1). Придать задаче статус выполненной можно не только посредством флажка, но и с помощью виджета Sortable плагина jQuery UI. Используя его, вы дадите пользователям возможность просто перетащить эле-

менты в список «Выполненные задачи», чтобы отметить задачу как завершенную. Кроме того, вы можете позволить посетителям перетаскивать завершенную задачу обратно в список «Предстоящие задачи».

- **Удалять задачи из любого списка с помощью кнопки «Удалить»** (№ 4 на рис. 14.1). Для полного удаления задачи со страницы вы предоставите кнопку «Удалить». Вы также можете использовать эффект jQuery UI для обеспечения зрелищности процесса удаления задачи.

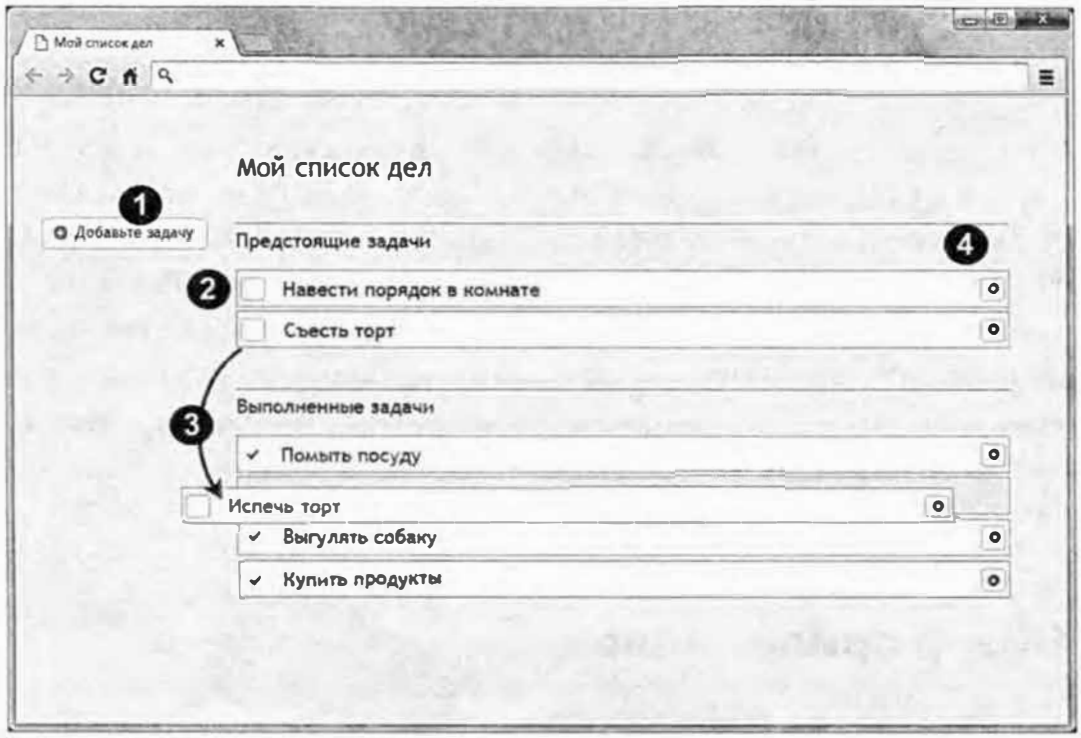

Рис. 14.1. Библиотека jQuery и плагин jQuery UI позволяют относительно легко создать простое приложение для построения списка дел. Эти инструменты решают такие сложные проблемы, связанные с пользовательским интерфейсом, как создание диалоговых окон, сортируемых списков и анимации элементов страницы, поэтому вы можете сконцентрироваться только на кодировании функциональности приложения

Поскольку плагин jQuery UI обеспечивает большую часть пользовательского интерфейса, вам нужно поработать только над базовой логикой приложения. Вы поэтапно решите эту задачу с помощью описанного выше плана. Начнете с добавления кнопки, потом добавите диалоговое окно, а затем создадите остальную часть приложения. В процессе программирования полезно бывает разбить задачу на более мелкие компоненты. Поэтому в работе с данным руководством, сделав одно дело, убедитесь, что все работает, и толькл затем переходите к следующему шагу.

Добавление кнопки

В первую очередь вы добавите кнопку и отформатируете ее с помощью плагина jQuery UI. В данном руководстве вы будете работать с двумя разными файлами, которые находятся в папке *глава14*: *index.html* и *todo.js*. Вы поместите весь код JavaScript в файл *todo.js* и добавите HTML-код для кнопки и диалогового окна в файл *index.html*.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

1. **В текстовом редакторе откройте файл *index.html* из папки *глава 14*.**

   Этот документ уже содержит ссылки на необходимые CSS-файлы jQuery и jQuery UI. Однако он еще не загружает файл *todo.js*, в который вы будете добавлять программный код.

2. **В строку под фрагментом `<script src="js/jquery-ui.min.js"></script>` добавьте следующий код:**

   ```
   <script src="todo.js"></script>
   ```

   Важно, чтобы файл *todo.js* являлся последним в списке, поскольку он зависит от jQuery и jQuery UI. Если вы загрузите его перед любым из этих файлов, то браузер предупредит о синтаксической ошибке при загрузке страницы.

   Далее вы добавите кнопку. Посетитель щелкнет по этой кнопке для добавления новой задачи в список дел.

3. **Найдите HTML-комментарий `<!-- добавьте кнопку сюда -->` и замените его следующим фрагментом HTML-кода:**

   ```
   <button id="add-todo">Добавьте задачу</button>
   ```

   В данном фрагменте кода нет ничего особенного, это просто обычный HTML-элемент `button`. Если вы сейчас просмотрите страницу, то увидите, что он выглядит как обычная невзрачная кнопка отправки формы. Далее вы превратите этот элемент в кнопку jQuery UI.

4. **Откройте в текстовом редакторе файл *todo.js* и внутри функции `$(document).ready()` введите:**

```
$('#add-todo').button();
```

Этот фрагмент кода просто применяет форматирование кнопки, предусмотренное темой jQuery UI (см. раздел «Знакомство с приложением ThemeRoller» главы 11). Вы можете ее немного украсить, добавив значок jQuery UI.

5. **Внутри функции `button()` добавьте литерал объекта JavaScript, который определяет значок для этой кнопки (дополнения выделены полужирным шрифтом):**

```
$('#add-todo').button({
icons: {
primary: "ui-icon-circle-plus"
}
});
```

Этот код добавляет на кнопку небольшой значок в виде символа «+» слева от текста «Добавьте задачу». Как уже говорилось в разделе «Стилизация кнопок» главы 10, плагин jQuery UI позволяет разместить на кнопке два значка: один слева (первичный значок `primary`) и один справа (вторичный значок `secondary`). Для данной кнопки достаточно одного значка.

6. **Сохраните оба файла *todo.js* и *index.html*. Просмотрите файл *index.html* в браузере.**

Кнопка должна соответствовать внешнему виду виджета jQuery UI Button (рис. 14.1). Если это не так, значит, JavaScript не работает. Перепроверьте свой код и воспользуйтесь консолью JavaScript вашего браузера, чтобы проверить код на наличие каких-либо ошибок.

Добавление диалогового окна

Теперь у вас есть кнопка, но она ничего не делает. Предполагается, что щелчок по ней должен приводить к открытию диалогового окна, поэтому следующая задача заключается в добавлении этого диалогового окна. Сначала вы добавите HTML-код:

1. **Найдите комментарий `<!-- добавьте диалоговое окно сюда -->` в файле *index.html* и замените его следующим HTML-кодом:**

```
<div id="new-todo" title="Добавьте задачу">
```

```
<form>
<p>
<label for="task">Название задачи:</label>
<input type="text" name="task" id="task">
</p>
</form>
</div>
```

Как вы узнали в разделе «Создание диалогового окна на практике» главы 9, вы можете превратить в диалоговое окно любой фрагмент HTML-кода. В данном случае существует элемент `div`, содержащий форму с единственным текстовым полем. Пользователи будут вводить название своей задачей в это поле при добавлении в список нового пункта.

Чтобы превратить этот HTML-код в диалоговое окно, вам нужно добавить код JavaScript.

2. **Вернитесь к файлу *todo.js*. Под кодом, добавленным в шаге 5 в предыдущем разделе, введите:**

```
$('#new-todo').dialog();
```

Сохраните файлы *index.html* и *todo.js*, а затем просмотрите файл *index.html* в браузере. Должно появиться диалоговое окно (см. верхнее изображение на рис. 14.2). Однако это окно появляется сразу, без нажатия кнопки. Это обычное поведение диалогового окна jQuery UI. Чтобы скрыть это диалоговое окно, вам необходимо передать несколько параметров функции `dialog()`.

3. **Добавьте объект с двумя свойствами в функцию `dialog()`:**

```
$('#new-todo').dialog({
modal : true,
autoOpen : false
});
```

Как вы помните из раздела «Настройка свойств диалогового окна» главы 9, параметр `modal` требует, чтобы пользователь закрыл диалоговое окно, чтобы получить возможность выполнять на веб-странице другие действия. Вам нужно, чтобы после щелчка по кнопке «До-

бавьте задание» основное внимание пользователя было сосредоточено на диалоговом окне.

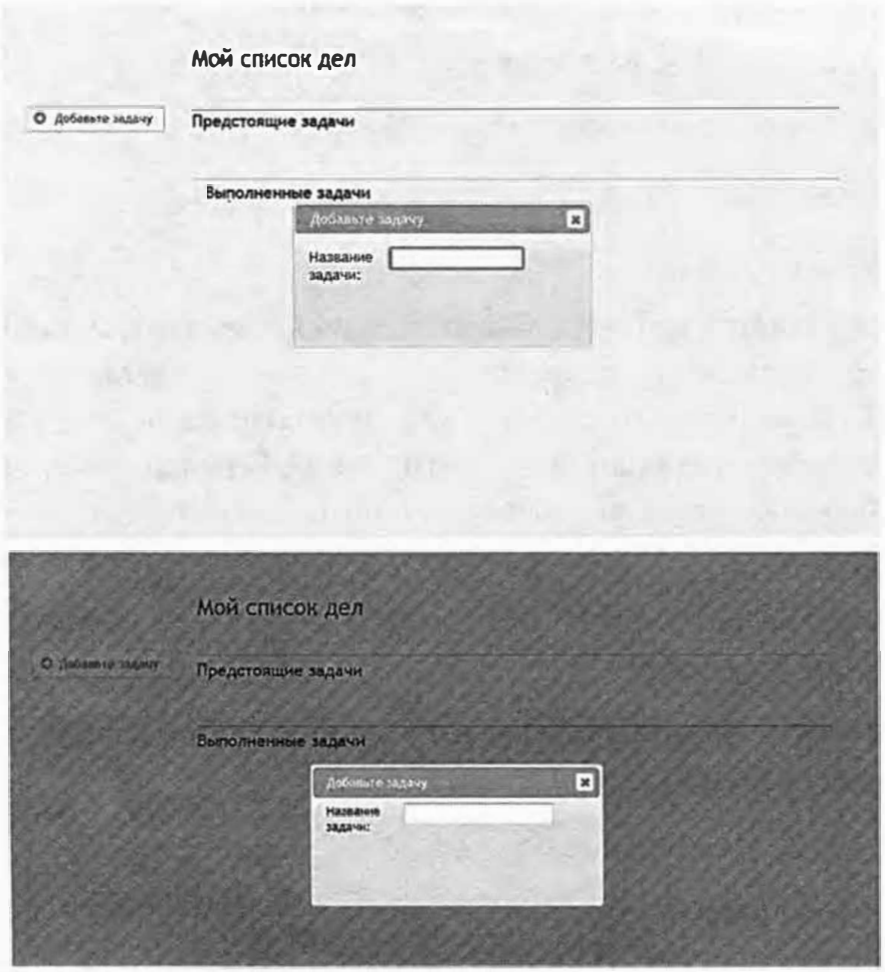

Рис. 14.2. Диалоговые окна jQuery UI автоматически открываются при загрузке страницы. Такое поведение отлично подходит для важных объявлений, которые должны увидеть посетители при попадании на страницу, но не очень полезно для большинства других целей, например, для добавления задачи в список дел. Чаще всего вам нужно будет, чтобы диалоговое окно оставалось скрытым, пока посетитель его не вызовет, например, пока он не щелкнет по кнопке «Добавьте задачу»

Если для параметра `autoOpen` установить значение `false`, то диалоговое окно не будет открываться при загрузке страницы. Теперь оно скрыто, и вам потребуется использовать программный код, чтобы оно открывалось после щелчка по кнопке!

4. **Добавьте к кнопке обработчик события `click`, связав функцию `click()` с окончанием функции `button()`:**

```
$('#add-todo').button({
icons: {
```

```
primary: "ui-icon-circle-plus"
}
}).click(function() {
$('#new-todo').dialog('open');
});
```

Вы можете «связать» jQuery-функции, просто добавив в конце одной функции точку, а после нее — еще одну функцию. В этом случае код сначала выберет элемент с идентификатором `add-todo` (кнопку), применит jQuery UI-функцию `button()`, а затем добавит обработчик события `click`. Этот обработчик события вызывает метод диалогового окна `open` (см. раздел «Открытие диалоговых окон с помощью событий» главы 9). Другими словами, теперь щелчок по кнопке должен приводить к открытию диалогового окна.

Теперь самое время проверить работу кода.

5. **Сохраните файл *todo.js* и просмотрите файл *index.html* в браузере. Щелкните по кнопке «Добавьте задачу».**

   Диалоговое окно должно появляться только после щелчка по кнопке (нижнее изображение на рис. 14.2). Кроме того, область под диалоговым окном должна затемняться и заполняться диагональными полосками. Данная визуальная подсказка указывает на то, что диалоговое окно является модальным. Это означает, что вы не можете ничего сделать, пока не закроете его. Но, как видите, у вас нет возможности его закрыть. На нем нет никаких кнопок!

   Плагин jQuery UI позволяет легко добавлять кнопки.

6. **Вернитесь к файлу *todo.js*. Добавьте в список параметров новый параметр `buttons` и задайте в качестве его значения пустой объект:**

```
$('#new-todo').dialog({
modal : true,
autoOpen : false,
buttons : {
}
});
```

Параметр `buttons` позволяет вам называть кнопки, которые плагин jQuery UI динамически добавляет в диалоговое окно. Кроме того, вы можете назначить каждой кнопке функции, чтобы при каждом щелчке по кнопке, что-нибудь происходило. Вы создаете этот код постепенно, шаг за шагом, поскольку в нем много чего происходит. Например, теперь у вас есть объект (свойство `buttons`) внутри другого объекта (объекта `options`, переданного функции `dialog()`).

Сначала просто добавьте кнопку с пустой функцией.

7. **Добавьте в объект `buttons` одно свойство (дополнения выделены полужирным шрифтом):**

```
$('#new-todo').dialog({
modal : true,
autoOpen : false,
buttons : {
"Добавить" : function () {
}
}
});
```

Этот код добавляет кнопку с меткой «Добавить». Когда пользователь щелкает по кнопке «Добавить», срабатывает функция. В настоящее время в этой функции ничего нет, поэтому она ничего не делает. Вы добавите функциональность кнопки в следующем разделе данного руководства. А сейчас вы добавите еще одну кнопку.

8. **Введите запятую после закрывающей фигурной скобки `}` функции `add task()` и добавьте еще одну кнопку с функцией:**

```
$('#new-todo').dialog({
modal : true,
autoOpen : false,
buttons : {
"Добавить" : function () {
},
"Отмена" : function () {
```

```
$(this).dialog('close');
}
}
});
```

Этот код добавляет вторую кнопку с меткой «Отмена». На этот раз кнопка выполняет некоторое действие. Фрагмент `$(this)` относится к элементу, который вызывает функцию, то есть к самому диалоговому окну.

При щелчке по кнопке «Отмена» вызывается метод `close` виджета Dialog плагина jQuery UI (см. раздел «Добавление кнопок в диалоговое окно» главы 9). Такой метод просто заставляет диалоговое окно закрыться и больше ничего не делать.

Пришло время проверить, как это работает.

9. **Сохраните документ *todo.js* и просмотрите файл *index.html* в браузере. Щелкните по кнопке «Добавьте задачу».**

Должно появиться диалоговое окно с двумя кнопками (рис. 14.3). Если вы щелкнете по кнопке «Добавить», то ничего не произойдет, поскольку вы еще не запрограммировали никаких действий. Тем не менее вы добавили код для кнопки «Отмена», поэтому при щелчке по ней диалоговое окно закрывается.

В следующем разделе вы добавите код, позволяющий добавить на страницу элементы списка дел.

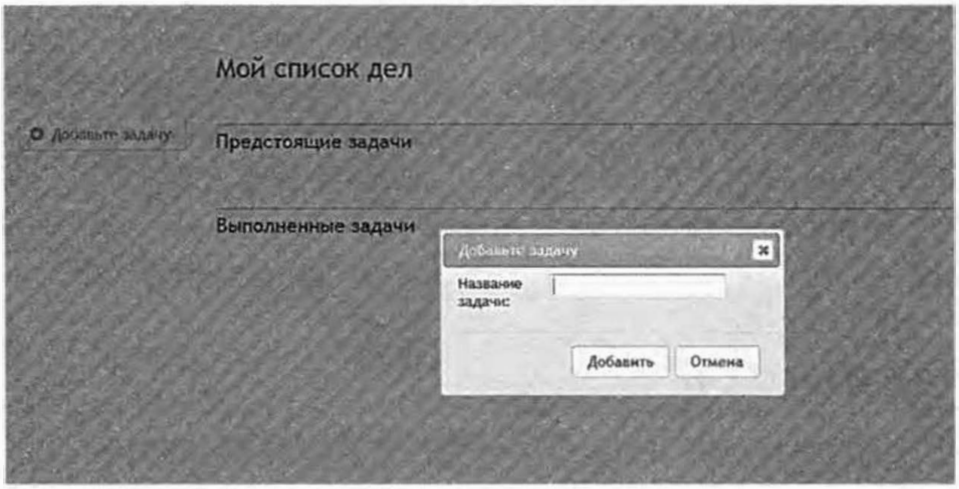

Рис. 14.3. С помощью всего лишь нескольких строк кода вы создали все основные компоненты пользовательского интерфейса, позволяющие добавлять элементы в список дел

Добавление задач

Приложение для создания списка дел выглядит хорошо, но делает немного. Пора написать код для добавления задачи в список «Предстоящие задачи». Весь этот код прикреплен к пустой функции, назначенной кнопке диалогового окна «Добавьте задачу», которую вы добавили на шаге 7 в предыдущем разделе. Процесс создания кода разделен на 4 этапа:

1. **Извлеките название задачи, введенное пользователем в диалоговое окно.**

   Это значение сохраняется в текстовом поле формы, поэтому вы можете легко использовать jQuery-функцию `val()` (см. раздел «Получение и установка значений элементов форм» главы 8), чтобы извлечь название задачи.

2. **Создайте элемент `li`, который нужно добавить на страницу.**

   Каждая задача будет представлена элементом неупорядоченного списка. Базовый HTML-код для каждого пункта списка выглядит следующим образом:

```
<li>
<span class="done">%</span>
<span class="delete">x</span>
<span class="task">Испечь торт</span>
</li>
```

   Первый элемент `span` соответствует области, по которой посетитель щелкает, чтобы отметить задачу как выполненную. Второй элемент `span` представляет собой кнопку удаления задачи. Последний элемент `span` содержит название задачи, которое посетитель вводит в диалоговом окне. Вы можете создать этот фрагмент HTML-кода путем объединения строк, содержащих только что перечисленные элементы, с названием задачи, предоставленным посетителем.

ПРИМЕЧАНИЕ

Вы станете использовать код CSS для форматирования кнопок «Выполнено» и «Удалить» каждого элемента списка. В самом деле, символы `%` и `x` в первых двух вышеприведенных элементах `span` будут иметь вид флажка и кнопки удаления благодаря веб-шрифту, в котором вместо букв используются значки.

3. **Добавьте новый элемент `li` в список «Предстоящие задачи».**

   Библиотека jQuery позволяет легко преобразовать строку в настоящий элемент DOM и добавить его на веб-страницу (в главе 4 это подробно описано). Поскольку в вашем распоряжении есть мощные эффекты jQuery UI, вы используете их для обеспечения зрелищности процесса добавления новой задачи.

4. **Закройте диалоговое окно.**

   Это легко. Вы уже программировали такую функциональность для кнопки «Отмена» диалогового окна в шаге 8 в предыдущем разделе.

Теперь, когда вы знаете основные этапы создания кода, пора приступить к работе. Во-первых, вы захватите данные, введенные посетителем в диалоговом окне:

1. **Найдите функцию, назначенную кнопке «Добавить» (внутри параметра `buttons` функции `dialog()`) и добавьте код, выделенный полужирным шрифтом:**

```
"Добавить" : function () {
var taskName = $('#task').val();
},
```

   Этот код создает новую переменную `taskName` и присваивает значение текстовому полю диалогового окна. Помните, в шаге 1 в разделе «Добавление диалогового окна» вы добавили HTML-код для диалогового окна. Он включал в себя поле ввода с идентификатором `task`, поэтому фрагмент `$('#task').val()` возвращает значение этого поля, иными словами, то, что посетитель ввел в качестве названия задачи.

   Существует вероятность того, что посетитель оставит это поле пустым и щелкнет по кнопке «Добавить». Поскольку вы не хотите добавлять пустую задачу в список дел, вам следует удостовериться в том, что переменная `TaskName` не является пустой.

2. **Добавьте еще три строки кода под той, которую вы добавили в предыдущем шаге (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
```

```
return false;
}
},
```

Этот код позволяет убедиться в том, что переменная `TaskName` не содержит пустой строки (то есть две одинарные кавычки рядом друг с другом: `''`), а также в том, что посетитель не просто щелкнул по кнопке «Добавить», ничего не введя в текстовое поле. Фрагмент `return false` просто обеспечивает выход из функции, в результате чего диалоговое окно остается открытым. Посетитель должен либо ввести название задачи, либо щелкнуть по кнопке «Отмена», чтобы закрыть диалоговое окно (или щелкнуть по маленькой кнопке X, которая находится в правом верхнем углу всех диалоговых окон jQuery UI).

Далее вы приступите к построению HTML-кода для этой задачи.

3. **Добавить новую переменную и добавить значения строк в нем, добавив еще три линии кода (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
},
```

Вы могли бы использовать всего одну строку кода, чтобы создать переменную `taskHTML` и сохранить в ней одну действительно длинную строку. Однако это привело бы к созданию очень длинной и сложной для восприятия инструкции JavaScript. Построение длинной инструкции на нескольких строках позволяет облегчить чтение кода. Операция += позволяет объединить строку со строкой, содержащейся в уже существующей переменной (см. раздел «Изменение значений в переменных» главы 2).

Обратите внимание на то, что вы не добавили в эту строку переменную `taskName`, то есть задачу, созданную посетителем.

4. **Добавьте еще одну переменную и присвойте ей объект jQuery, созданный из строки HTML-кода (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
},
```

Библиотека jQuery позволяет преобразовать текстовую строку, например, `'<h1>A headline</h1>'`, в настоящий элемент DOM. Другими словами, `taskHTML` является переменной, которая просто содержит строку, то есть серию символов. Тем не менее эта строка не является «настоящим» HTML-кодом. Передача строки, отформатированной как HTML-код, функции `$()` превращает ее в элемент DOM. Еще лучше то, что при этом строка превращается в объект jQuery, так что вы можете применить к нему все обычные jQuery-функции. Хотя символ `$` в начале имени переменной `$newTask` не является обязательным, его использование считается обычной практикой среди JavaScript-программистов, которые используют библиотеку jQuery. Символ `$` наглядно показывает программисту, что переменная содержит объект jQuery и что к ней применимы различные методы jQuery, например, `.show()` и `.addClass()`.

Далее вы добавите задачу, создаваемую посетителем.

5. **Добавьте еще одну строку кода в функцию `"Добавить"` (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
```

```
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
$newTask.find('.task').text(taskName);
},
```

Метод `find()` библиотеки jQuery ищет элемент внутри текущего элемента (см. раздел «Обход дерева DOM» главы 15). В данном случае библиотека jQuery берет элемент `li`, содержащийся внутри элемента `$newTask`, и ищет внутри него другой элемент с именем класса `task` — это элемент `span`, куда должно помещаться название задачи (см. шаг 3). JQuery-функция `text()` затем добавляет содержимое переменной `taskName` в этот элемент `span`.

Зачем все эти сложности, когда можно просто добавить переменную `taskName` при создании строки в шаге 4:

```
taskHTML += '<span class="task">' + taskName + ↵
'</span></li>';
```

Если вы так сделаете, то какой-нибудь нечестный посетитель может ввести в качестве названия задачи что-то вроде: `<script>alert('ха, ха, ха, я нарушаю работу вашего приложения');</script>`. Этот код будет добавлен на страницу в качестве простого фрагмента HTML-кода, и JavaScript в этом примере будет выполнен. Тем не менее, jQuery-функция `text()` преобразует все HTML-элементы в безопасные эквиваленты, то есть вместо `<script>` получается `<script&gt`.

Даже если у посетителя нет злых намерений, этот шаг позволяет ему ввести такую задачу, как «Добавить элемент `h1` на главную страницу», не нарушая работы приложения.

Теперь пришло время добавить на страницу новый элемент списка дел.

6. **Добавьте еще две строки кода внутри функции "Добавить" (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
```

```
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
$newTask.find('.task').text(taskName);
$newTask.hide();
$('#todo-list').prepend($newTask);
},
```

Поскольку элемент `$newTask` является объектом jQuery, вы можете применить к нему функции этой библиотеки. Сначала вы скрываете его, чтобы затем отобразить, используя какой-нибудь зрелищный анимационный эффект. Когда он окажется скрытым, выберите список «Предстоящие задачи» — неупорядоченный список на странице с идентификатором `todo-list`, а затем добавьте новый (скрытый) элемент в начало списка (см. раздел «Добавление содержимого на страницу» главы 4, чтобы узнать, как работает метод `prepend()`).

Теперь вам нужно отобразить новый элемент списка и закрыть диалоговое окно.

7. **Добавьте еще две строки в код, отвечающий за функциональность кнопки «Добавьте задачу» (дополнения выделены полужирным шрифтом):**

```
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
$newTask.find('.task').text(taskName); $newTask.hide();
```

```
$('#todo-list').prepend($newTask);
$newTask.show('clip',250).effect('highlight',1000);
$(this).dialog('close');
},
```

Первая новая строка выбирает только что добавленный и скрытый элемент списка, а затем использует метод jQuery `show()`, чтобы отобразить его. Для обеспечения зрелищности вы используете два эффекта jQuery UI: эффект `clip` (см. раздел «Эффекты» главы 12) обеспечивает визуальное увеличение нового элемента. После отображения элемента метод `effect()` привлекает к нему внимание с помощью вспышки желтого цвета, что вызывает восторг пользователей приложения.

Последняя строка закрывает диалоговое окно (вы уже применяли этот код к кнопке «Отмена» на шаге 8 в разделе «Добавление диалогового окна» ранее в главе). Пришло время проверить, как работает код.

8. **Сохраните файл *todo.js* и откройте файл *index.html* в браузере. Щелкните по кнопке «Добавьте задачу», введите название задачи в появившемся диалоговом окне, а затем щелкните по кнопке «Добавить».**

   Новая задача появится под заголовком «Предстоящие задачи». Если это не так, проверьте свой код и консоль JavaScript вашего браузера на предмет наличия ошибок.

   Попробуйте добавить еще одну задачу. У вас должна возникнуть небольшая проблема — поле для ввода текста диалогового окна запоминает последнюю задачу, которую вы ввели. Чтобы добавить новое задание, сначала нужно удалить из текстового поля старое. Исправить эту проблему легко.

9. **После параметра `buttons` внутри объекта, переданного функции `dialog()`, введите запятую и добавьте следующий фрагмент кода:**

   ```
   close: function() {
   $('#new-todo input').val('');
   }
   ```

   Полный код для функции `dialog()` должен выглядеть так:

   ```
   $('#new-todo').dialog({
   ```

```
modal : true,
autoOpen : false,
buttons : {
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
$newTask.find('.task').text(taskName); $newTask.hide();
$('#todo-list').prepend($newTask);
$newTask.show('clip',250).effect('highlight',1000);
$(this).dialog('close');
},
"Отмена" : function () {
$(this).dialog('close');
}
},
close: function() {
$('#new-todo input').val('');
}
});
```

Как много кода. Убедитесь в том, что вы добавили этот новый код за пределами объекта `buttons`. Параметр `close` виджета Dialog jQuery UI позволяет вам выполнить функцию, когда посетитель закрывает диалоговое окно. В этом случае, вы очищаете текстовое поле, так что при следующем открытии диалогового окна поле будет пустым и готовым к приему названия новой задачи. (Вы также можете очистить поле ввода в рамках функции «Добавить», однако этот пример предназначен для демонстрации параметра `close`.)

Маркировка задачи как выполненной

Одно из самых приятных действий при работе со списком дел заключается в установке флажка рядом с названием выполненной задачи. К сожалению, у вашего списка дел отсутствует такая полезная функция. В этом разделе вы исправите это упущение. Этот процесс довольно прост: пользователь устанавливает флажок слева от названия задачи, и эта задача (элемент списка) перемещается из одного неупорядоченного списка в другой.

Делегирование событий

Чтобы отметить задачу как выполненную, пользователь должен установить флажок рядом с элементом списка. Обычно вы используете jQuery для выбора элемента и добавляете обработчик события следующим образом:

```
$('.done').click(function () {
// выполнить какое-либо действие при щелчке по данному элементу
});
```

Тем не менее существует разница в плане действий, совершаемых над элементами этого списка дел. При первой загрузке веб-страницы нет никаких элементов списка и никаких флажков, позволяющих пометить задачу, как выполненную. Если вы примените обработчик события после загрузки страницы, ничего не произойдет. Поскольку после загрузки страницы на ней отсутствуют какие-либо элементы списка и флажки, то нет ничего, к чему можно было бы применить обработчик. Только после добавления задачи на странице появляется элемент, по которому можно щелкнуть. Вы можете добавлять метод `click()` при создании каждого следующего элемента списка, однако библиотека jQuery предусматривает лучший метод, называемый «делегированием событий» (см. раздел «Делегирование событий с помощью функции `on()`» главы 5).

В двух словах, делегирование событий позволяет выбрать другой уже существующий элемент страницы, который будет содержать элементы, добавляемые динамически после загрузки страницы. Этот элемент-контейнер станет отслеживать определенное событие, например, событие `click`, происходящее внутри этого элемента. Когда он получит это событие, он удостоверится в том, что событие на самом деле сработало

на конкретном элементе (например, на флажке рядом с названием задачи), и если это так, то сработает обработчик события.

В данном случае пустой неупорядоченный список уже находится на веб-странице при ее загрузке:

```
<ul id="todo-list">
</ul>
```

По мере того как посетитель добавляет задачи в список дел, ваша программа динамически добавляет элементы списка:

```
<ul id="todo-list">
<li>
<span class="done">%</span>
<span class="delete">x</span>
<span class="task">Испечь торт</span>
</li>
</ul>
```

Чтобы отреагировать на событие `click`, сработавшее на элементе `done`, вы должны делегировать обработчик события существующему неупорядоченному списку:

1. **После кода для функции `dialog()`, но внутри функции `$(document).ready()` введите следующий код:**

   ```
   $('#todo-list').on('click', '.done', function() {
   });
   ```

   Это основная схема делегирования событий. Сначала вы выбираете неупорядоченный список. Затем используете метод jQuery `on()`, который принимает три аргумента. Первым из них является строка, содержащая имя события — `'click'`. Вторым аргументом является элемент внутри неупорядоченного списка, по которому нужно щелкнуть. В данном случае это элемент `span` с классом `done`. Наконец вы добавляете функцию — программный код, который должен выполняться, когда посетитель устанавливает флажок.

2. **Внутри функции обработчика события добавьте следующий выделенный полужирным шрифтом код:**

   ```
   $('#todo-list').on('click', '.done', function() {
   ```

```
var $taskItem = $(this).parent('li');
});
```

Помните о том, что смысл данной функции заключается в перемещении элемента из списка «Предстоящие задачи» в список «Выполненные задачи». Когда посетитель устанавливает флажок, библиотека jQuery «знает» только об одном этом элементе `span` в элементе списка, однако нам необходимо получить доступ ко всему элементу `li`. Для этого вы будете использовать метод jQuery `parent()`, позволяющий выбрать родительский элемент текущей выборки.

Здесь ключевое слово `$(this)` относится к элементу, по которому щелкает посетитель, — к элементу `<span class="done">%</span>`. Таким образом, фрагмент `$(this).parent('li')` выбирает ближайшего предка, который является элементом `li`. Иными словами, он выбирает нужный вам элемент списка!

Этот элемент сохраняется в переменной `$taskItem`, которую вы сейчас заставите исчезнуть.

ПРИМЕЧАНИЕ

Более подробное описание ключевых слов this и $(this) вы можете найти в разделе «Ключевые слова `this` и `$(this)`» главы 4.

3. **Внутри функции обработчика событий добавьте код, выделенный полужирным шрифтом:**

```
$('#todo-list').on('click', '.done', function() {
var $taskItem = $(this).parent('li'); $taskItem. ↵
slideUp(250, function() {
});
});
```

Метод jQuery `slideUp` позволяет скрыть элемент страницы интересным способом (см. раздел «Скользящие элементы» главы 6). Однако он на самом деле не удаляет элемент со страницы. После выполнения функции `slideUp` элемент по-прежнему находится на странице, он просто скрыт с помощью кода CSS. Как вы помните из раздела «Эффекты jQuery» главы 6, все анимационные jQuery-функции (например, `hide()`, `show()` и `slideUp()`) могут принимать аргументы. В данном случае первый аргумент — `250` — соответствует длительности анимации, то есть 250 миллисекундам.

Вторым аргументом здесь является функция обратного вызова. Это функция, которая срабатывает после того, как завершится анимация jQuery. Помните о том, что после выполнения метода `slideUp` элемент все еще находится в списке «Предстоящие задачи», он просто скрыт. Вы должны переместить его в другой неупорядоченный список. Вы можете сделать это в рамках функции обратного вызова.

4. **Внутри функции обратного вызова добавьте код, выделенный полужирным шрифтом:**

```
$('#todo-list').on('click', '.done', function() {
var $taskItem = $(this).parent('li'); $taskItem. ↵
slideUp(250, function() {
var $this = $(this);
$this.detach();
});
});
```

Первая новая строка просто выбирает элемент списка — `$(this)` — и сохраняет его в другой переменной. Это делается потому, что каждый запуск jQuery-функции — `$()` — заставляет браузер выполнять работу. Вы должны совершить несколько действий с этим элементом списка, поэтому вместо многократного вызова jQuery-функции вы можете запустить ее только один раз и сохранить результаты ее работы в переменной. (Эта практика более подробно описана в разделе «Сохранение выборок в переменных» в главе 15).

Во второй строке используется метод jQuery `detach()`, чтобы удалить со страницы выбранный HTML-элемент или элементы, но не полностью избавиться от них. Другими словами, элемент списка отсутствует в списке, но все еще существует в памяти. На самом деле он все еще хранится в переменной `$this`. Теперь вы можете переместить отделенный элемент в другое место на странице — в другой список!

ПРИМЕЧАНИЕ

Посетите страницу **api.jquery.com/detach/**, чтобы подробнее узнать о методе jQuery `detach()`.

5. **Закончите функцию обратного вызова, добавив еще две строки (выделены полужирным шрифтом):**

```
$('#todo-list').on('click', '.done', function() {
var $taskItem = $(this).parent('li');
$taskItem.slideUp(250, function() {
var $this = $(this);
$this.detach();
$('#completed-list').prepend($this);
$this.slideDown();
});
});
```

Вы уже видели метод `prepend()` раньше (в шаге 6 в разделе «Добавление задач» ранее в главе). Этот метод позволяет вставить HTML-код в другой элемент. В данном случае отделенный элемент — `$(this)` — добавляется в список «Выполненные задачи» (другой неупорядоченный список с идентификатором `completed-list`). Наконец, поскольку элемент был скрыт с помощью метода `slideUp()`, вы теперь можете отобразить его в новом месте страницы с помощью метода `slideDown()` (см. раздел «Скользящие элементы» главы 6).

6. **Сохраните файл *todo.js* и просмотрите файл *index.html* в браузере. Чтобы проверить работоспособность кода, добавьте несколько новых элементов списка дел, а затем установите флажок слева от названия задачи, чтобы отметить ее как выполненную.**

   Вы должны иметь возможность добавлять задачи и отмечать их как выполненные (рис. 14.4). Если вы не можете этого сделать, проверьте свой код и консоль JavaScript вашего браузера на предмет наличия ошибок.

   Теперь неплохо было бы обеспечить возможность изменения порядка следования задач в списке. Например, после добавления в список множества задач вы можете расположить их в том порядке, в котором вы собираетесь их выполнять, например: «Купить поваренную книгу», «Испечь торт» и «Съесть торт». Виджет jQuery UI Sortable позволяет вам легко добавить такую функцию. Таким образом, вы можете свободно перемещать элементы между списками и отмечать задачу как «завершенную», просто перетащив ее в список «Выполненные задачи».

7. **Откройте файл *index.html* в текстовом редакторе и найдите элементы `ul` для списков «Предстоящие задачи» и «Выполненные за-**

дачи». Добавьте в каждый из них фрагмент `class="sortlist"`, чтобы код выглядел следующим образом:

```
<ul id="todo-list" class="sortlist">
```

и

```
<ul id="completed-list" class="sortlist">
```

Назначив обоим спискам одинаковое имя класса, вы можете легко выбрать оба списка и сделать их сортируемыми.

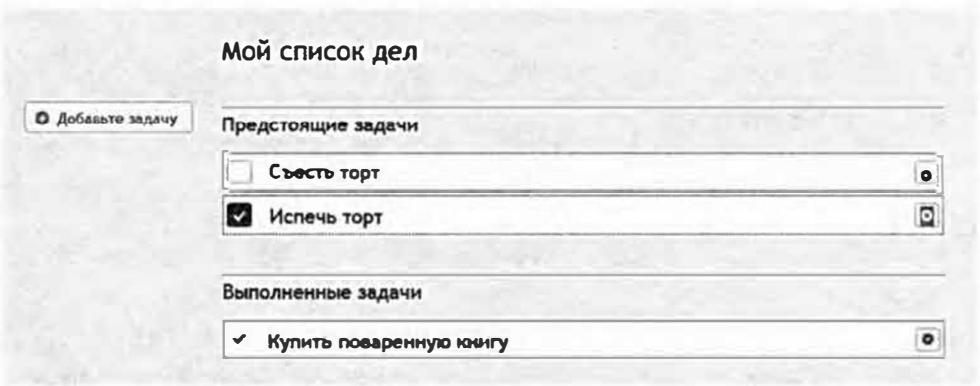

Рис. 14.4. Кнопки, относящиеся к элементам списка, которые позволяют отметить задачу, как выполненную, или удалить ее, изменяют вид при наведении на них указателя мыши. Это обеспечивается не JavaScript-кодом, а кодом CSS, в частности, псевдоклассом CSS `:hover`, который позволяет изменять внешний вид элемента при наведении на него указателя мыши. Чтобы увидеть, как работает этот эффект, обратитесь к файлу *todo.css* в папке *глава14/css*

8. **Сохраните файл *index.html* и откройте файл *todo.js* в текстовом редакторе. После кода для обработчика делегированных событий, который вы только что добавили, но в рамках функции `$(document).ready()`, введите следующий фрагмент кода:**

```
$('.sortlist').sortable();
```

Теперь два списка являются сортируемыми. Вы можете сохранить файлы и просмотреть файл *index.html*, если хотите проверить его работоспособность. Вы заметите, что, несмотря на возможность переупорядочивания задач, вы не можете перетащить элемент из одного списка в другой. Сначала вы должны соединить два виджета Sortable.

9. **Внутри функции `sortable()`, которую вы только что добавили, вставьте объект со свойством и значением:**

```
$('.sortlist').sortable({
connectWith : '.sortlist'
});
```

Параметр `connectWith` виджета Sortable (см. раздел «Параметры виджета Sortable» главы 12) позволяет подключить один список к другому. Назначение обоим спискам одного и того же класса обеспечивает соединение этих списков. Теперь вы можете свободно перетаскивать элементы между списками, однако перед тем как закончить работу с виджетами Sortable, вам предстоит добавить несколько последних элементов пользовательского интерфейса.

10. **Добавьте еще три параметра в объект виджета Sortable `option`:**

```
$('.sortlist').sortable({
connectWith : '.sortlist',
cursor : 'pointer',
placeholder : 'ui-state-highlight',
cancel : '.delete,.done'
});
```

Не забудьте запятую после параметра `connectWith`. Параметр `cursor` (см. раздел «Параметры виджета Draggable» главы 12) изменяет внешний вид указателя мыши при перетаскивании элемента списка. Параметр `placeholder` (см. раздел «Параметры виджета Sortable» главы 12) выделяет позицию в списке, на которую посетитель может поместить элемент при перетаскивании. И, наконец, параметр `cancel` (см. раздел «Параметры виджета Sortable» главы 12) определяет элементы сортируемого виджета, за которые его нельзя будет перетаскивать: в данном случае посетитель не можете перетащить задачу за значок «Удалить задачу» или за флажок «Отметить как выполненную».

11. **Сохраните файлы *index.html* и *todo.js* и просмотрите файл *index.html* в веб-браузере.**

Добавьте несколько задач. Перетащите задачи между двумя списками. Вы должны иметь возможность отметить задачу как выполненную, просто перетащив ее списка «Предстоящие задачи» в список «Выполненные задачи» (см. рис. 14.1).

Удаление задач

Осталось добавить возможность удаления задачи. Эта функция важна потому, что пользователь может случайно добавить ненужную задачу. Кроме того, когда список «Завершенные задачи» становится слишком длинным, пользователь может решить удалить некоторые элементы этого списка.

1. **После кода для функции `sortable()`, но внутри функции `$(document).ready()`, введите следующий фрагмент:**

```
$('.sortlist').on('click', '.delete', function() {
});
```

Здесь снова применяется делегирование событий. При загрузке страницы на ней нет никаких элементов списка и кнопок для удаления, поэтому вам следует использовать метод делегирования событий, чтобы исправить это. В данном случае фрагмент `$('.sortlist')` выбирает оба неупорядоченных списка на странице (поскольку пользователи должны иметь возможность удалять задачи из любого списка), а метод `on()` заставляет библиотеку jQuery отслеживать события `click` на любом элементе с классом `delete`. При щелчке по этому элементу вызывается функция обработчика данного события.

Далее вы запрограммируете обработчик событий.

2. **Внутри функции обработчика событий добавьте три строки кода, выделенные полужирным шрифтом:**

```
$('.sortlist').on('click','.delete',function() {
$(this).parent('li').effect('puff', function() {
$(this).remove();
});
});
```

В этом коде происходит очень много всего, но вы уже должны к этому привыкнуть. Далее приведен его анализ:

- Фрагмент `$(this).parent('li')` выбирает элемент, по которому щелкнул пользователь — `$(this)`, а затем находит предка, являющегося элементом `li`. Другими словами, он выбирает элемент списка, который вы хотите удалить.
- Метод jQuery UI `effect()` применяет к этому элементу один из многочисленных эффектов. В данном случае эффект `puff` заставляет элемент увеличиваться в размерах, выцветать и исчезать.
- Функция внутри метода `effect()` является функцией обратного вызова и запускается после срабатывания эффекта. В данном случае он выбирает пункт, к которому был применен метод `effect()` (это элемент списка, выбранный с помощью ключевого слова `$(this)`) и удаляет его со страницы с помощью метода jQuery `remove()` (см. раздел «Замена и удаление выборок» главы 4). В отличие от описанного выше метода `detach()`, метод `remove()` полностью удаляет выбранные элементы со страницы.

3. **Сохраните файлы и просмотрите документ *index.html* в веб-браузере.**

   Вы должны быть в состоянии добавлять, перемещать и удалять элементы списка дел. Страница должна соответствовать рис. 14.1. На случай возникновения каких-либо проблем далее приведен полный код данного руководства:

```
$(document).ready(function(e) {
$('#add-todo').button({
icons: {
primary: "ui-icon-circle-plus"
}
}).click(function() {
$('#new-todo').dialog('open');
});
$('#new-todo').dialog({
modal : true,
autoOpen : false,
buttons : {
"Добавить" : function () {
var taskName = $('#task').val();
if (taskName === '') {
return false;
}
var taskHTML = '<li><span class="done">%</span>';
taskHTML += '<span class="delete">x</span>';
taskHTML += '<span class="task"></span></li>';
var $newTask = $(taskHTML);
$newTask.find('.task').text(taskName);
$newTask.hide();
$('#todo-list').prepend($newTask);
$newTask.show('clip',250).effect('highlight',1000);
$(this).dialog('close');
},
"Отмена" : function () {
$(this).dialog('close');
}
},
```

```
close: function() {
$('#new-todo input').val('');
}
});
$('#todo-list').on('click','.done', function() {
var $taskItem = $(this).parent('li'); $taskItem. ↵
slideUp(250, function() {
var $this = $(this);
$this.detach();
$('#completed-list').prepend($this); $this. ↵
slideDown();
});
});
$('.sortlist').sortable({
connectWith : '.sortlist',
cursor : 'pointer',
placeholder : 'ui-state-highlight',
cancel : '.delete,.done'
});
$('.sortlist').on('click','.delete',function() {
$(this).parent('li').effect('puff', function() {
$(this).remove();
});
});
}); // конец ready
```

ПРИМЕЧАНИЕ

Финальную версию данного примера вы найдете в файлах *complete-index.html* и *complete-todo.js* в папке *глава14*.

Усовершенствование приложения

Поздравляем, вы создали свое первое веб-приложение! Однако существуют способы его улучшения. Вы, вероятно, определили дополнительные функции, какие хотели бы добавить. В этом разделе описаны некоторые возможные улучшения, которые вы могли бы внести в приложение, а также указаны ресурсы, где подробно объясняется процесс внедрения описанных изменений.

Редактирование задач

На данном этапе приложение не позволяет пользователю исправить опечатку в названии задачи. Если он создаст новую задачу под названием «Испечь торт», то ему придется удалить ее и создать новую задачу с правильным названием. Существуют два способа решения этой проблемы. Первый способ заключается в добавлении кнопки «Редактировать» для каждого элемента списка. Щелчок по этой кнопке будет приводить к открытию диалогового окна для редактирования названия задачи, которое содержится внутри элемента `span` с классом `task`.

Вы можете добавить на страницу другое диалоговое окно вроде того, что вы создали в разделе «Добавление диалогового окна» ранее в данной главе. Щелчок по кнопке «Редактировать» приводит к открытию диалогового окна и помещает название задачи в редактируемое текстовое поле. Закрытие окна обновляет название задачи в списке.

Другой метод подразумевает использование HTML-свойства `contentEditable`. Это свойство делает любой HTML-элемент доступным для редактирования. Например, HTML-код, обеспечивающий возможность редактирования названия задачи в приведенном примере, будет выглядеть следующим образом:

```
<span class="task" contentEditable>
```

Это свойство можно также применять динамически с помощью jQuery: `$('.task').prop('contentEditable', true);`

Существует одна проблема, связанная с использованием свойства `contentEditable` в данном приложении. Виджет jQuery UI Sortable не допускает возможности выбора текста внутри сортируемых элементов, поэтому вы не можете выбрать текст, чтобы изменить его, даже если свойство `contentEditable` включено. Вы можете обойти эту проблему, приказав плагину jQuery UI проигнорировать элементы из списка задач при применении функции `sortable()`. Вы уже делали нечто подобное для кнопки удаления задачи и флажка в шаге 10 в разделе «Делегирование событий» ранее в главе. Вам просто нужно добавить в список селектор `.task:cancel : '.delete,.done,.task'`

Если вы пойдете этим путем, то от области элемента, по которой пользователь может щелкнуть и перетащить, мало что останется. В этом случае вы должны четко определить маркер, с помощью которого посетитель может перетаскивать задачи в списке. Для этого подойдет параметр `handle` виджета Sortable (см. раздел «Параметры виджета Sortable» главы 12).

Подтверждение удаления

В настоящий момент щелчок по кнопке «Удалить» безвозвратно удаляет задачу. Вы можете добавить модальное диалоговое окно, которое просит посетителя подтвердить действие. Если посетитель щелкает по кнопке «Да», задача удаляется, если по кнопке «Нет», задача остается в списке.

Сохранение списков

Самая большая проблема с данным приложением заключается в том, что оно не сохраняет список после закрытия окна браузера. Список дел является временным и не сохраняется между разными сессиями или при просмотре страницы на разных компьютерах. Существует несколько способов для сохранения состояния вашего списка дел.

Локальное хранилище данных

Все современные браузеры предусматривают функцию под названием *локальное хранилище*, которая позволяет сохранять данные на компьютере посетителя и извлекать эти данные, когда посетитель возвращается на страницу. Вы можете использовать локальное хранилище, чтобы делать снимок элементов при каждом обновлении списка. Когда посетитель возвращается на страницу, вы проверяете наличие данных в локальном хранилище и, если они есть, добавляете их на страницу.

Подробнее о локальном хранилище вы можете узнать на странице: **developer.mozilla.org/ru/docs/Web/Guide/API/DOM/Storage**. Существует также плагин jQuery, облегчающий процесс использования локального хранилища: **github.com/julien-maurel/jQuery-Storage-API**.

Сохранение на сервере

Другой подход заключается в сохранении списка дел на веб-сервере. Преимущество этого подхода состоит в том, что вы можете получить доступ к списку дел с любого компьютера. Недостатком является то, что вам необходимо создать некую систему для управления доступом к списку дел. В противном случае любой может посетить страницу и увидеть ваши списки задач (и удалить их).

В этой книге не освещается тема серверного программирования. Тем не менее вам придется использовать код JavaScript для отправки элементов списка дел на сервер. Вы будете использовать технологию Ajax, описанную в предыдущей главе, чтобы отправить данные на сервер.

Лучше всего захватить все элементы списка с помощью jQuery и использовать метод `.each()`, чтобы обойти все элементы списка и извлечь только название задачи. Вы можете создать объект JavaScript со списком, содержащим как предстоящие, так и выполненные задачи. Наконец, чтобы иметь возможность отправить объект, содержащий список задач, на сервер с помощью технологии Ajax, вам необходимо сериализовать этот объект, то есть преобразовать его в обычную строку, которая может быть отправлена на сервер. Это можно сделать с помощью следующей функции:

```
function getData() {
var todoData = {
toDo : [],
completed : []
};
$('#todo-list').each( function() {
var task = $(this).find('.task').text();
todoData.toDo.push(task);
});
$('#completed-list').each( function() {
var task = $(this).find('.task').text();
todoData.completed.push(task);
});
return $.param(todoData);
}
```

Вы можете вызвать эту функцию, когда вам понадобится извлечь элементы списка дел в том виде, в котором они могут быть отправлены на сервер. Программа на вашем веб-сервере декодирует полученные данные и произведет над ними соответствующие действия.

Дополнительные идеи

Если у вас есть дополнительные идеи для улучшения этого приложения, постарайтесь развить их. Это отличный проект для отработки навыков использования JavaScript, jQuery и jQuery UI. Вы можете найти постоянно дорабатываемую версию этого проекта по адресу: **github.com/sawmac/jquery-todo**.

Часть V

ДИАГНОСТИКА, СОВЕТЫ И НЮАНСЫ

Глава 15. Дополнительные возможности библиотеки jQuery

Глава 16. Совершенствуемся в программировании на языке JavaScript

Глава 17. Диагностика и отладка

Глава 15

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ БИБЛИОТЕКИ JQUERY

Библиотека jQuery заметно упрощает процесс программирования на языке JavaScript и позволяет быстро добавлять на сайт интерактивные элементы любого уровня сложности. В этой книге вы познакомились с примерами использования плагинов библиотеки jQuery для проверки формы и создания эффектов перекатывания (ролловера). Однако даже в этой библиотеке встречаются трудные места, которые потребуют от вас дополнительных знаний — если, конечно, вы твердо решили полностью освоить выбранный инструмент. В этой главе мы продолжим изучение возможностей библиотеки jQuery. Вы узнаете, как пользоваться документацией к библиотеке, как работать с модулями расширения, реализующими стандартные средства взаимодействия, а также познакомитесь с некоторыми полезными приемами.

Полезные советы и сведения о библиотеке jQuery

Итак, библиотека jQuery значительно облегчает процесс программирования, но это еще не значит, что нельзя упростить работу с самой библиотекой. В этом разделе собраны подсказки, которые помогут лучше понять, как устроена библиотека и как ее использовать.

Конструкция `$()` равнозначна функции `jQuery()`

В статьях и блогах, посвященных библиотеке jQuery, часто можно встретить следующий программный код:

```
jQuery('p').css('color','#F03');
```

Вы уже знакомы с селектором `$('p')`, который выбирает на странице все элементы абзаца, `p`, но при чем здесь функция `jQuery()`? На

самом деле это одно и то же. Приведенный программный код можно записать так:

```
$('p').css('color','#F03');
```

Конструкция `$()` равнозначна функции `jQuery()`: они взаимозаменяемы. Джон Резиг (John Resig), создатель библиотеки jQuery, в какой-то момент понял, что при использовании созданного им инструмента конструкция `jQuery()` будет встречаться чуть ли не в каждой строчке, и решил облегчить жизнь программистам, заменив `jQuery()` на `$()`.

Таким образом, при написании кода можно пользоваться как функцией `jQuery()`, так и `$()` — в зависимости от того, что вам больше нравится. Но так как конструкция `$()` все-таки короче, вы, вероятно, (как и большинство программистов) предпочтете ее.

ПРИМЕЧАНИЕ

Функцию `$()` использует еще одна библиотека языка JavaScript — Prototype (**www.prototypejs.org**). Если вы тоже применяете библиотеку Prototype на своем сайте, то вам следует использовать функцию `jQuery()`. Кроме того, специально для таких случаев в библиотеку jQuery включена функция `.noConflict()`. Ее подробное описание можно найти на странице: **api.jquery.com/jQuery.noConflict/.**

Сохранение выборок в переменных

Каждый раз, когда вы выбираете элемент страницы с помощью селектора `$()` — скажем, `$('#tooltip')` — происходит обращение к функции `jQuery()`. И при каждом обращении браузер посетителя инициирует выполнение некоторого программного кода, что часто замедляет работу программ.

Допустим, нужно применить несколько функций библиотеки jQuery к следующей выборке:

```
$('#tooltip').html('<p>Трубкозуб</p>');
$('#tooltip').fadeIn(250);
$('#tooltip').animate({left : 100px},250);
```

В приведенном коде в первой строке происходит выбор элемента с идентификатором `tooltip`, в который затем вставляется элемент аб-

заца — p. Во второй строке снова выполняется выбор этого же элемента, на этот раз для того чтобы реализовать его плавное отображение. И, наконец, в третий раз этот элемент выбирается для того, чтобы сместить его левый край (свойство `left`) на 100 пикселов. Каждая из этих операций — `$('#tooltip')` — вызывает функцию `jQuery()`. Так как все три строки работают с одним элементом, достаточно было выбрать этот элемент один раз.

Одним из способов, описанных ранее в разделе «Связывание функций» главы 4, является формирование цепочки методов. После селектора, выбирающего нужный элемент, выстраивается цепочка функций, как например, в следующем коде:

```
$('#tooltip').html('<p>Трубкозуб</p>').fadeIn(250). ↵
animate({left :100px},250);
```

Но иногда цепочки получаются громоздкими и запутанными. Один из способов решения этой проблемы заключается в добавлении пустой строки после каждого связанного метода. Поскольку JavaScript не так чувствителен к использованию пробелов и пустых строк, вы можете сделать следующее:

```
$('#tooltip').html('<p>Трубкозуб</p>')
 .fadeIn(250)
 .animate({left:100px},250);
```

Несмотря на то, что функции `.html`, `.fadeIn` и `.animate` находятся на разных строках, они являются частью одной длинной инструкции, состоящей из связанных методов.

Можно поступить иначе: один раз вызвать функцию `jQuery()` и сохранить результат ее выполнения в переменной, которую затем можно будет использовать повторно.

Вот как можно переписать приведенный выше программный код:

```
1 var tooltip = $('#tooltip')
2 tooltip.html('<p>Трубкозуб</p>');
3 tooltip.fadeIn(250);
4 tooltip.animate({left : 100px},250);
```

Теперь в первой строке вызывается функция `jQuery()` для выбора элемента с идентификатором `tooltip`, а полученный элемент сохраняется в переменной с именем `tooltip`. Таким образом, вы избавляете

себя от необходимости постоянно повторять процедуру выбора: теперь достаточно применить нужную функцию к переменной `tooltip` (которая содержит выборку jQuery).

При использовании этой стратегии перед именем переменной, в которой хранится выборка, обычно ставят знак `$`. Это делается для того, чтобы выделить переменные, содержащие результат работы селектора `jQuery()`, среди переменных других типов: строк, переменных, массивов, литералов объектов и т. п. Например:

```
var $tooltip = $('#tooltip');
```

При работе с событиями выборку также удобно сохранять в переменной. Как вы помните из раздела «Ключевые слова `this` и `$(this)`» главы 4, в теле обработчиков событий часто используется переменная `$(this)` — ссылка на элемент, к которому относится данное событие. Однако при каждом обращении к переменной `$(this)` происходит неявный вызов функции `jQuery()`, поэтому частое ее использование в теле обработчика впустую растрачивает вычислительные мощности компьютера. Чтобы избежать этого, сохраните результат выполнения селектора `$(this)` в переменной; выполнив эту операцию в первых строках обработчика, вы сможете далее в пределах этой функции обращаться к новой переменной и исключить повторные вызовы функции `jQuery()`:

```
$('a').click(function() {
var $this = $(this); // сохранение ссылки на элемент а
$this.css('outline','2px solid red');
var href = $this.attr('href');
window.open(href);
return false;
}); // конец click
```

Сокращение числа операций по добавлению контента

В главе 4 вы познакомились с некоторыми функциями библиотеки jQuery, которые позволяют добавлять содержимое в элементы страницы. Функция `.text()`, например, дает возможность заменить текст в элементе, а функция `.html()` заменяет HTML-код (см. раздел «Добавление содержимого на страницу» главы 4). Если бы вам понадоби-

лось вставить в элемент span с идентификатором passwordError сообщение об ошибке, можно было бы написать:

```
$('#passwordError').text('Пароль должен содержать ↵
не менее 7 знаков.');
```

Существуют также функции, позволяющие вставить содержимое после определенного элемента (функция append()) или до него (функция prepend()) (см. раздел «Добавление содержимого на страницу» главы 4).

Возможность добавлять и менять содержимое позволяет создавать сообщения об ошибках и всплывающие подсказки (см. раздел «Добавление всплывающих подсказок» главы 9) или даже вставлять «броские цитаты» (см. раздел «Автоматические «броские цитаты» главы 4), но при этом увеличивается нагрузка на браузер. Каждая подобная операция влечет за собой множество вспомогательных действий — фактически меняется структура модели DOM (см. раздел «Объектная модель документа (DOM)» главы 4), а это значит, что браузеру приходится изрядно поработать, хоть этого и не видно. Частое изменение содержимого может существенно снизить производительность страницы.

Обратите внимание, что на производительности страницы сказывается не объем содержимого, а лишь частота обновлений. Предположим, вы хотите создать всплывающую подсказку: при наведении указателя мыши на элемент формы должен появиться, например, элемент div с некоторым содержимым. Добавим нужный элемент формы и содержимое на страницу следующим образом:

```
1 // добавление элемента div после HTML-кода элемента
2 $('#elemForTooltip').append('<div id="tooltip"></div');
3 // добавление заголовка к элементу tooltip
4 $('#tooltip').append('<h2>Заголовок подсказки</h2>');
5 // добавление содержимого
6 $('#tooltip').append('<p>Содержимое подсказки</p>');
```

Приведенный код описывает все нужные действия. Строка 2 добавляет div к элементу, на который будет наведен указатель мыши; строка 4 определяет заголовок элемента tooltip, а строка 6 добавляет к подсказке содержимое абзаца. Однако три операции append() триж-

ды меняют структуру модели DOM. Это создает нагрузку на браузер, и если у нас получится уменьшить число модификаций модели DOM, производительность страницы заметно возрастет.

Мы можем заменить все операции `append()` в примере одной-единственной: создадим сразу весь HTML-код подсказки и сохраним его в переменной, а затем вставим содержимое этой переменной на страницу. В результате получим примерно такой программный код:

```
1 var tooltip = '<div id="tooltip"><h2>Заголовок ↵
подсказки</h2> <p>Содержимое подсказки</p></div';
2 $('#elemForTooltip').append(tooltip);
```

ПРИМЕЧАНИЕ

Символ ↵ (возврат каретки) в конце строки кода указывает, что следующая строка является на самом деле продолжением предыдущей. Длинные строки кода на языке JavaScript могут не помещаться в одну книжную строку, поэтому приходится разбивать их на две части. Также обратите внимание на то, что в случае со строкой JavaScript чувствителен к пробелам и возврату каретки. Вы не можете разбить строку на несколько строк так:

```
var longString = "Пришло время всем хорошим людям
объединиться для помощи своей стране";
```

В этом случае у вас возникнет синтаксическая ошибка.

В первой строке приведенного программного кода создается переменная, содержащая HTML-код подсказки, во второй строке данный код вставляется в элемент страницы. То есть в данном случае выполняется лишь одна операция `append()` и, в зависимости от используемого браузера, полученный программный код будет работать примерно в 20 раз быстрее предыдущего, с тремя функциями `append()`.

Подводя итоги: если на страницу нужно добавить фрагмент HTML-кода, делайте это с помощью одной операции, а не нескольких.

Оптимизация селекторов

Когда говорят о гибкости библиотеки jQuery, подразумевают, что один и тот же результат можно получить различными способами. Например, есть несколько способов выбрать элемент на странице: восполь-

зоваться селектором CSS или — для более точных операций — функциями обхода модели DOM, которые рассматриваются в разделе «Обход дерева DOM» далее в главе. Сейчас мы рассмотрим приемы, которые ускоряют работу операций поиска и, соответственно, повышают производительность программ на языке JavaScript.

- **Используйте идентификаторы везде, где только можно.** Самый быстрый способ выбрать элемент на странице — использовать идентификатор. Эта возможность существует в браузерах с момента появления языка JavaScript и до сих пор этот считается самым быстрым методом выбора. Если для вас важна производительность, лучше присвоить идентификатор каждому выбираемому элементу, а не полагаться, например, на селектор потомков.

- **Используйте идентификатор в селекторе потомков.** По идентификатору можно выбрать только один элемент. Что же делать, если нужно обработать целую группу, например, все элементы ссылок, `a`, в определенном элементе `div` или все абзацы на странице? Если все выбираемые элементы находятся в теле элемента с заданным идентификатором, можно воспользоваться селектором потомков, содержащим искомый идентификатор. Предположим, что на странице нужно выбрать все элементы ссылок, `a`. Так получилось, что все эти элементы оказались в `div` с идентификатором `main`. Селектор `$('#main a')` сработает гораздо быстрее селектора `$('a')`.

- **Применяйте функцию `.find()`.** Библиотека jQuery предусматривает функцию для поиска элементов в выборке. Она работает подобно селектору потомков: ищет одни HTML-элементы внутри других. Об этой функции вы подробнее узнаете в разделе «Обход дерева DOM» далее в главе. Если вкратце, то сначала выполняется выбор элементов с помощью селектора jQuery, а затем применяется функция `.find()`, которой передается результат работы селектора. Другими словами конструкцию `$('#main a')` можно записать так:

```
$('#main').find('a');
```

Иногда замена селектора потомков функцией `.find()` приводит к тому, что поиск выполняется в два раза быстрее.

ПРИМЕЧАНИЕ

Скорость работы функции `.find()` можно проверить на сайте: **jsperf.com/sawmac-selector-test**.

- **Избегайте излишней детализации.** Возможно, вы привыкли использовать отдельные правила для создания стилей, управляющих конкретными элементами на странице. Например, правило `#main .sidebar .note ul.nav li a` очень конкретно, но может медленно обрабатываться функциями библиотеки jQuery. Лучше использовать селектор потомков, который и короче, и точнее, — например, `$('.sidebar .nav a')` — или применить функцию `.find()`, о которой говорилось в предыдущем пункте: `$('#main').find('.sidebar').find('.nav a')`

Использование документации к библиотеке jQuery

На сайте библиотеки jQuery (**docs.jquery.com/**) размещена очень подробная документация (рис. 15.1).Там вы найдете ссылки на разделы о том, как начать работу с библиотекой jQuery, куда обратиться за помощью, где искать практические руководства и многое другое. Однако самый важный раздел этого сайта посвящен интерфейсу jQuery API*, в который входят функции из библиотеки jQuery. Это, например, обработчики событий, о которых шла речь в главе 5 (`.click()`, `.hover()` и др.), функции для работы с таблицами стилей CSS, с которыми вы познакомились в главе 4 (`.css()`, `.addClass()` и `.removeClass()`) и, наконец, главная функция библиотеки — `$()` — позволяющая выбирать элементы на странице.

На странице сайта с документацией jQuery вы найдете информацию о каждом методе библиотеки. На главной странице все они перечислены в алфавитном порядке, что полезно в том случае, когда вы знаете название функции. В противном случае вам может быть трудно ориентироваться в этом обширном списке. Вместо этого вы, вероятно, предпочтете воспользоваться навигационной панелью слева. Далее перечислены основные категории:

- **Селекторы (api.jquery.com/category/selectors/)**. В эту категорию входят самые полезные функции библиотеки jQuery. Заглядывайте на эту страницу чаще: здесь перечислено много различных способов выбора элементов страницы с помощью средств библиотеки jQuery. Кое-что вы уже знаете из главы 4, однако этот раздел тоже может многому научить.

* Английская аббревиатура словосочетания «интерфейс программирования приложений» (Application Programming Interface) *(прим. ред.)*.

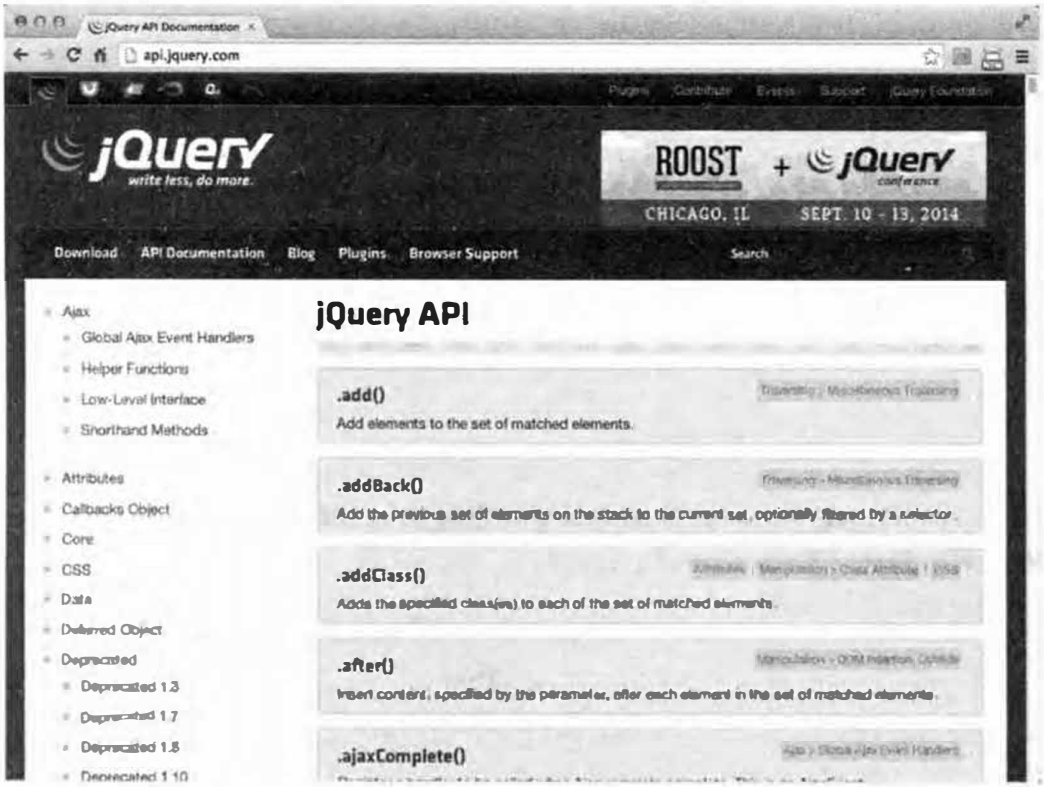

Рис. 15.1. На главной странице сайта с документацией к библиотеке jQuery много полезных ссылок. В левой части экрана перечислены все функции библиотеки jQuery. В некоторых случаях они сгруппированы по темам, например, функции Ajax, функции Event и функции Manipulation

Поскольку библиотека jQuery предусматривает так много функций, относящихся к селекторам, в левой панели вы найдете подкатегории, в которых сгруппированы базовые селекторы, селекторы формы, селекторы атрибутов и т. д.

- **Атрибуты (api.jquery.com/category/attributes/)**. На этой странице собраны функции библиотеки jQuery, которые помогают получить и установить значения атрибутов для элементов кода HTML. С помощью этих функций можно, например, назначить элементу некоторый класс, получить или задать значение атрибута (скажем, атрибута `href` для элемента ссылки, `a`) или получить значение элемента формы.

ПРИМЕЧАНИЕ

Одна и та же функция может быть описана в разных разделах сайта с документацией по библиотеке jQuery. Например, функция `.val()`, которая получает и устанавливает значения полей формы, встречается как в категории «Атрибуты» (Attributes), так и в категории «Формы» (Forms).

- **Функции обхода модели DOM (api.jquery.com/category/traversing/)**. Содержит ссылки на функции для работы с наборами элементов страницы. Функция `.find()`, например, позволяет найти элемент в выборке jQuery. Представьте, что вам нужно сначала выбрать элемент на странице (например, элемент `ul`), выполнить с ним некоторые действия (назначить этому элементу класс, плавно отобразить и т. п.), а затем найти в его теле другой элемент для обработки (например, в исходном элементе `ul` найти элемент `li`). Библиотека jQuery содержит множество функций для обхода элементов HTML-кода; с некоторыми из них вы познакомитесь в этой главе.
- **Управление (api.jquery.com/category/manipulation/)**. Добавление и удаление элементов страницы подразумевает работу с моделью DOM этой страницы (Document Object Model, объектная модель документа). В этот раздел документации включены сведения о функциях, которые служат для редактирования страницы. С некоторыми из них вы уже познакомились в главе 4 (см. раздел «Добавление содержимого на страницу» главы 4) — например, с функцией `.html()`, которая добавляет на страницу код HTML, функцией `.text()` для вставки текста на страницу и др. Функции в этом разделе представляют важную часть библиотеки, поскольку в программировании на языке JavaScript широко используется динамическое изменение вида и содержимого страницы.
- **Таблицы стиля CSS (api.jquery.com/category/css/)**. Функции библиотеки jQuery, приведенные в этой категории, предназначены для получения или установки значений свойств, относящихся к таблицам стилей CSS. С помощью этих функций можно, например, назначить элементу некоторый класс или же удалить уже существующий класс, непосредственно задать значение для свойства таблицы CSS, а также управлять высотой, шириной и местоположением элемента или же просто получать текущие значения этих атрибутов. Некоторые из функций данной категории описаны в разделе «Чтение и изменение свойств CSS» главы 4.
- **События (api.jquery.com/category/events/)**. В главе 5 вы узнали о том, как с помощью функций библиотеки jQuery запрограммировать реакцию на действия пользователя — наведение указателя мыши на ссылку или нажатие кнопки на странице. На странице категории «События» (Events) перечислены функции для обработки событий. А в разделе «Поиск в строке: техника `indexOf()`» главы 16 этой

книги можно прочесть о некоторых расширенных возможностях обработчиков событий.

- **Эффекты (api.jquery.com/category/effects/)**. В этом разделе представлена информация о функциях библиотеки jQuery, предназначенных для создания эффектов. Некоторыми из этих функций являются, например, `.slideDown()`, `.fadeIn()` и `.animate()`, с которыми вы познакомились в главе 6.
- **Формы (api.jquery.com/category/forms/)**. А в эту категорию входят функции для работы — внимание, барабанная дробь! — с формами. Здесь, в основном, перечислены события, применяемые к элементам форм, а также функция `.val()` для получения или установки значения поля формы и, кроме того, несколько функций для отправки форм по технологии Ajax (см. главу 13).
- **Ajax (api.jquery.com/category/ajax/)**. Здесь перечислены функции, выполняющие динамическое обновление страницы с использованием информации, получаемой или отправляемой на веб-сервер. О технологии Ajax вы можете прочесть в главе 13 данной книги.
- **Утилиты (api.jquery.com/category/utilities/)**. Библиотека jQuery содержит немало функций, которые могут существенно упростить решение повседневных задач, например, помогают найти элемент в массиве, обработать каждый элемент массива или любого другого объекта (функция `$.each()`, описанная в разделе «Работа с каждым элементом выборки» главы 4) — все, что может понадобиться увлеченному программисту. Возможно, на данном этапе карьеры эти функции вам еще не пригодятся: они не реализуют какие-то особые эффекты и не служат для обновления содержимого или внешнего вида страницы. Однако позже, когда вы наберетесь опыта, загляните на эту страницу.

Существует еще несколько категорий реже используемых функций, которые следует иметь в виду:

- **Данные (api.jquery.com/category/data/)**. На этой странице перечислены функции для внесения данных в элементы страницы. В библиотеке jQuery есть функция `.data()`, которая служит для записи данных в элемент и позволяет добавить пару имя/значение в любой элемент страницы. Элемент, таким образом, превращается в некое подобие небольшой базы данных. Эта и другие функции для работы с данными могут пригодиться при разработке веб-приложений, в которых нужно сохранять данные или отслеживать изменения в них. Неплохое обзорное руководство по использованию этих функций можно найти на странице: **tutorialzine.com/2010/11/jquery-data-method/.**

- **Отложенный объект (api.jquery.com/category/deferred-object/).** Не пытайтесь разобраться с этой категорией сейчас, запомните только, что отложенный объект в библиотеке jQuery — это довольно сложное понятие. (Во вступительной статье сказано, что отложенный объект — это «вспомогательный объект, который можно включать в цепочки и который служит для регистрации нескольких обратных вызовов в очереди функций обратного вызова, обращения к очереди функций обратного вызова, а также сообщения об удачном или неудачном выполнении синхронных и асинхронных функций».) Этот объект чаще всего применяется для управления порядком выполнения функций, организованных в очереди. Подробная информация о категории находится на соответствующей странице сайта с документацией по библиотеке jQuery.
- **Размеры (api.jquery.com/category/dimensions/).** Здесь описаны функции для определения высоты и ширины объектов. Эти же функции перечислены и в ранее упомянутой категории «Таблицы стиля CSS» (CSS).
- **Служебные функции (api.jquery.com/category/internals/).** Категория содержит много функций самого разнообразного назначения. Свойство `.jquery`, например, возвращает версию библиотеки jQuery, использующуюся на данной странице:

```
// открыть окно оповещения с номером версии
   alert($().jquery); // например, 1.6.2
```

  Скорее всего, вы проживете долгую счастливую жизнь, так и не воспользовавшись ни одной из этих функций.
- **Смещение (api.jquery.com/category/offset/).** В этой категории перечислены функции, определяющие местоположение объекта относительно экрана или родительского элемента. Эти функции используются при определении или задавании позиции элемента на странице.

Описание страницы сайта с документацией

Каждой функции, включенной в библиотеку jQuery, на сайте выделена отдельная страница, описывающая, что именно делает эта функция и как она работает.

Рисунок 15.2 изображает часть страницы, посвященной функции `.css()`. На странице приводятся имя функции (в данном случае это

`.css()`), а также список категорий и подкатегорий, к которым данная функция относится. Щелчок по названию категории или подкатегории инициирует переход на страницу, где перечислены все функции этой категории.

Иногда функции имеют двойное или даже тройное назначение и работают по-разному, в зависимости от типа и количества переданных им аргументов. Такие функции указываются несколько раз. Например, на рис. 15.2 функция `.css()` приведена два раза (цифры 1 и 2 на рисунке).

Рис. 15.2. В описании каждой функции библиотеки jQuery приводятся все возможные способы использования этой функции. В нашем случае под заголовком «Содержимое:» (Contents:) видно, что функции `.css()` можно передать один (№ 1) или два (№ 2) аргумента. Результат работы функции зависит от выбранного варианта. Щелкнув по примеру функции (обведено), можно перейти к описанию выбранного способа использования функции

В первом случае (№ 1) функция записана как `css(propertyName)`. Это значит, что функции передается только один аргумент — имя свойства CSS. При этом функция возвращает значение указанного свойства для данного элемента (обратите внимание на выражение `Returns:String` в правой части темной полосы в середине рисунка). Ниже на странице

написано, что функция css, которой передан один аргумент, возвращает строку. Предположим, что нам нужно определить ширину элемента `div` с идентификатором `tooltip`; для этого напишем следующий код:

```
var tipWidth = $('#tooltip').css('width');
// получение значения ширины
```

В этом программном коде функция получает аргумент `'width'` и возвращает строку.

(Однако в данном случае, поскольку искомым свойством является ширина, строка будет состоять из значения ширины в пикселах, например, '300').

ПРИМЕЧАНИЕ

Библиотека jQuery предлагает другой вариант — `css( propertyNames )`, который также возвращает значения CSS-свойства элемента. Однако вместо того, чтобы принять только одно имя свойства и возвратить одно значение, этот вариант принимает массив (см. раздел «Массивы» главы 2) с именами свойств и возвращает объект JavaScript (см. раздел «Одновременное изменение нескольких свойств CSS» главы 4), состоящий из пар имя/значение. Имя соответствует имени свойства CSS из массива, а значение представляет собой текущее значение этого свойства CSS конкретного HTML-элемента.

Второй способ использования функции `.css()` состоит в том, что функция записывается в виде `css(propertyName, value)`; это означает, что она получает два аргумента — имя свойства CSS и его значение. В этом случае выбранному элементу присваивается заданное свойство CSS. Например, чтобы задать ширину элемента `div` с идентификатором `tooltip`, нужно передать в качестве первого аргумента строку `width`, а в качестве второго — требуемое значение:

```
$('#tooltip').css('width',300);
// задать ширину div равной 300 пикселам
```

В документации указаны еще два способа задать свойства CSS, используя функцию `.css()`:

- **`.css( propertyName, function )`**. Эта запись означает, что для свойств CSS можно задавать значения, вычисляемые динамически с помощью указанной функции. Это удобно, когда, например,

надо установить различные значения для разных элементов коллекции; например, для ряда элементов `div`, у которых свойство `left` должно определять местоположение таким образом, чтобы они расположились последовательно друг за другом. Очевидно, что в этом случае значение свойства `left` для каждого из элементов должно быть отличным от остальных.

- **`.css( properties )`**. Этот формат описан в разделе «Чтение и изменение свойств CSS» главы 4 и означает, что функции можно передать литерал объекта и таким образом задать значения сразу нескольких свойств CSS.

Важно понимать, что одна и та же функция библиотеки jQuery может принимать различные аргументы и выполнять различные действия. Функция `.css()`, например, может, как получать, так и устанавливать значения свойств CSS. Вы еще не раз столкнетесь с тем, что функции библиотеки jQuery работают как в качестве «считывателей» (получая информацию об элементе), так и в качестве «присваивателей» (задавая определенное значения свойства элемента).

Страница документации описывает все возможные варианты применения данной функции и, как правило, содержит практические примеры использования. Документация по библиотеке jQuery регулярно обновляется и представлена в удобной и простой форме. Вам не помешает время от времени читать документацию, особенно касающуюся функций, с которыми вы часто работаете.

Обход дерева DOM

Итак, вы уже знаете, как выбирать элементы на странице с помощью функции `jQuery()` и основного синтаксиса CSS: селектор `$('p')`, например, выбирает все абзацы. Над выбранными элементами можно выполнять различные действия, например, отнести их к некоторому классу или же удалить присвоенный класс, изменить значение свойства CSS или заставить элемент исчезнуть. Но иногда требуется найти *другие* элементы, относящиеся к элементам из первоначальной выборки. В терминологии языка JavaScript эта операция называется *обходом модели DOM* (Document Object Model, объектная модель документа).

Чаще всего обход модели DOM происходит, когда вы присоединяете событие к одному элементу, а затем хотите выполнить некое действие над

другим элементом. Допустим, существует элемент `div` с идентификатором `gallery`, который содержит несколько эскизов изображений. Нужно, чтобы после щелчка по этому элементу что-нибудь происходило: изображение должно уменьшаться или увеличиваться, дрожать или еще что-нибудь в этом роде. Событие прикрепляется к элементу `div` таким образом:

```
$('#gallery').click(function() {
}); // конец click
```

В функцию нужно добавить программный код для анимации изображений. То есть пользователь будет щелкать кнопкой мыши по элементу `div`, но реагировать должны *изображения*. В приведенном фрагменте кода вы выбрали элемент `div`, значит ключевое слово `$(this)` станет ссылаться на этот элемент (если для вас это — новость, вернитесь в раздел «Ключевые слова `this` и `$(this)`» главы 4 и вспомните, что означает конструкция `$(this)`). После выбора элемента `div` нужно найти в нем все изображения. К счастью, в библиотеке jQuery для этих целей существует функция `.find()`. Она предназначена для создания новой выборки на основе текущей; *внутри* существующей выборки выполняется поиск элементов, удовлетворяющих новому условию поиска. Для выбора изображений в элементе `div` добавим в программный код строку, выделенную полужирным шрифтом:

```
$('#gallery').click(function() {
      $(this).find('img');
}); // конец click
```

Конструкция `$(this).find('img')` создает новую выборку элементов; поскольку ключевое слово `$(this)` ссылается на элемент `div`, функция `find('img')` выбирает все изображения внутри этого элемента. Конечно, приведенный фрагмент кода никак не обрабатывает элементы новой выборки, но вы и сами можете добавить любые эффекты, о которых узнали в предыдущих главах. Например, вот так можно заставить изображения медленно исчезать, а затем медленно появляться:

```
$('#gallery').click(function() {
    $(this).find('img').fadeTo(500,.3).fadeTo(250,1);
}); // конец click
```

Как вы уже читали в разделе «Постепенное появление и исчезновение элементов» главы 6, функция `fadeTo()` в качестве аргументов

принимает значение длительности и степень непрозрачности. Таким образом, при выполнении данного программного кода все изображения будут плавно исчезать, достигая степени непрозрачности 30% в течение 500 мс, а затем вновь появляться — достигать 100 % степени непрозрачности — в течение 250 мс (чтобы увидеть эффект собственными глазами, откройте файл *15_01.html* в папке с руководствами *глава15*).

На практике обход модели DOM используется так часто, что разработчики включили в библиотеку jQuery множество функций для обработки выборок и поиска элементов, связанных с выбранными. Проиллюстрируем работу этих функций, воспользовавшись небольшим фрагментом HTML-кода, представленным на рис. 15.3: **api.jquery.com/category/manipulation/**. Это элемент `div` с идентификатором `gallery`, содержащий четыре ссылки, каждая из которых заключает изображение.

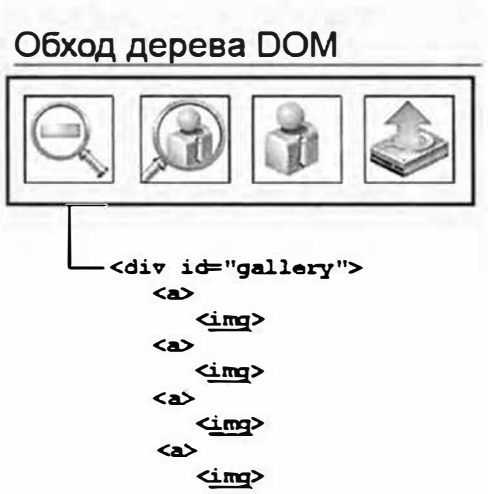

Рис. 15.3. При создании выборки (например, чтобы добавить для элемента обработчик события, как для изображенных на рисунке элементов `a`) программист часто планирует работать не с выбранными элементами, а с другими, которые как-то связаны с выбранными (скажем, добавить описание к элементу `div` или изменить элемент `img`). В таких случаях на помощь приходят функции обхода дерева DOM из библиотеки jQuery

Как уже обсуждалось в разделе «Объектная модель документа (DOM)» главы 4, связи между элементами на странице описываются как «родственные». Например, на рис. 15.3 элемент `div` является родительским по отношению к элементам `a`, а каждый элемент `a` в свою очередь является родительским для размещенного в нем элемента `img`. И наоборот, элементы `a` считаются дочерними по отношению к элементу `div`, а каждый отдельный элемент `a` находится на том же уровне, что

и остальные. Каждый элемент `img` является дочерним по отношению к тому элементу `a`, в который он заключен, и поскольку в элементе `a` нет других элементов, у `img` нет элементов того же уровня.

Рассмотрим несколько наиболее полезных функций обхода дерева DOM:

- Функция **`.find()`** находит заданные элементы в текущей выборке. Вы записываете выборку jQuery, добавляете к ней функцию `.find()` и передаете ей CSS-селектор:

```
$('#gallery').find('img')
```

  Этот код выполняет поиск всех изображений в элементе `div` с идентификатором `gallery`. Разумеется, такой же результат можно получить, используя селектор потомков, например, `$('#gallery img')`. Как мы говорили ранее, функция `.find()` чаще всего применяется в ситуациях, когда приходится иметь дело с уже обработанным набором элементов. Допустим, к исходному набору было присоединено событие `click`, тогда для дальнейших действий мы создаем новую выборку на основе существующей.

  Функция `.find()` используется для поиска потомков (элементов, размещенных в теле другого элемента) в существующей выборке. Таким образом, в примере на рис. 15.3 функция `.find()` может использоваться для поиска элементов `a` или `img` в выбранном элементе `div`.

ПРИМЕЧАНИЕ

Заметные преимущества при использовании функции `.find()` перечислены на предыдущей странице. С помощью функции `.find()` выбор элементов выполняется гораздо быстрее, чем при использовании селектора потомков.

- Функция **`.children()`** похожа на функцию `.find()`. Она также принимает в качестве аргумента селектор, но выполняет поиск лишь в наборе непосредственных потомков текущей выборки. Рассмотрим, к примеру, элемент `div`, в теле которого содержится несколько других элементов `div`. Нужно сделать так, чтобы при щелчке по главному элементу появлялись изначально скрытые элементы, обведенные красным контуром. С помощью функции `.find()` эту задачу можно решить так:

```
$('#mainDiv').click(function() {
$(this).find('div').show().css('outline','red 2px solid');
});
```

Однако если в одном из элементов, находящихся внутри основного, тоже содержатся элементы `div`, возникнет проблема. Код `.find('div')` выберет абсолютно все элементы `div`, в том числе и содержащиеся в других элементах. Может получиться, что на странице будут обведены все имеющиеся элементы `div`, в то время как должны быть обведены только непосредственно дочерние элементы `div` основного элемента `div`. Для решения данной проблемы перепишем этот фрагмент кода, воспользовавшись функцией `.children()`:

```
$('#mainDiv').click(function() {
$(this).children('div').show().css('outline','red ↵
2px solid');
});
```

Теперь будут выбраны лишь непосредственные дочерние элементы основного элемента `div` и пропущены любые элементы, размещенные внутри дочерних элементов `div`.

- Функция **`.parent()`**. В то время как функция `.find()` находит элементы, находящиеся внутри выбранного элемента, функция `.parent()` перемещается по дереву DOM в поисках родителя текущего элемента. Такая функция удобна, например, если вы присоединяете событие `hover` к элементам `a` на рис. 15.3, но при этом необходимо еще обработать элемент `div` (например, добавить рамку или изменить цвет фона). В таком случае для поиска элемента `div` будет использована функция `.parent()`:

```
$('#gallery a').hover(
   function() {
var $this = $(this);
// добавление контура к ссылке
$(this).css('outline','2px solid red');
// добавление цвета фона элементу div
$(this).parent().css('backgroundColor','white');
   },
   function() {
```

```
var $this = $(this);
// удаление контура ссылки
$(this).css('outline','');
// удаление цвета фона элемента div
$(this).parent().css('backgroundColor','');
        }
); // конец hover
```

В этом примере при наведении указателя мыши на ссылку вокруг ссылки появится рамка, затем будет выбран родительский элемент этой ссылки (элемент `div`) и у элемента изменится цвет фона. После того как указатель мыши покинет ссылку, рамка пропадет, и цвет фона снова изменится на исходный (подробную информацию о событии `.hover()` см. в разделе «События наведения и смещения указателя мыши» главы 5). (Пример работы с функцией доступен в файле *15_02.html* в папке *глава15*.)

- Функция **`.closest()`** находит ближайшего предка, удовлетворяющего заданному селектору. В отличие от функции `.parent()`, которая выбирает непосредственного родителя текущего элемента, функция `.closest()` принимает в качестве аргумента селектор и находит ближайшего предка, соответствующего этому селектору. На рис. 15.3, например, каждое изображение заключено в элемент `a`, другими словами, элемент `a` является родителем элемента изображения. Однако что делать, если нужно выбрать элемент `div`, окружающий элементы `a` (предка, находящегося выше в HTML-цепочке)? Эта задача решается с помощью функции `.closest()`:

```
1 $('#gallery img').click(function() {
2  var $this = $(this);
3  $(this).css('outline','2px red solid');
4  $(this).closest('div').css('backgroundColor','white');
5}); // конец click
```

В строке 3 код `$(this)` ссылается на элемент `img`; код `closest('div')` ищет ближайшего предка, которым является элемент `div`. Ближайшим предком на самом деле является элемент `a`, но поскольку это не `div`, библиотека jQuery его пропустит и закончит поиск не раньше, чем найдет элемент `div`.

- Функция **.siblings()** пригодится в случаях, когда нужно выбрать элемент, который находится на одном уровне с уже выбранным. Вспомните снова HTML-код на рис. 15.3: теперь нам нужно, чтобы при щелчке по ссылке остальные ссылки плавно исчезали или появлялись. В данном случае событие — `click` — будет задано для одного элемента ссылки, но эффект должен быть применен ко всем ссылкам в том же элементе `div`. Иначе говоря, нужно выбрать все ссылки, находящиеся на одном уровне с нажатой. Вы можете добиться этого так:

```
1 $('#gallery a').click(function() {
2  $(this).siblings().fadeTo(500,.3).fadeTo(250,1);
3}); // конец click
```

  В приведенном примере код `$(this)` относится к нажатой ссылке, поэтому функция `.siblings()` выберет остальные ссылки из того же элемента `div`.

  Функция `.siblings()` также может принимать в качестве аргумента имя селектора, чтобы ограничить круг поиска нужным элементом. Например, в элементе `div` на рис. 15.3 элементам ссылок предшествуют заголовок и абзац. Поскольку и заголовок, и абзац находятся в элементе `div` наряду с элементами ссылок `<a>`, они тоже являются элементами того же уровня, что и элемент `a`. Другими словами, при выполнении приведенного программного кода щелчок по ссылке приведет к плавному исчезновению и появлению не только ссылок, но и заголовка, и абзаца. Чтобы эффект распространялся только на ссылки, нужно изменить строку 2:

```
$(this).siblings('a').fadeTo(500,.3).fadeTo(250,1);
```

  Часть `'a'` в коде `siblings('a')` ограничивает выборку элементами `a`, находящимися на том же уровне, что и исходный элемент. Пример работы с функцией можно найти в файле *15_03.html* в папке *глава15*.

- Функция **.next()** находит следующий элемент того же уровня в текущей выборке. С работой этой функции вы уже познакомились в разделе «Создание страницы ЧаВо на практике» главы 5. В этом руководстве последовательные щелчки по тексту вопроса приводят к тому, что элемент `div` с ответом появляется и снова исчезает. Каждый вопрос представлен элементом `h2`, а каждый ответ — `div`, который размещен следом за соответствующим элементом `h2`. Заголовок и `div` являют-

ся элементами одного уровня, но они также являются элементами одного уровня для остальных вопросов и ответов на этой странице. Поэтому при щелчке по заголовку нужно выбрать именно следующий элемент того же уровня. Подобно функции `.siblings()`, функция `.next()` принимает в качестве аргумента селектор, поэтому процесс выбора можно ограничить поиском следующего элемента того же уровня и заданного типа. (Пример работы с функцией можно найти в файле *готовый_05_02.html* в папке *глава05*.)

- Функция **`.prev()`** действует так же, как и функция `.next()`, но ищет ближайший предшествующий элемент того же уровня.

ПРИМЕЧАНИЕ

Подробная информация о функциях для обхода дерева DOM, приведена на сайте: **api.jquery.com/category/traversing/**

Дополнительные функции для работы с кодом HTML

Веб-программисты постоянно сталкиваются с необходимостью динамически добавлять, удалять или менять HTML-код на странице. Приведем простейший пример: вы хотите, чтобы при отправке формы после нажатия соответствующей кнопки на экране появлялось сообщение: «Подождите, идет передача информации». Или чтобы при наведении указателя мыши на фотографию сверху в рамке появлялась надпись и сведения об авторе фотографии. В обоих случаях придется добавить код HTML на исходную страницу. С самыми нужными функциями вы уже познакомились в главе 4в разделе «Добавление содержимого на страницу». Напомним:

- Функция **`.text()`** заменяет текстовое содержимое выбранных элементов выборкой, переданной ей в качестве аргумента.

  Например:

  ```
  $('#error').text('Укажите адрес электронной почты');
  ```

- Функция **`.html()`** действует так же, как функция `.text()`, но вставляется не текстовая строка, а код HTML:

  ```
  $('#tooltip').html('<h2>Модель Avalon компании ↵
  Esquif</h2> <p>Предназначена для путешествий ↵
  на каноэ.</p>');
  ```

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Функция .end(): покончим с обходом дерева DOM

С помощью связывания функций библиотека jQuery дает возможность вместить в тот же объем программного кода гораздо больше полезных действий. Создание цепочек функций уже рассматривалась в разделе «Связывание функций» главы 4. Если вкратце, этот механизм позволяет выбрать несколько элементов страницы и затем последовательно выполнить над ними несколько операций, записав нужные функции друг за другом. Предположим, нужно выбрать все абзацы на странице и заставить их плавно скрыться, а затем снова плавно появиться. Вот как будет выглядеть соответствующий программный код:

```
$('p').fadeOut(500).fadeIn(500);
```

Цепочка может объединять неограниченное число функций, включая функции обхода дерева DOM, рассмотренные ранее. Можно, например, выбрать элемент `div`, обвести его контуром, а затем выбрать в нем все элементы `a` и изменить цвет текста ссылок. Это делается так:

```
$('div').css('outline','2px red solid'). ↵
find('a').css('color','purple');
```

Рассмотрим этот фрагмент по шагам:

1. Код `$('div')` выбирает все элементы `div`.
2. Фрагмент `.css('outline','2px red solid')` добавляет к элементу `div` красный контур толщиной в 2 пиксела.
3. Часть `find('a')` создает на базе исходной выборки (элементы `div`) новую, включающую все элементы `a` из `div`.
4. Код `.css('color','purple')` меняет цвет текста всех ссылок (но не самого элемента `div`) на фиолетовый.

Если добавить в цепочку одну из функций обхода дерева DOM, то выборка изменится. В приведенном выше коде селектор сначала выбирает элемент `div`, а затем посреди цепочки формируется новая выборка — из ссылок, содержащихся в элементе `div`. Однако иногда нужно вернуться к исходной выборке. Иными словами, нужно что-то выбрать, затем выбрать еще что-то на основе первой выборки, и, наконец, вернуться к исходной выборке. Допустим, нужно, чтобы при щелчке по элементу `<div>` со степенью непрозрачности 50 % степень непрозрачности увеличивалась бы до 100 %, изменялся бы цвет заголовка в элементе `div`, а в каждом абзаце этого элемента `div` изменялся бы цвет фона. То есть одно действие — щелчок кнопкой мыши — потребует выполнения нескольких операций с различными элементами страницы. Ниже приведен один из способов осуществить этот сценарий:

```
$('div').click(function() {
        var $this = $(this);
  $(this).fadeTo(250,1); // появление элемента div
        $(this).find('h2').css('color','#F30');
  $(this).find('p') .('backgroundColor','#F343FF');
}); // конец click
```

В этом случае связывание функций является очень полезным: троекратный вызов функции `$(this)` можно легко заменить одним, выстроив цепочку из следующих друг за другом функций. Однако при объединении функций в цепочку могут возникнуть трудности:

```
$('div').click(function() {
  $(this).fadeTo(250,1)
.find('h2')
.css('color','#F30')
.find('p')
.('backgroundColor','#F343FF');
}); // конец click
```

На первый взгляд тут все правильно, но после выполнения кода `.find('h2')`, который меняет выборку с элементом `div` на выборку, содержащую элемент `h2` из `div`, возникает проблема. При выполнении следующей функции — `.find('p')` — запустится поиск элементов абзаца, `p`, в элементе `h2`, а не `div`. К счастью, в этом случае можно воспользоваться функцией `.end()`, чтобы отменить последние изменения и вернуться к исходной выборке. В этом фрагменте кода мы воспользуемся функцией `.end()` для возврата к выборке по элементу `div` и затем выберем элементы абзаца, `p`, находящиеся в элементе `div`. Получим следующий программный код:

```
$('div').click(function() {
  $(this).fadeTo(250,1)
.find('h2')
.css('color','#F30')
.end()
.find('p')
.('backgroundColor','#F343FF');
}); // конец click
```

Обратите внимание, что в этом фрагменте кода функция `.end()` после фрагмента `.css('color','#F30')` снова возвращает выборку по элементу `div`, поэтому следующий за ней код `.find('p')` выберет все элементы абзаца, `p`, из `div`.

- Функция **.append()** добавляет код HTML в конец элемента (например, прямо перед закрывающим тегом). Эта функция отлично подходит для случаев, когда надо добавить элементы в конец списка.

- Функция **.prepend()** добавляет код HTML в начало элемента (например, сразу после открывающего тега).

- Функция **.before()** вставляет заданное содержимое перед элементами выборки.

- Функция **.after()** похожа на функцию .before(), но вставляет указанное содержимое после элементов выборки (после соответствующего закрывающего тега).

Выбор функций, разумеется, зависит от поставленной задачи. Язык JavaScript подходит для автоматизации задач веб-разработки, которые обычно выполняются вручную: написания кода HTML и создания стилей CSS для страницы. Если вы пишете программу, которая должна динамически добавлять на страницу содержимое — подсказки, сообщения об ошибках, «броские цитаты» и т. д. — просто представьте себе, как это должно выглядеть, какой для этого понадобится код HTML и какие стили CSS.

Если, например, при наведении указателя мыши на кнопку на странице должно появиться некоторое сообщение, попробуйте сначала создать страницу с этим сообщением, не используя язык JavaScript — постройте ее с помощью правил CSS и HTML-кода. Если полученная HTML/CSS модель заработает, обратите внимание, какой именно код HTML вы использовали. Может быть, HTML-код, описывающий это сообщение, находится прямо перед кодом определенного элемента? (Тогда примените функцию .before().) А может быть, код HTML находится в определенном элементе? (Тогда вспомните о функции .append() или .prepend().)

Библиотека jQuery включает также функции для удаления содержимого страницы:

- Функция **.replaceWith()** полностью заменяет выбранные элементы (тег и все, что внутри) содержимым, полученным в качестве аргумента. Кнопку отправки формы можно заменить текстом «Выполняется...» вот так:

```
$(':submit').replaceWith('<p>Выполняется...</p>');
```

- Функция **`.remove()`** удаляет выбранные элементы из модели DOM, фактически удаляя их со страницы. К примеру, для удаления со страницы элемента `div` с идентификатором `error` можно использовать такую строку кода:

  ```
  $('#error').remove();
  ```

  Возможно, функций, перечисленных выше и описанных в главе 4, вам будет достаточно, но помните, что в библиотеке jQuery есть дополнительные функции, которые позволяют управлять HTML-кодом.

- Функция **`.wrap()`** заключает каждый элемент выборки в пару HTML-тегов. Предположим, вы хотите создать необычные подписи под изображениями на странице. Выберите нужные изображения на странице и заключите их в элемент `div` класса `figure`; затем добавьте в него элементы `p` класса `caption`.

Далее отформатируйте элемент `div` и подписи с помощью таблиц CSS. Ниже представлен один из способов решения этой задачи:

```
1 // обход списка изображений
2 $('img').each(function() {
3     // сохранение ссылки на текущее изображение
4     var $this = $(this);
5     // получение свойства alt для подписи
6     var caption = $this.attr('alt');
7     // добавление HTML
8     $this.wrap('<div class="figure"></div>') ↵
.after('<p>' + caption + '</p>');
9}); // конец функции each
```

Приведенный фрагмент кода сначала выбирает на странице все изображения, а затем с помощью функции `.each()` (см. раздел «Работа с каждым элементом выборки» главы 4) обходит их в цикле. В строке 4 текущее изображение сохраняется в переменной на каждом шаге цикла (это признак хорошего стиля программирования, как говорилось в начале данной главы). В строке 6 значение атрибута `alt` текущего изображения извлекается и сохраняется в переменной `caption`. И в заключение, в строке 8, изображение заключается в элемент `div`, а после него с помощью ранее описанной функции `.after()` добавляется подпись.

ПРИМЕЧАНИЕ

Пример работы с функцией `.wrap()`, приведенный в строке 8, можно найти в файле *15_04.html*, в папке *глава15*.

Функция `.wrap()` получает полный набор элементов — `$('p').wrap('<div></div>')` — или даже набор вложенных элементов:

```
$('#example').wrap('<div id="outer"><div id="inner"> ↵
</div></div>');
```

В приведенном программном коде выборка должна быть окружена двумя элементами `div`, а значит, код будет выглядеть так:

```
<div id="outer">
      <div id="inner">
<div id="example">Исходная разметка страницы</div>
      </div>
</div>
```

- Функция **`.wrapInner()`** окружает HTML-элементами содержимое каждого элемента выборки. Предположим, код HTML содержит следующий фрагмент:

  ```
  <div id="outer">
  <p>Содержимое элемента outer</p>
  </div>
  ```

  Если браузер встретит вот такой программный код `$('#outer').wrapInner('<div id="inner"></div>');`, то HTML-код станет таким:

  ```
  <div id="outer">
  <div id="inner">
  <p>Содержимое элемента outer</p>
  </div>
  </div>
  ```

- Функция **`.unwrap()`** удаляет родительский элемент, окружающий выборку. Предположим, на странице присутствует такой фрагмент кода HTML:

```
<div>
<p>некоторый абзац</p>
<div>
```

Выполнение кода `$('p').unwrap()` изменит этот фрагмент следующим образом:

```
<p>некоторый абзац</p>
```

Внешний элемент `div` просто исчез. Обратите внимание, что в отличие от других рассмотренных функций функция `.unwrap()` не принимает аргументов — другими словами, не помещайте ничего в круглых скобках функции `.unwrap()`, иначе она не будет работать.

- Функция **`.empty()`** удалит все содержимое элементов выборки, но сама выборка при этом останется. Например, пусть на странице находится элемент `div` с идентификатором `messageBox`. Средствами языка JavaScript в этот элемент можно динамически добавлять содержимое и отображать сообщения, взаимодействуя с посетителями. Элемент `div` можно заполнить множеством заголовков, изображений и абзацев текста, чтобы предоставить сообщения о статусе. В какой-то момент (к примеру, если ничего не нужно сообщать) вы можете очистить рамку, но не удалять ее; так вы сохраните возможность отображать сообщения в дальнейшем. Удаление всех элементов внутри рамки выполняется так:

  ```
  $('#messageBox').empty();
  ```

  Как и `.unwrap()`, функция `.empty()` не принимает аргументов.

ПРИМЕЧАНИЕ

В библиотеке jQuery есть еще немало функций для работы с HTML-кодом. Информацию о них можно найти на сайте: **api.jquery.com/category/manipulation/.**

Глава 16

СОВЕРШЕНСТВУЕМСЯ В ПРОГРАММИРОВАНИИ НА ЯЗЫКЕ JAVASCRIPT

Данная глава описывает различные концепции, которые помогут вам совершенствоваться в программировании на языке JavaScript. Вам не понадобятся все описанные здесь идеи для написания функционирующих JavaScript-программ, так что не волнуйтесь, если вы не понимаете их все. Первые несколько разделов содержат полезные советы и методы работы со строками, числами и датами. Разобравшись в основах, вы сможете обрабатывать данные, введенные пользователями в формы, работать с HTML-кодом и HTML-атрибутами, а также генерировать данные для календаря. Раздел «Выводы» данной главы содержит несколько полезных советов для начинающих программистов, однако информация, содержащаяся во всех остальных разделах данной главы, может вам никогда не пригодиться. Тем не менее если вы хотите расширить свои познания, эта глава поможет вам в этом.

Работа со строками

Строки являются наиболее часто встречающимся типом данных: информация, введенная в элемент формы, путь к изображению, URL-адрес и HTML-код, который вы хотите заменить, — все это примеры букв, символов и чисел, объединенных в строки. Вы получили базовые знания о строках в главе 2, однако язык JavaScript предусматривает множество полезных способов манипулирования и работы со строками.

Определение длины строки

Иногда вам необходимо узнать количество символов в строке. Например, когда посетитель создает учетную запись на вашем сайте, вы хотите удостовериться, что новый пароль содержит от 6 до 15 символов. Строки имеют свойство `length`, которое предоставляет вам эту ин-

формацию. Чтобы узнать количество символов в строке, добавьте после имени переменной точку и слово `length`: `name.length`.

Допустим, у вас есть форма с текстовым полем, которое имеет идентификатор `password`. Чтобы гарантировать то, что пароль будет содержать нужное количество знаков, вы можете использовать управляющую инструкцию (см. раздел «Управляющие инструкции» главы 3) для проверки длины пароля:

```
var password = $('#password').val();
if (password.length <= 6) {
alert('Этот пароль слишком короткий.');
} else if (password.length > 15) {
alert('Этот пароль слишком длинный.');
}
```

Код в данном примере использует библиотеку jQuery для получения значения поля ввода пароля — `$('#password').val()`, однако остальная часть кода представляет собой простой JavaScript. Отличным способом использования этого фрагмента является его помещение в собственную функцию:

```
function verifyPassword() {
var password = $('#password').val();
if (password.length <= 6) {
alert('Этот пароль слишком короткий.');
} else if (password.length > 15) {
alert('Этот пароль слишком длинный.');
}
}
```

После этого вы можете вызвать ее как часть обработчика события `submit` (см. раздел «События формы» главы 8), чтобы проверить, ввел ли посетитель достаточно длинный пароль при передаче формы:

```
$('form').submit(verifyPassword);
```

Вы также можете поместить этот фрагмент кода внутри события `blur()` (см. раздел «События формы» главы 8), присвоенного полю ввода пароля, поэтому, когда посетитель щелкает вне поля ввода паро-

ля или покидает его, нажав клавишу **Tab**, вы можете проверить, имеет ли пароль правильную длину. Например, если поле ввода пароля имеет идентификатор `password`, то вы можете отправить функцию `verifyPassword` обработчику события `blur`:

```
$('#password').blur(verifyPassword);
```

Изменение регистра строки

Язык JavaScript предусматривает два способа смены регистра букв строки, так что вы можете изменить строки «привет» на «ПРИВЕТ» или «НЕТ» на «нет». Зачем вам может это потребоваться? Приведение строк к одному и тому же регистру облегчает процесс их сравнения. Например, вы создали программу-викторину вроде той, что описана в главе 3 (см. раздел «Создание простой викторины на практике»), и одним из вопросов является: «Как звали первого человека, побывавшего в космосе?» Для проверки ответа испытуемого вы можете использовать такой код:

```
var correctAnswer = 'Юрий Гагарин';
var response = prompt(Как звали первого человека, ↵
побывавшего в космосе?', '');
if (response == correctAnswer) {
// правильно
} else {
// неправильно
}
```

Правильный ответ — 'Юрий Гагарин', но что если испытуемый ввел строку '*юрий гагарин*'? Управляющая инструкция будет выглядеть так: `'юрий гагарин' == 'Юрий Гагарин'`. Так как язык JavaScript воспринимает буквы верхнего и нижнего регистра по-разному, строчные символы 'ю' и 'г' в строке 'юрий гагарин' не будут соответствовать прописным 'Ю' и 'Г' в строке 'Юрий Гагарин', поэтому получится, что испытуемый дал неправильный ответ. То же самое произошло бы в том случае, если бы при вводе ответа была активирована клавиша **Caps Lock**: 'ЮРИЙ ГАГАРИН'.

Чтобы решить эту проблему, вы можете привести строки к одному регистру, а затем сравнить их:

```
if (response.toUpperCase() == correctAnswer ↵
.toUpperCase()) {
// правильно
} else {
// неправильно
}
```

В этом случае управляющая инструкция изменит регистр строки ответа испытуемого и строки правильного ответа на верхний, так что строки `'юрий гагарин'` и `'Юрий Гагарин'` изменятся на `'ЮРИЙ ГАГАРИН'`.

Чтобы изменить регистр букв на нижний, используйте функцию `toLowerCase()`:

```
var answer = 'Юрий Гагарин';
alert(answer.toLowerCase()); // 'юрий гагарин'
```

Обратите внимание, что ни один из этих методов не изменяет исходную строку, содержащуюся в переменной — они просто возвращают эту строку, написанную либо строчными, либо прописными буквами. Поэтому в вышеприведенном примере переменная `answer` по-прежнему содержит строку `'Юрий Гагарин'`, даже после появления окна оповещения. (Другими словами, данные методы работают подобно функции, которая возвращает некоторое другое значение, как описано в разделе «Возвращение информации от функций» главы 3.)

Поиск в строке: техника `indexOf()`

Язык JavaScript предусматривает несколько техник поиска слова, числа или другой серии символов внутри строки. Поиск может пригодиться в случае, если вы хотите узнать, какой браузер использует посетитель для просмотра вашего веб-сайта. Каждый веб-браузер идентифицирует себя с помощью строки, содержащей множество различных статистик. Вы можете увидеть эту строку, добавив следующий JavaScript-код на страницу и просмотреть ее в браузере:

```
<script>
alert(navigator.userAgent);
</script>
```

Navigator — это один из объектов браузера, а userAgent — это свойство данного объекта. Свойство userAgent содержит длинную строку данных, например, для браузера Internet Explorer 10, работающего в ОС Windows 8 свойство userAgent имеет такое значение: Mozilla/5.0 (compatible; MSInternet Explorer 10.0; Windows NT 6.2; Trident/6.0). Поэтому, если вы хотите проверить, является ли браузер программой Internet Explorer 10, вы можете просто поискать в строке userAgent фрагмент MSInternet Explorer 10.

Одним из способов поиска в строке является функция indexOf(). Чтобы воспользоваться этим способом, необходимо после строки добавить точку, функцию indexOf() и искомую последовательность символов. Базовая структура такова:

```
string.indexOf('искомая строка')
```

Функция indexOf() возвращает число: если искомая строка не найдена, то функция возвращает значение –1. Так что если вам нужно проверить, был ли браузер программой Internet Explorer, то используйте такой код:

```
var browser = navigator.userAgent; // это строка
if (browser.indexOf('MSInternet Explorer') != -1) {
// это браузер Internet Explorer
}
```

В данном случае, если функция indexOf() не находит строку 'MSInternet Explorer' в свойстве userAgent, то она возвратит значение –1, поэтому условие проверяет не равен ли результат –1 (!=).

Когда функция indexOf() *находит* искомую строку, она возвращает число, равное положению первого символа искомой строки. Следующий пример прояснит эту концепцию:

```
var quote = 'To be, or not to be.'
var searchPosition = quote.indexOf('To be');
// возвращает 0
```

Здесь функция indexOf() ищет положение фрагмента 'To be' внутри строки 'To be, or not to be.'. Строка начинается с части 'To be', так что функция indexOf() находит искомую строку на первой позиции. Однако в странном мире программирования первой позицией является 0, вторая буква (о) находится на позиции 1, а третья буква (в данном случае пробел) — на позиции 2 (как отмечалось в раз-

деле «Доступ к элементам массива» главы 2, элементы массива индексируются таким же образом).

Функция `indexOf()` начинает поиск с начала строки. Вы также можете начать поиск с конца строки, используя функцию `lastIndexOf()`. Например, в цитате Шекспира слово `'be'` появляется дважды, так что вы можете найти первое слово `'be'` с помощью функции `indexOf()`, а второе — с помощью функции `lastIndexOf()`:

```
var quote = "To be, or not to be."
var firstPosition = quote.indexOf('be'); // возвращает 3
var lastPosition = quote.lastIndexOf('be');
// возвращает 17
```

Результаты применения этих двух функций изображены на рис. 16.1. В обоих случаях, если бы в строке отсутствовало слово `'be'`, то результатом было бы значение –1, а если бы оно встречалось только один раз, то результаты применения функций `indexOf()` и `lastIndexOf()` оказались бы одинаковыми и представляли собой индекс первого символа искомого фрагмента внутри строки.

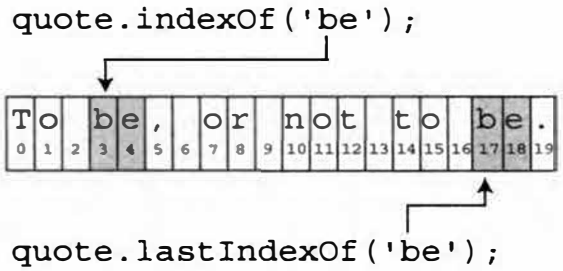

Рис. 16.1. Функции `indexOf()` и `lastIndexOf()` осуществляют поиск конкретного фрагмента внутри строки. Если искомый фрагмент найден, то возвращается индекс его первого символа

Извлечение части строки с помощью метода `slice()`

Чтобы извлечь часть строки, используйте метод `slice()`. Данная функция возвращает часть строки. Например, у вас есть строка `http://www.sawmac.com` и вы хотите избавиться от части `http://`. Одним из способов это сделать является извлечение каждого символа строки, находящегося после части `http://`:

```
var url = 'http://www.sawmac.com';
var domain = url.slice(7); // www.sawmac.com
```

Функции `slice()` необходимо предоставить число, определяющее *индекс* первого символа извлекаемой строки (рис. 16.2). В данном примере (`url.slice(7)`) число 7 указывает на восьмую букву строки (помните, что первая буква находится на позиции 0). Функция возвращает все символы, начиная с указанного индекса до конца строки.

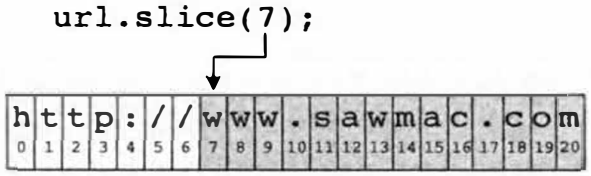

Рис. 16.2. Если вы не передадите функции `slice()` второй аргумент, то она просто извлечет все символы, начиная с указанного индекса (в данном примере 7) до конца строки

Вы также можете извлечь определенное число символов строки, передав функции `slice()` второй аргумент. Вот базовая структура функции `slice()`:

```
string.slice(начало, конец);
```

Аргумент *начало* — это число, указывающее на первый символ извлекаемой строки. Со значением *конец* немного сложнее — это не индекс последнего символа извлекаемой строки, а индекс последнего символа + 1. Например, если бы вы хотели извлечь первые пять символов строки *'To be, or not to be.'*, то вы бы указали 0 в качестве первого аргумента и 5 — в качестве второго. Как видно на рис. 16.3, 0 — это первая буква строки, а 5 — это шестая буква, однако последняя указанная буква не извлекается из строки. Другими словами, символ, на который указывает второй аргумент, *никогда* не является частью извлекаемой строки.

▶ СОВЕТ

Если вы хотите извлечь конкретное число символов строки, просто добавьте это число в качестве второго значения. Например, если вы хотите извлечь первые 10 букв строки, первым аргументом будет 0 (первая буква), а вторым — 0 + 10, или просто 10: `slice(0,10)`.

Вы также можете указать отрицательные числа, например, `quote.slice(-6,-1)`. Отрицательное число ведет отсчет от конца строки, как показано на рис. 16.3.

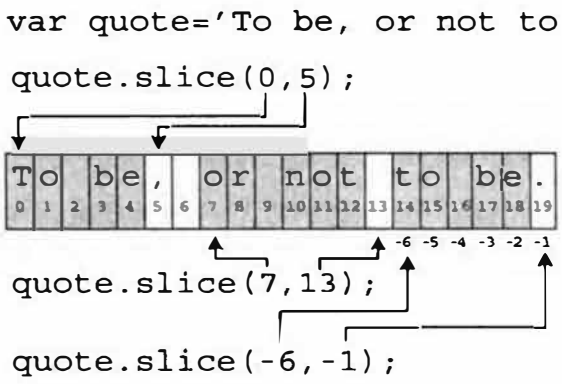

Рис. 16.3. Метод `slice()` извлекает часть строки. Сама строка не меняется. Например, строка, содержащаяся в переменной `quote`, не меняется в результате выполнения кода `quote.slice(0,5)`. Функция просто возвращает извлеченную строку, которую вы можете сохранить в переменной, отобразить в окне оповещения или даже передать в качестве аргумента функции

СОВЕТ

Если вам необходимо извлечь строку, содержащую последние X символов строки, передайте функции `slice()` одно значение — отрицательное число, соответствующее нужному количеству букв. Например, если вы хотите извлечь последние 6 букв строки, вы можете написать следующее:

```
var end_of_string = quote.slice(-6);
```

Поиск по маске в строках

Иногда вам необходимо найти в строке не конкретное значение, а определенное сочетание символов. Например, вам нужно удостовериться в том, что при заполнении формы заказа посетитель вводит номер телефона в корректном формате. Вы не ищете конкретный номер телефона вроде 495-333-1212. Вместо этого вы ищете по общей маске: три цифры, дефис, три цифры, еще один дефис и еще четыре цифры. Вы хотите проверить значение, введенное посетителем, и если оно соответствует маске (например, 495-555-3843, 812-333-3782 или 831-234-4828 и т. д.), то все в порядке. Однако если введенное значение не соответствует маске (например, посетитель ввел *823арфпдар*), то вам нужно отобразить сообщение вроде: «Эй, дружище, не пытайся нас надуть!».

Язык JavaScript позволяет вам использовать *регулярные выражения* для поиска масок в строке. Регулярное выражение — это последовательность символов, определяющая искомую маску. Как и в случае с многими другими терминами программирования, название «регулярное

выражение» может ввести в заблуждение. Например, вот, как выглядит регулярное выражение:

```
/^[-\w.]+@([a-zA-Z0-9][-a-zA-Z0-9]+\.)+[a-zA-Z]{2,4}$/
```

Данное выражение не выглядит таким уж *регулярным*, если только вы не являетесь пришельцем с планеты «Омикрон 9». Для создания маски вы используете такие символы, как `*`, `+`, `?` и `\w`, которые заменяются интерпретатором JavaScript на соответствующие символы строки: буквы, цифры и т. д.

ПРИМЕЧАНИЕ

Профессионалы часто используют слово *«регекс»* вместо термина *«регулярное выражение»* (слово «regex» образовано от англ. «regular expression»).

Создание и использование регулярного выражения

Чтобы создать регулярное выражение на языке JavaScript, вы должны создать объект регулярного выражения, который представляет собой последовательность символов, заключенных между двумя слешами. Например, для создания регулярного выражения, которое соответствует слову «привет», вы можете ввести следующее:

```
var myMatch = /привет/;
```

Как открывающая и закрывающая кавычки создают строку, так открывающий / и закрывающий / создают регулярное выражение.

Существует несколько функций, использующих регулярные выражения (вы узнаете о них далее в главе), самой простой из которых является функция `search()`. Она работает подобно функции `indexOf()`, но вместо того, чтобы искать одну строку внутри другой, она ищет в строке маску (регулярное выражение). Например, вы хотите найти строку *'To be'* в строке *'To be or not to be'*. Вы видели, как это можно сделать, используя метод `indexOf()`, а вот как можно этого достичь, используя регулярное выражение:

```
var myRegEx = /To be/; // регулярное выражение не ↵
заключено в кавычки
var quote = 'To be or not to be.';
var foundPosition = quote.search(myRegEx); //
возвращает 0
```

Если функция `search()` находит совпадение, то возвращает индекс первой буквы, а если не находит — возвращает значение –1. Поэтому в вышеприведенном примере переменная `foundPosition` содержит значение 0, так как строка *'To be'* находится в самом начале строки.

Как говорилось в начале главы, функция `indexOf()` работает точно так же. Вы, вероятно, спрашиваете себя: «Зачем изучать регулярные выражения, если обе функции работают одинаково?» Преимуществом регулярных выражений является то, что они могут находить маски в роке, так что они могут делать более сложные сравнения, чем функция `indexOf()`, которая всегда ищет точное совпадение с конкретной строкой. Например, вы могли бы использовать метод `indexOf()` для проверки того, содержит ли строка веб-адрес *http://www.eksmo.ru/*, но вам пришлось бы использовать регулярные выражения для поиска любого текста, соответствующего формату URL-адреса — это то, что вам нужно для проверки, ввел ли посетитель веб-адрес при размещении комментария в вашем блоге.

Однако чтобы освоить регулярные выражения, вам нужно изучить часто вводящие в заблуждение символы, необходимые для построения регулярных выражений.

Построение регулярного выражения

Хотя регулярное выражение может состоять из слова или слов, чаще всего вы будете использовать комбинацию букв и специальных символов для определения маски, соответствия которому вы собираетесь искать. Регулярные выражения предусматривают разные символы для обозначения разных типов знаков. Например, точка (`.`) представляет любой символ, а `\w` соответствует любой букве или цифре (за исключением пробелов и таких символов, как `$` или `%`). Табл. 16.1 предоставляет список наиболее часто используемых символов.

ПРИМЕЧАНИЕ

Если от всех этих разговоров о «регулярных» выражениях у вас разболелась голова, то вы будете рады узнать, что эта книга предоставляет некоторые полезные регулярные выражения (см. раздел «Полезные регулярные выражения» далее в главе), которые вы можете скопировать и использовать в своих сценариях, не вдаваясь в подробности их работы.

Табл. 16.1. Символы, часто используемые в регулярных выражениях

Символ	Соответствует
.	Любому символу: букве, цифре, пробелу и др
\w	Любому символу, входящему в состав слова, включая буквы a–z, A–Z, цифры 0–9 и символ нижнего подчеркивания: _
\W	Любому символу, не входящему в состав слова. Это полная противоположность символу \w
\d	Любой цифре от 0 до 9
\D	Любому символу, кроме цифры. Это полная противоположность символу \d
\s	Пробелу, табуляции, возврату каретки или новой строке
\S	Всем символам, кроме пробела, табуляции, возврата каретки или новой строки
^	Началу строки. Полезен, когда вам нужно убедиться, что перед строкой не находится ни один символ
$	Концу строки. Используйте знак $, чтобы удостовериться, что искомые символы находятся в конце строки. Например, /com$/ соответствует строке com, но только если эти буквы являются последними тремя буквами строки. Другими словами, /com$/ соответствует com в строке Infocom, но не в строке communication
\b	Пробелу, началу строки, концу строки или любому другому небуквенному и нечисловому знаку: +, = или '. Используйте символ \b для сопоставления начала или конца слова, даже если это слово находится в начале или конце строки
[ ]	Любому символу в квадратных скобках. Например, конструкция [aeiou] соответствует любой из этих букв в строке. Для диапазона символов используйте дефис: [a-z] соответствует любой строчной букве; [0-9] соответствует любой цифре (то же, что и \d)
[^]	Любому символу, кроме тех, которые находятся в квадратных скобках. Например, [^aeiouAEIOU] соответствует любому символу, кроме гласных букв. [^0-9] соответствует любому символу, который не является цифрой (то же, что и \D)
|	Символу перед или после знака |. Например, a|b соответствует либо a, либо b, но не обоим. (см подраздел «Веб-адрес» далее в главе, чтобы увидеть, как работает этот символ)
\	Используется для поиска буквального значения символов, используемых при построении регулярных выражений (*, ., \, / и т. д.). Например, точка (.) в регулярном выражении означает «любой символ», но если вы хотите найти точку в строке, вам нужно указать на это так: \.

Регулярные выражения — это тема, которую лучше всего объяснять с помощью примеров, так что оставшаяся часть этого раздела содержит несколько примеров регулярных выражений. Допустим, вы хотите найти шесть цифр, идущих подряд, возможно, чтобы проверить наличие в строке почтового индекса:

1. **Найдите одну цифру.**

   Первый шаг — это найти соответствие одной цифре. Если вы обратитесь к табл. 16.1, то увидите, что существует специальный символ `\d`, который соответствует любой цифре.

2. **Найдите шесть цифр, идущих подряд.**

   Так как символ `\d` соответствует одной цифре, легко найти шесть цифр, используя такое регулярное выражение: `\d\d\d\d\d\d`. (в разделе «Группировка частей маски» далее в главе предложен более компактный способ записи этого выражения).

3. **Найдите только шесть цифр.**

   Регулярное выражение подобно высокоточной управляемой ракете: она нацеливается на первую часть строки. Поэтому иногда вы получаете соответствие части слова или набору символов. Данное регулярное выражение находит первые шесть цифр, идущих подряд, которые ему попадаются. Например, оно найдет `123456` в числе `12345678998`. Очевидно, `12345678998` — это не почтовый индекс, поэтому вам нужно регулярное выражение, которое нацеливается только на шестизначные номера.

   Символ `\b` (так называемый символ *границы слова*) соответствует любому небуквенному или нечисловому знаку, так что вы можете переписать ваше регулярное выражение следующим образом: `\b\d\d\d\d\d\d\b`. Вы также можете использовать знак `^` для сопоставления начала строки и символ `$` — для сопоставления ее окончания. Этот способ пригодится, если вы хотите, чтобы вашему регулярному выражению соответствовала вся строка. Например, если посетитель ввел «выпфпфп 88888 лфопдорп» в поле почтового индекса формы заказа, то вы можете попросить этого посетителя прояснить (исправить) введенный индекс перед отправкой формы. В конце концов, вы ожидаете получить что-то вроде `604267` (без каких-либо других символов). В данном случае, регулярное выражение будет иметь такой вид: `^\d\d\d\d\d\d$`.

ПРИМЕЧАНИЕ

В зависимости от страны индексы могут содержать разное количество цифр. Например, формат ZIP + 4 включает дефис и четыре дополнительные цифры после первых пяти: 97213-1234. Чтобы регулярное выражение учло такую возможность, обратитесь к подразделу «Почтовый индекс» далее в главе.

4. Задействуйте свое регулярное выражение в коде JavaScript.

Допустим, введенные пользователем данные уже содержатся в переменной *zip*, и вы хотите проверить, соответствуют ли эти данные формату почтового индекса:

```
var zipTest = /^\d\d\d\d\d\d$/; //создание ↵
регулярного выражения
if (zip.search(zipTest) == -1) {
alert('Это некорректный почтовый индекс');
} else {
// корректный формат
}
```

Данный пример регулярного выражения работает, однако шестикратный ввод символов `\d` является не очень удобным. Что если вам нужно сопоставить 100 цифр, идущих подряд? К счастью, язык JavaScript включает несколько символов для сопоставления множества одинаковых знаков. Они перечислены в табл. 16.2. Вы помещаете символ сразу *после* знака, который вы сопоставляете.

Табл. 16.2. Символы, используемые для сопоставления нескольких экземпляров одного и того же знака или маски

Символ	Соответствует
`?`	Нулю или одному экземпляру предыдущего элемента, то есть наличие предыдущего элемента необязательно, однако если он присутствует, то может появиться только однажды. Например, регулярное выражение `colou?r` соответствует и `color`, и `colour`, но не `colouur`
`+`	Одному или нескольким экземплярам предыдущего элемента. Предыдущий элемент должен появиться как минимум один раз
`*`	Любому количеству экземпляров предыдущего элемента. Наличие предыдущего элемента необязательно, он может присутствовать в любом количестве. Например, `.*` соответствует любому количеству знаков
`{n}`	Конкретному количеству экземпляров предыдущего элемента. Например, `\d{3}` соответствует трем цифрам, идущим подряд
`{n, }`	Предыдущему элементу, присутствующему *n* или более раз. Например, `a{2,}` соответствует букве `a`, присутствующей в двух или более экземплярах, то есть будет соответствовать строке `aa` в слове `aardvark` и строке `aaaa` в слове `aaaahhhh`
`{n,m}`	Предыдущему элементу, присутствующему не менее *n* и не более *m* раз. Поэтому `\d{3,4}` будет соответствовать трем или четырем цифрам подряд (но не двум и не пяти цифрам подряд)

Например, чтобы сравнить пять цифр, вы можете написать: `\d{5}`. Знак `\d` соответствует одной цифре, затем часть `{5}` сообщает интерпретатору JavaScript сопоставить пять цифр. Таким образом, код `\d{100}` соответствует 100 цифрам, идущим подряд.

Рассмотрим еще один пример. Допустим, вы хотите найти в строке имя какого-либо файла в формате GIF. Кроме того, вы хотите извлечь имя файла и каким-то образом его использовать в своем сценарии (например, вы можете использовать функцию `match()`, описанную в разделе «Сопоставление маски» далее в главе). Иначе говоря, вам нужно найти любую строку, соответствующую базовой маске имени файла в формате GIF, например, *logo.gif*, *banner.gif* или *ad.gif*.

1. **Определите общую маску имен.**

   Чтобы построить регулярное выражение, вам сначала необходимо определить маску, с которой вы станете сравнивать строки. В данном примере, поскольку вы ищете файлы в формате GIF, вы знаете, что все имена будут заканчиваться частью *.gif*. Другими словами, перед фрагментом *.gif* может находиться любое количество букв, цифр или других символов.

2. **Найдите часть *.gif*.**

   Так как вы ищете строку `.gif`, вы можете подумать, что данная часть регулярного выражения будет записана просто как `.gif`. Однако если вы посмотрите в табл. 16.1, то увидите, что точка имеет особое значение и «соответствует любому знаку». Поэтому часть *.gif* будет соответствовать как .gif, так и tgif. Точка соответствует любому символу, поэтому кроме того, что она соответствует точке, она соответствует еще и букве *t* в строке *tgif*. Чтобы создать регулярное выражение с точкой, которая бы воспринималась буквально, добавьте перед ней слеш (косую черту). Таким образом, сочетание символов `\.` означает «найди символ точки», поэтому регулярное выражение для нахождения строки *.gif* будет таким: `\.gif`.

3. **Найдите любое количество знаков перед частью .gif.**

   Вам нужно найти соответствие чему-то вроде *a.gif*, *photo.gif*, а не просто *.gif* (поскольку это не является корректным GIF-файлом). Чтобы найти любое количество знаков, вы можете использовать символы `.+`, означающие: «найди как минимум один (+) экземпляр символа (.)». Однако если вы используете такое регулярное выражение `.+\.gif`, то можете получить не только имя файла: данное

регулярное выражение соответствует всем буквам любой строки. Например, если у вас есть строка `'file logo.gif '`, то регулярное выражение `.+\.gif` будет соответствовать всей строке, хотя вам нужно получить только фрагмент *logo.gif*. Чтобы этого добиться, используйте символы `\S`, соответствующие любому знаку, кроме пробела (табуляции, возврата каретки или новых строк): регулярное выражение `\S+\.gif` соответствует только имени *logo.gif* в строке.

4. **Убедитесь в том, что после части `gif` ничего нет.**

   Теперь регулярное выражение находит только имя файла... или нет? С регулярными выражениями могут возникнуть сложности. Например, если вы примените созданное ранее регулярное выражение к строке `e-mail: alex.gifford@example.com`, то получите результат `alex.gif`, что вообще не является именем файла. Чтобы предотвратить это, вам следует удостовериться в том, что после части `gif` ничего нет. Для этого вы можете добавить границу (`\b`), например, `\S+\.gif\b`, чтобы исключить части других строк, включающих фрагмент `.gif`.

5. **Сделайте поиск нечувствительным к регистру.**

   Есть еще одна проблема с данным регулярным выражением: оно находит только файлы, заканчивающиеся на *.gif*, однако *.GIF* — это тоже корректное расширение, тем не менее данное регулярное выражение не учтет имя файла вроде *logo.GIF*. Чтобы заставить регулярное выражение игнорировать различия между прописными и строчными буквами, используйте аргумент `i`:

   ```
   /\S+\.gif\b/i
   ```

   Обратите внимание на то, что аргумент `i` находится за пределами маски и справа от символа `/`, определяющего окончание маски регулярного выражения.

6. **Задействуйте регулярное выражение в коде.**

   ```
   var testString = 'Файл logo.gif автор alex.gifford@ ↵
   example.com'; // строка для проверки
   var gifRegex = /\S+\.gif\b/i; // регулярное выражение
   var results = testString.match(gifRegex);
   var file = results[0]; // logo.gif
   ```

Этот код находит в строке имя файла. (Вы узнаете о том, как работает функция `match()`, в разделе «Сопоставление маски» далее в главе.)

Группировка частей маски

Вы можете использовать круглые скобки для создания подгруппы в пределах маски. Подгруппы могут пригодиться, когда вы используете любой из символов, перечисленных в табл. 16.2 для сопоставления нескольких экземпляров одной и той же маски.

Например, вы хотите проверить наличие строк `Apr` или `April` — обе начинаются с фрагмента `Apr`, так что станете сопоставлять эту часть, однако вы не можете использовать только строку Apr для сравнения, так как она соответствует и части `Apr` в словах `Apricot` или `Aprimecorp`. Таким образом, вы должны сопоставить строку `Apr`, за которой следует пробел или окончание слова (это символ `\b`, описанный в табл. 16.1) или строку `April`, за которой следует окончание слова. Иначе говоря, строка `il` необязательна. Вот, как вы можете записать это, используя круглые скобки:

```
var sentence = 'April is the cruelest month.';
var aprMatch = /Apr(il)?\b/;
if (sentence.search(aprMatch) != -1) {
// найдена строка Apr или April
} else {
//не найдено
}
```

Регулярное выражение, используемое здесь (`/Apr(il)?\b/`), обусловливает обязательное наличие строки `Apr` и необязательное наличие подмаски (`il`) (знак `?` означает нуль или один раз). Наконец, знак `\b` соответствует окончанию слова, так что вы не получите результат, вроде `Apricot` или `Aprimecorp`. (О другом методе использования подмасок вы можете узнать из врезки «Использование подмасок для замены текста» далее.)

Чтобы предотвратить возникновение проблем с этим регулярным выражением, вы можете добавить границу `\b` в его начале.

```
var aprMatch = /\bApr(il)?\b/;
```

В этом случае результатами сопоставления не станут wApril или SApr. (Конечно, существует весьма небольшая вероятность столкнуться с такими строками, однако лучше перестраховаться, чем потом сожалеть.)

СОВЕТ

Вы можете найти полную библиотеку регулярных выражений по адресу: **www.regexlib.com**. На этом сайте вы найдете регулярные выражения на все случаи жизни.

Полезные регулярные выражения

Создание регулярных выражений имеет свои сложности. Вы должны не только понимать принцип работы различных знаков, но и конструировать правильные маски. Например, при поиске соответствия почтовому индексу, вы должны принимать во внимание тот факт, что почтовый индекс может содержать разное количество цифр: и шесть (199034), и пять (97213), и даже 5+4 (97213-3333). Чтобы начать использовать регулярные выражения, рассмотрим несколько примеров.

ПРИМЕЧАНИЕ

Если вам не хочется вводить эти регулярные выражения (кто может вас за это винить?), вы можете обратиться к файлу *пример_регулярное выражение.txt* в папке *глава16*. (См. примечание в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.)

Почтовый индекс

Почтовые индексы варьируются от страны к стране, например, индексы городов России состоят из шести цифр, поэтому регулярное выражение для них является довольно простым:

```
\d{6}
```

Рассмотрим более сложный пример, в США индексы содержат либо пять цифр, либо девять цифр (пять цифр, за которыми следуют дефис и еще четыре цифры). Вот регулярное выражение, соответствующее обоим вариантам:

```
\d{5}(-\d{4})?
```

ПРИМЕЧАНИЕ

Чтобы найти регулярные выражения, соответствующие почтовым индексам других стран, посетите сайт: **regexlib.com/Search.aspx?k=postal+code**.

Данное регулярное выражение можно разбить на следующие небольшие фрагменты:

- Часть `\d{5}` ищет соответствие пяти цифрам, например, 97213.
- Круглые скобки `()` создают подмаску. Содержимое скобок считается единой маской, соответствие которой необходимо найти. Сейчас вы узнаете, почему это так важно.
- Часть `-\d{4}` соответствует дефису и следующим за ним четырем цифрам: –1234.
- Знак `?` соответствует нулю или одному экземпляру предыдущей маски. Вот, где вступают в игру круглые скобки: выражение `(-\d{4})` воспринимается как единое целое, так что символ `?` соответствует нулю или одному экземпляру конструкции, состоящей из дефиса и четырех следующих за ним цифр. Так как вы не обязаны включать дефис + четыре цифры, данная маска может отсутствовать. Другими словами, если вы тестируете значение вроде 97213, вы получите совпадение, так как наличие фрагмента, состоящего из дефиса и четырех цифр, не является обязательным.

ПРИМЕЧАНИЕ

Чтобы убедиться в том, что регулярному выражению соответствует вся строка, начните регулярное выражение с символа `^` и завершите символом `$`. Например, если вы хотите убедиться, что посетитель ввел в поле формы только почтовый индекс в корректном формате, используйте регулярное выражение: `^\d{5}(-\d{4})?$`, чтобы предотвратить получение ответа типа «бла 97213 бла бла».

Телефонный номер

Рассмотрим формат телефонного номера, состоящего из трехзначного кода города, за которым следуют семь цифр. Люди могут записать такой номер разными способами: 812-333-1212, (812) 333-1212, 812.333.1212 или просто 812 333 1212. Регулярное выражение для данной маски имеет следующий вид:

```
\(?(\d{3})\)?[ .-](\d{3})[ .-](\d{4})
```

ПРИМЕЧАНИЕ

Чтобы найти регулярные выражения, соответствующие телефонным номерам разных стран, посетите сайт: **regexlib.com/Search.aspx?k=phone+number**.

Данное регулярное выражение кажется сложным, однако если вы разобьете его на части (и переведете их, как показано далее), то разобраться в нем будет нетрудно:

- `\(` соответствует буквальному смыслу открывающей круглой скобки. Так как круглые скобки используются для группировки масок (см. пример с почтовым индексом), то открывающая круглая скобка имеет особый смысл в регулярных выражениях. Чтобы сообщить интерпретатору JavaScript о том, что вы ищете соответствие открывающей круглой скобке, вам нужно использовать косую черту (как в случае с кавычками (см. врезку «Кавычки в строках» главы 2)).
- Символ `?` означает, что наличие знака `(` не является обязательным, так что маска будет также соответствовать телефонному номеру без скобок, например, 812-333-1212.
- Часть `(\d{3})` — это подмаска, соответствующая любым трем цифрам.
- Фрагмент `\)?` говорит о том, что наличие закрывающей круглой скобки не является обязательным.
- Конструкция `[ .-]` будет соответствовать либо пробелу, либо дефису, либо точке. (Имейте в виду то, что обычно вам нужно использовать слеш (`\.`), чтобы сообщить интерпретатору JavaScript о том, что вы ищете соответствие именно точке, а не *любому* символу. Однако внутри квадратных скобок точка всегда воспринимается буквально).
- Часть `(\d{3})` — это еще одна подмаска, соответствующая любым трем цифрам.
- Фрагмент `[ .-]` соответствует либо пробелу, либо дефису, либо точке.
- Конструкция `(\d{4})` представляет собой последнюю подмаску, которая соответствует любым четырем цифрам.

Адрес электронной почты

Проверка правильности электронного адреса, введенного в поле формы, является распространенной задачей при обработке пользовательского ввода. Многие люди пытаются уклониться от предоставления настоящего электронного адреса, используя ответы типа «не ваше дело» или просто вводят адрес с ошибками (например, *missing@sawmac.*

commm). Следующее регулярное выражение может проверить, содержит ли строка электронный адрес в правильном формате:

```
[-\w.]+@([A-z0-9][-A-z0-9]+\.)+[A-z]{2,4}
```

ПРИМЕЧАНИЕ

Данное регулярное выражение не выясняет, является ли введенный адрес действующим, а просто проверяет его формат.

Данное регулярное выражение состоит из следующих частей:

- Часть `[-\w.]+` соответствует дефису, любому слову или точке, присутствующим один или более раз. Таким образом, она будет соответствовать строкам «ivan», «ivan.ivanov» или «ivan-ivanov».
- Знак `@` — это символ @, присутствующий в адресе электронной почты: *missing@sawmac.com*.
- Фрагмент `[A-z0-9]` соответствует одной букве или цифре.
- Конструкция `[-A-z0-9]+` соответствует одному или нескольким экземплярам буквы, цифры или дефиса.
- Сочетание символов `\.` соответствует точке, например, как в *sawmac.com*.
- Символ `+` соответствует одному или нескольким экземплярам маски, включающего три вышеприведенных совпадения. Данный знак позволяет учесть имена поддоменов, например, *bob@mail.sawmac.com*.
- Конструкция `[A-z]{2,4}` — любая буква, присутствующая 2, 3 или 4 раза. Она соответствует строке *com* в *.com* или *ru* в *.ru*.

ПРИМЕЧАНИЕ

Приведенное регулярное выражение для электронного адреса не соответствует *всем* форматам адресов электронной почты. Так `!#$%&'*+-/=?^_`.{|}~@example.com` является допустимым электронным адресом, однако описанное в данном разделе регулярное выражение не будет ему соответствовать. Оно сконструировано для поиска наиболее вероятных адресов. Если вы хотите большей точности, используйте следующее регулярное выражение:

```
/^[\w!#$%&\'*+\/=?^`{|}~.-]+@(?:[a-z\d][a-z\d-]*(?:\. ↵
[a-z\d][a-z\d-]*)?)+\.(?:[a-z][a-z\d-]+)$/i
```

Дата

Дату можно записать разными способами, например, 28/09/2014, 28.09.2014, 28 09 2014 или даже 28-9-2014. Поскольку ваши посетители могут ввести дату как угодно, вам нужен способ, позволяющий проверить, ввели ли они дату в допустимом формате. (Из текста в рамке «Объект `Date, работающий за кадром`» далее в главе вы узнаете, как преобразовать введенные данные в единый стандартный формат, чтобы удостовериться в том, что все даты отформатированы должным образом).

Ниже представлено регулярное выражение, проверяющее формат введенной даты:

```
([0123]?\d)[-\/ .]([01]?\d)[-\/ .](\d{4})
```

- Символы `()` объединяют следующие две маски в группу. Вместе они формируют число.
- Часть `[0123]?` Соответствует цифрам 0, 1, 2 или 3, встречающимся нуль или более раз. Поскольку 40-го дня месяца не бывает, вы ограничиваетесь этими четырьмя цифрами. Этот маска не является обязательной (на что указывает знак `?`), так как посетитель может ввести 9 вместо 09 для указания девятого дня месяца.
- Фрагмент `\d` соответствует любой цифре.
- Конструкция `[-\/ .]` будет соответствовать дефису, слешу, точке или пробелу. Существует несколько приемлемых разделителей, например, 21/10, 21 10, 21.10 или 21-10.
- Символы `()` определяют следующую подмаску, соответствующую-месяцу.
- Часть `[01]?` соответствует либо 0, либо 1, а знак `?` делает наличие данной цифры необязательным. Это первая цифра месяца. Очевидно, что она не может быть больше 1 — 22-го месяца не существует. Кроме того, если указанный месяц находится в диапазоне от января до сентября, то вы можете получить, например, 5 вместо 05. Вот, почему наличие этой цифры необязательно.
- Фрагмент `\d` соответствует любой цифре.
- Конструкция `[-\/ .]` описана выше.
- Символы `()` описывают год.
- Часть `\d{4}` соответствует любым четырем цифрам, например, 1908 или 2880.

ПРИМЕЧАНИЕ

Несмотря на то, что вы можете самостоятельно обеспечить проверку формы, используя некоторые из описанных здесь масок регулярных выражений, это не является единственным вариантом. Кое-кто уже решил эту проблему, так почему бы не воспользоваться этим? Более простой способ удостовериться в том, что посетители вашего сайта ввели корректную информацию в форму, заключается в применении плагина jQuery Form Validation, описанного в разделе «Проверка формы» главы 8.

Веб-адрес

Проверка соответствия строки формату веб-адреса используется, если вы просите посетителя ввести адрес его веб-сайта и хотите удостовериться, что он его ввел, или если вы хотите просмотреть некий текст и найти в нем все URL-адреса. Базовым регулярным выражением для URL-адреса является код:

```
((\bhttps?:\/\/)|(\bwww\.))\S*
```

Данное выражение сложно для восприятия из-за использования множества скобок для группировки различных частей. Рисунок 16.4 поможет вам разобраться в этом регулярном выражении. Один набор круглых скобок (1) окружает две другие группы, заключенные в скобки (2 и 3). Знак | между двумя группами означает «или». Иначе говоря, регулярное выражение должно соответствовать либо части 2, либо части 3.

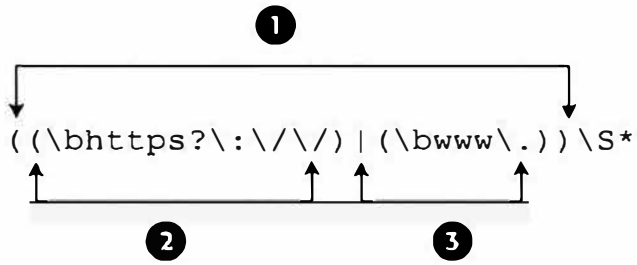

Рис. 16.4. Вы можете группировать выражения, используя круглые скобки и искать любое из двух выражений, применяя знак |. Например, внешнее выражение (1) будет соответствовать любому тексту, который соответствует либо части 2, либо части 3

- Символ (— это начало внешней группы (1 на рис. 16.4).
- Символ (— это начало внутренней группы (2 на рис. 16.4).
- Часть \b соответствует началу слова.

- Фрагмент `http` соответствует началу полного веб-адреса, начинающегося с *http*.
- Сочетание `s?` означает, что присутствие буквы *s* необязательно. Поскольку веб-страница может быть передана через защищенное соединение, веб-адрес может начинаться с части *https*.
- `:\/\/` соответствует символам `://`. Так как слеш в регулярных выражениях имеет особое значение, вы должны поставить перед ним обратный слеш, чтобы он воспринимался буквально.
- Символ `)` — это конец внутренней группы (2 на рис. 16.4). Эта группа будет соответствовать либо *http://* либо *https://*.
- Знак `|` означает одну из двух групп (2 или 3 на рис. 16.4).
- Символ `(` — это начало второй внутренней группы (3 на рис. 16.4).
- Часть `\b` соответствует началу слова.
- `www\.` соответствует части *www*.
- Символ `)` — это конец второй внутренней группы (3 на рис. 16.4). Эта группа будет соответствовать URL-адресу, в котором нет части *http://*, начинающемуся с части *www*.
- Символ `)` — это конец внешней группы (1 на рис. 16.4). На данном этапе регулярное выражение будет соответствовать тексту, начинающемуся с *http://*, *https://* или *www*.
- Сочетание `\S*` соответствует нулю или более знакам, не являющимся пробелами.

Данное выражение не является совершенным (например, оно будет соответствовать бессмысленному адресу *http://#$*%&*@**), однако оно достаточно простое и станет успешно находить соответствие настоящим URL-адресам, например, такому: *http://www.sawmac.com/missing/js/index.html*.

СОВЕТ

Чтобы проверить, содержит ли строка только URL-адрес (то есть перед и после URL-адреса ничего нет), используйте символы `^` и `$` в начале и в конце регулярного выражения, а также удалите символы `\b`: `^((https?:\/\/)|(www\.))\S*$`.

Сопоставление маски

Метод `search()`, описанный в разделе «Создание и использование регулярного выражения» данной главы, — это один из способов проверить строку на наличие конкретной маски. Другим способом является функция `match()`. Вы можете применить эту функцию к строке не только для того, чтобы проверить наличие в ней маски, но также и для того, чтобы выделить эту маску для дальнейшего использования в сценарии. Например, в вашей форме присутствует текстовое поле, предназначенное для ввода комментариев посетителей. Возможно, вы хотите проверить, содержат ли комментарии URL-адрес, и если да, то извлечь этот URL-адрес для дальнейшей обработки.

Следующий код находит и извлекает URL-адрес, используя функцию `match()`:

```
// строка для проверки введенного URL-адреса
var text='мой веб-сайт: www.eksmo.ru';
// создать регулярное выражение
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/
// найти в строке соответствие регулярному выражению
var url = text.match(urlRegex);
alert(url[0]); // www.eksmo.ru
```

Сначала код создает переменную, содержащую строку, которая включает URL-адрес `www.eksmo.ru`. Эта переменная здесь просто в качестве примера (чтобы вы могли увидеть, что делает функция `match()`). Если бы вы хотели действительно проверить содержимое текстового поля формы, вы могли бы использовать такой код:

```
var text = $('#comments').val();
```

Далее код создает регулярное выражение, соответствующее URL-адресу (см. подраздел «Веб-адрес»). Наконец, код применяет к строке функцию `match()`. Функция `match()` является строковым методом, так что вы начинаете с имени переменной, содержащей строку, добавляете точку, а затем функцию `match()`, которой вы передаете регулярное выражение.

В приведенном примере переменная `url` содержит результат выполнения функции `match()`. Если маска регулярного выражения в строке

не найдена, то результатом будет специальное значение языка JavaScript `null`. Если соответствие найдено, то сценарий возвратит массив, первым значением которого окажется соответствующий текст. В данном примере переменная `url` содержит массив, первым элементом которого является найденный текст. В этом случае массив `url[0]` содержит значение `www.eksmo.ru` (см. раздел «Массивы» главы 2 для получения дополнительных сведений о массивах).

ПРИМЕЧАНИЕ

В языке JavaScript значение `null` расценивается как значение `false`, так что вы можете проверить наличие соответствия так:

```
var url = text.match(urlRegex);
if (! url) {
//совпадений нет
} else {
//совпадение найдено
}
```

Метод `match()` также предоставляет некоторые дополнительные сведения. В дополнение к массиву, чьим первым элементом является сопоставляемая строка, метод `match()` также возвращает свойство `index`, которое указывает на позицию в строке, где начинается совпадение. Например:

```
var string = 'To be or not';
ar regex = /be/;
var result = string.match(regex);
alert(result.index); // выводит результат 3
```

В переменной `result` сохраняются значения, возвращаемые методом `match()`. В данном случае есть массив, чей первый и единственный элемент является совпавшей строкой: переменная `result[0]` содержит текст `be`. Метод `match()` также добавляет к результату свойство `index`. В данном примере это свойство имеет значение `3`, так как строка `be` начинается на четвертой позиции в строке (помните о том, что в языке JavaScript отсчет начинается с 0). Это то же значение, которое было бы возвращено методом `search()` (см. раздел «Поиск по шаблону в строках» ранее в главе) и подобно тому, как метод `indexOf()` (см. раздел «Поиск в строке: Техника `indexOf()`» ранее в главе) возвращает

начальную позицию подстроки большей строки. Кроме того, это то же значение, которое вы использовали бы с методом `slice()` (см. раздел «Извлечение части строки с помощью метода `slice()`» ранее в главе), чтобы извлечь подстроку из большей строки.

Метод `match()` также возвращает дополнительные элементы массива, если вы использовали скобки `()` для создания подмасок, как описано в тексте в рамке «Использование подмасок для замены текста» далее в главе. Первый элемент в массиве всегда является совпавшей строкой, но если вы использовали подмаски, то каждый дополнительный элемент массива содержит совпадающее значение в рамках подмаски.

Нахождение всех экземпляров маски

Функция `match()` работает двумя способами в зависимости от того, как вы построили свое регулярное выражение. В вышеприведенном примере метод возвращает массив с первым попавшимся соответствующим текстом и свойство `index`, содержащее начальную позицию совпадающей строки (а также дополнительные элементы массива, если вы использовали подмаски в своем регулярном выражении). Таким образом, если у вас была длинная строка, содержащая несколько URL-адресов, то только первый из них будет найден. Однако вы также можете активировать свойство регулярного выражения `global`, чтобы найти все совпадения в строке.

Вы делаете поиск глобальным, добавляя аргумент `g` в конец регулярного выражения (так же, как в случае с аргументом `i` для нечувствительного к регистру поиска, как описано в разделе «Построение регулярного выражения» ранее в главе):

```
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
```

Обратите внимание, что аргумент `g` находится после слеша `/`, который используется для заключения маски. Это регулярное выражение производит глобальный поиск; при использовании с функцией `match()` оно ищет каждое совпадение в строке и возвращает массив, содержащий весь соответствующий текст, что является отличным способом нахождения, например, всех URL-адресов в сообщении блога или всех экземпляров слова в длинном тексте.

Вы могли бы переписать код, приведенный в начале раздела, используя глобальный поиск, так:

```
// создание переменной, содержащей строку с URL-адресом
var text='существует много отличных сайтов вроде ↵
www.litres.ru и http://www.eksmo.ru';
// создание регулярного выражения с глобальным поиском
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
// нахождение соответствия регулярному выражению в строке
var url = text.match(urlRegex);
alert(url[0]); // www.litres.ru
alert(url[1]); // http://www.eksmo.ru
```

Вы можете определить количество совпадений, получив доступ к свойству `length` результирующего массива: `url.length`. Этот пример возвратит число 2, так как в проверяемой строке найдены два URL-адреса. Кроме того, вы получаете доступ к каждой найденной строке, используя индекс массива (как описано в разделе «Доступ к элементам массива» главы 2), так, в этом примере, часть `url[0]` указывает на первое совпадение, а часть `url[1]` — на второе.

Имейте в виду то, что использование глобального поиска совместно с методом `match()` не возвращает значение свойства `index` совпавшей строки, а также какой-либо информации о подмасках. Глобальный поиск возвращает только массив, содержащий каждое совпадение в рамках искомой строки.

Замена текста

Вы также можете использовать регулярные выражения для замены текста в строке. Например, у вас есть строка, содержащая дату в формате: 28.10.2014. Однако вам нужно, чтобы дата была в формате 28/10/2014. Этого можно добиться с помощью функции `replace()`, таким образом:

```
string.replace(regex,'replace');
```

Функция `replace()` принимает два аргумента: первый — это регулярное выражение, которое вы хотите найти в строке, второе — строка, заменяющая любое совпадение. Поэтому для изменения формата даты с 28.10.2014 на 28/10/2014 вы можете использовать такой код:

```
1 var date='28.10.2014'; // строка
```

```
2 var replaceRegex = /\./g // регулярное выражение
3 var date = date.replace(replaceRegex, '/');
//заменить.на/
4 alert(date); // 28/10/2014
```

Строка 1 создает переменную и сохраняет в ней строку `'28.10.2014'`. В настоящей программе эта строка могла бы представлять данные, введенные в форму. Строка 2 создает регулярное выражение: слеши / и / отмечают начало и окончание маски регулярного выражения, символы `\.` указывают на буквальное значение точки, а аргумент `g` означает глобальную замену, то есть замену каждого экземпляра точки. Если бы вы не использовали аргумент `g`, то заменена была бы только первая найденная точка, и в итоге бы получилась строка: `'28/10.2014'`. Строка 3 выполняет замену — заменяет каждую `.` на `/` и сохраняет результат в переменной `date`. Наконец, по-новому отформатированная дата — 28/10/2014 — отображается в окне оповещения.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Использование подмасок для замены текста

Функция `replace()` может не только заменять текст (как в примере с точкой (`.`) в дате 28.10.2014) другой строкой (вроде слеша (`/`)), но и запоминать *подмаски* в регулярном выражении и использовать их при замене текста. Как говорилось в примечании в подразделе «Номер телефона», подмаска — это любая часть регулярного выражения, заключенная в круглые скобки. Например, `(il)` в регулярном выражении `/Apr(il)?\b/` является подмаской.

Использование функции `replace()`, показанное в начале данного раздела изменяет строку `28.10.2014` на `28/10/2014`. Однако что если вы также хотите перевести даты в таких форматах, как `28 10 2014` или `28-10-2014` в тот же формат `28/10/2014`? Вместо того чтобы писать несколько строк кода JavaScript для замены точек, пробелов и дефисов, вы можете создать общую маску, соответствующую любому из данных форматов:

```
var date='28-10-2014';
var regex = /([0123]?\d)[-\/ .]([01]?\d)[-\/ .](\d{4})/;
date = date.replace(regex, '$1/$2/$3');
```

Этот пример использует регулярное выражение для поиска соответствия формату даты (см. подраздел «Дата» данной главы). Обратите

внимание на сгруппированные маски в круглых скобках, например, `([0123]?\d)`. Каждая подмаска соответствует одной части даты. Функция `replace()` запоминает строки, соответствующие этим подмаскам, и может использовать их в качестве частей заменяющей строки. В этом случае заменяющая строка такова: `'$1/$2/$3'`. Знак `$`, за которым следует число, представляет одну из соответствующих подмасок. Выражение `$1`, например, соответствует первой подмаске — числу. Таким образом, эта строка переводится, так: «помести сюда первую подмаску, за которым следует /, подмаска, / и последняя подмаска».

Тестирование регулярных выражений

Вы найдете примеры регулярных выражений в файле *пример_регулярное выражение.txt* в папке *глава16*. Кроме того, вы найдете файл *регулярное выражение_тест.html* в папке *эксперименты*. Вы можете открыть эту веб-страницу в браузере и попробовать создать собственные регулярные выражения (рис. 16.5). Просто введите строку, которую вы хотите проверить в поле «Строка для проверки», а затем введите регулярное выражение (опустите начальный и конечный символы /, используемые при создании регулярного выражения на языке JavaScript, просто введите маска.)

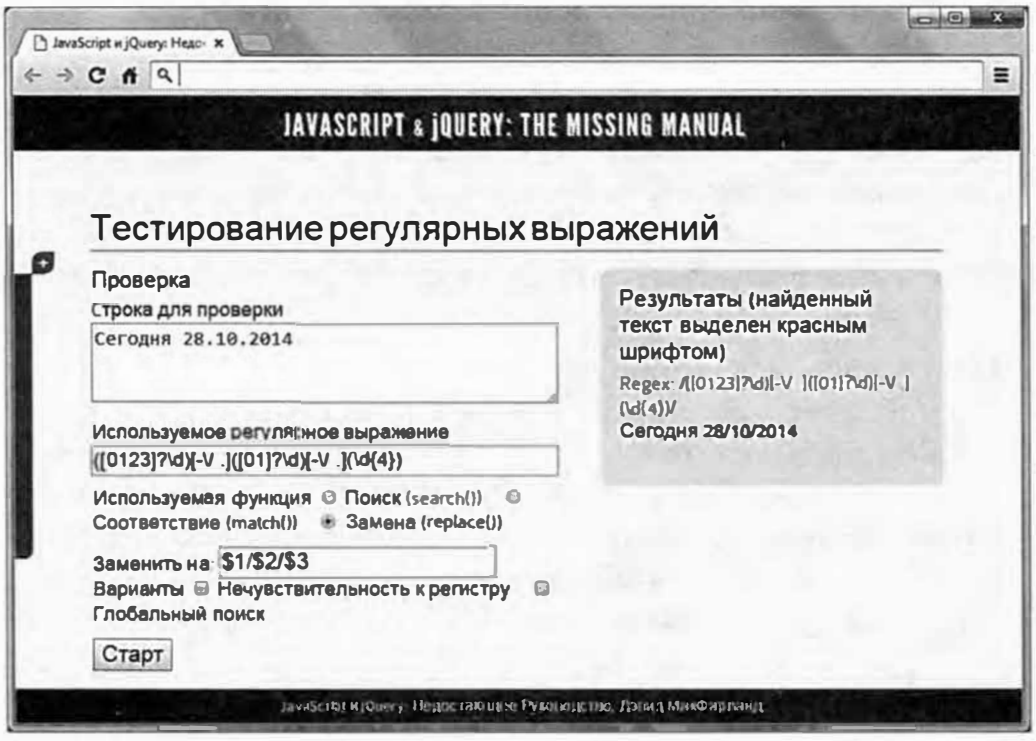

Рис. 16.5. Эта страница, находящаяся в папке с файлами примеров, позволяет протестировать регулярные выражения, используя разные способы («Поиск» или «Соответствие») и варианты («Нечувствительность к регистру» или «Глобальный поиск»)

Затем вы можете выбрать метод: «Поиск», «Соответствие» или «Замена», а также варианты «Нечувствительность к регистру» или «Глобальный поиск». Нажмите кнопку «Старт» и посмотрите, как работают регулярные выражения.

Работа с числами

Числа являются важной частью процесса программирования. Они позволяют вам выполнять такие задачи, как расчет итоговой стоимости заказа, определение расстояния между двумя точками или генерирование случайных чисел от 1 до 6, как при игре в кости. Язык JavaScript предусматривает множество различных способов работы с числами.

Преобразование строки в число

Когда вы создаете переменную, вы можете сохранить в ней число таким образом:

```
var a = 3.25;
```

Однако бывают случаи, когда число является строкой. Например, при использовании функции `prompt()` (см. раздел «Запрос информации на практике» главы 2) для извлечения пользовательского ввода, даже если кто-то введет 3.25, вы получите строку, содержащую число. Иначе говоря, результатом будет `'3.25'` (строка), а не 3.25 (число). Часто этот метод не создает проблем, так как интерпретатор JavaScript обычно конвертирует строку в число, когда считает, что это необходимо. Например:

```
var a = '3';
var b = '4';
alert(a*b); // 12
```

В данном примере, несмотря на то, что обе переменные `a` и `b` являются строками, интерпретатор JavaScript конвертирует их в числа, чтобы выполнить умножение (3×4) и возвращает результат: 12.

Однако когда вы используете операцию +, интерпретатор JavaScript не будет выполнять преобразование, и вы можете получить такой странный результат:

```
var a = '3';
```

```
var b = '4';
alert(a+b); // 34
```

В данном случае обе переменные a и b являются строками. Операция + не только выполняет сложение, но и объединяет (конкатенирует) две строки (см. раздел «Объединение строк» главы 2). Поэтому вместо результата сложения чисел 3 и 4, вы получите объединение двух строк: 34.

Для преобразования строки в число язык JavaScript предусматривает несколько способов:

- Функция Number() преобразует любую переданную ей строку в число:

  ```
  var a = '3';

  a = Number(a); // переменная a теперь является числом 3
  ```

  Таким образом, проблема со сложением двух строк, содержащих числа, может быть решена так:

  ```
  var a = '3';

  var b = '4';

  var total = Number(a) + Number(b); // 7
  ```

  В целях экономии времени можно использовать операцию +, который делает то же, что и функция Number(). Просто добавьте символ + перед переменной, содержащей строку, и интерпретатор JavaScript преобразует строку в число.

  ```
  var a = '3';

  var b = '4';

  var total = +a + +b // 7
  ```

  Недостаток этих двух техник в том, что если строка содержит что-то кроме чисел, одной точки и знаков + или – в начале строки, то в результате вы получите значение JavaScript (NaN), что означает «нечисло» (см. следующий раздел).

- Функция parseInt() также пытается преобразовать строку в число. Однако в отличие от функции Number() метод parseInt() постарается преобразовать в число даже строку, содержащую буквы, если эта строка начинается с числа. Эта команда может пригодиться,

когда вы получаете строку вроде `'20 лет'` в качестве ответа на вопрос о чьем-то возрасте:

```
var age = '20 лет';
age = parseInt(age); //20
```

Функция `parseInt()` ищет либо число, либо знак + или – в начале строки и продолжает искать числа, пока не встретит «нечисло». Таким образом, в вышеприведенном примере функция возвращает число `20` и игнорирует оставшуюся часть строки: `' лет'`.

В дополнение к строковому значению метод `parseInt()` принимает второй аргумент `radix`, который определяет, какую систему счисления следует использовать. В большинстве случаев вы будете использовать привычную десятичную систему счисления. Однако это не единственный вариант. Существуют также восьмеричные и шестнадцатеричные числа.

ПРИМЕЧАНИЕ

Восьмеричные числа начинаются с 0, поэтому вам может встретиться вариант применения метода `parseInt()` к строке `"010"`, а поскольку строка начинается с 0, браузер может обработать ее как восьмеричное число и выдать странный результат. В этом случае восьмеричное число 10 соответствует десятичному значению 8. Некоторые браузеры в этих случаях будут использовать по умолчанию восьмеричную систему счисления, а другие — десятичную (что является актуальным стандартом JavaScript). Другими словами, разные браузеры могут предоставить различные результаты.

Таким образом, при использовании метода `parseInt()` всегда следует применять десятичную систему счисления.

```
var age = '010' лет;
age = parseInt(age,10); // 10
```

- Функция `parseFloat()` похожа на функцию `parseInt()`, но вы используете ее, когда строка может содержать десятичную дробь. Например, если у вас есть строка вроде `'4.5 акра'`, то вы можете использовать функцию `parseFloat()` для извлечения всего значения, включая дробную часть:

```
var space = '4.5 акра';
space = parseFloat(space); // 4.5
```

Если бы вы использовали метод `parseInt()` в предыдущем примере, то получили бы в результате просто число 4, так как функция `parseInt()` извлекает только целые числа.

Выбор метода зависит от ситуации: если вашей целью является сложение двух чисел, являющихся строками, то используйте функцию `Number()` или операцию `+`. Однако если вы хотите извлечь число из строки, которая может содержать букву, например `'200 px'` или `'1.5 em'`, то используйте функцию `parseInt()` для поиска целых чисел (например, 200) или метод `parseFloat()` для извлечения десятичных дробей (например, 1.5).

Тест на числа

При использовании кода JavaScript для обработки пользовательского ввода вам часто нужно удостовериться, что введенная пользователем информация соответствует определенному типу. Например, если вы спрашиваете год рождения человека, вы хотите быть уверены, что он введет число. Точно так же, когда вы выполняете математические расчеты, если используемые данные не являются числами, то ваш сценарий не будет работать.

Чтобы удостовериться в том, что строка является числом, используйте метод `isNaN()`. Эта функция принимает в качестве аргумента строку и проверяет, является ли строка «нечислом». Если строка содержит что-то кроме знака + или – (для положительных и отрицательных чисел), за которым следуют числа, а также десятичного значения (его присутствие не обязательно), то она считается «нечислом», таким образом, строка `'-23.25'` является числом, а строка `'24 пиксела'` — нет. Этот метод возвращает либо значение `true` (если строка не является числом) или `false` (если она является числом). Вы можете использовать функцию `isNaN()` в качестве части управляющей инструкции так:

```
var x = '10'; // это число
if (isNaN(x)) {
// не будет выполняться, так как x является числом
} else {
// будет выполняться, так как x является числом
}
```

Округление чисел

Язык JavaScript предусматривает способ округления дробных чисел, например, можно округлить 4.5 до 5. Округление пригодится, когда вы производите расчеты, результатом которых должно стать целое число. Например, вы используете код JavaScript, чтобы динамически устанавливать высоту элемента `div` в пикселах, основываясь на высоте окна браузера. Другими словами, высота элемента `div` рассчитывается, используя значение высоты окна браузера. Любое вычисление может привести к получению десятичной дроби (например, 300.25), но так как нет такого понятия, как .25 пикселов, вам необходимо округлить результат до ближайшего целого значения (например, 300).

Вы можете округлить число, используя метод `round()` объекта `Math`. Синтаксис этой функции несколько необычен:

```
Math.round(number)
```

Вы передаете дробное число (или переменную, содержащую дробное число) функции `round()`, а она возвращает целое число. Если исходное число содержало дробную часть, меньшую .5, то число округляется в меньшую сторону, если дробная часть была больше .5, то число округляется в большую сторону. Например, число 4.4 будет округлено до 4, а 4.5 — до 5.

```
var decimalNum = 10.25;
var roundedNum = Math.round(decimalNum); // 10
```

ПРИМЕЧАНИЕ

Язык JavaScript предусматривает два других метода для округления чисел: `Math.ceil()` и `Math.floor()`. Они используются, как и метод `Math.round()`, но функция `Math.ceil()` всегда округляет число в большую сторону (например, код `Math.ceil(4.0001)` возвращает число `5`), а функция `Math.floor()` всегда округляет число в меньшую сторону: код `Math.floor(4.99999)` возвращает число `4`.

Форматирование значений в валюте

При расчете стоимости заказа могут использоваться десятичные дроби: 9.99. Однако даже если значение стоимости представлено целым числом, часто употребляются два нуля: 99.00. Значение стоимости 8.9

записывается так: 8.90. К сожалению, язык JavaScript не воспринимает числа таким образом: он отбрасывает нули (например, 10 вместо 10.00, 8.9 вместо 8.90).

К счастью, существует метод `toFixed()`, который позволяет вам преобразовать число в строку, соответствующую нужному количеству десятичных знаков. Чтобы использовать эту функцию, добавьте точку после числа (или после имени переменной, содержащей число) и часть `toFixed(2)`:

```
var cost = 100;
var printCost = cost.toFixed(2) + 'руб.'; // 100.00 руб.
```

Число, которое вы передаете функции `toFixed()` определяет количество десятичных знаков. Для значений в валюте используйте число 2, чтобы в результате получить такие значения, как 100.00 или 99.90. Если вы используете число 3, то получите результат, содержащий три десятичных знака: 10.000 или 9.900.

Если исходное число имеет больше десятичных знаков, чем вы указали, то это число округляется до указанного количества знаков после запятой. Например:

```
var cost = 100.299;
var printCost = cost.toFixed(2) + 'руб.'; // 100.30 руб.
```

В данном случае число 100.299 округляется до 100.30.

ПРИМЕЧАНИЕ

Функция `toFixed()` применима только к числам. Если вы используете ее по отношению к строке, то в результате получите ошибку:

```
var cost='10';//строка
var printCost= cost.toFixed(2) + 'руб.';//ошибка
```

Чтобы обойти эту проблему, вам нужно преобразовать строку в число, как описано в разделе «Преобразование строки в число» данной главы:

```
var cost='10';//строка
cost = +cost; // или cost = Number(cost);
var printCost= cost.toFixed(2) + 'руб.';// 10.00 руб.
```

Генерация случайных чисел

Случайные числа помогают добавить в программу разнообразия. Например, у вас есть массив с вопросами для викторины (как в разделе «Создание простой викторины на практике» главы 3). Вместо того чтобы всегда задавать вопросы в одном и том же порядке, вы можете выбирать их случайным образом. Или использовать код JavaScript для случайного выбора из массива имени графического файла и отображения различных изображений при загрузке страницы. Обе эти задачи решаются путем генерации случайных чисел.

Язык JavaScript предусматривает метод `Math.random()` для генерации случайных чисел. Данный метод возвращает случайное число от 0 до 1 (например, .9716907176080688 или .10345038010895868). Хотя вам могут быть не нужны подобные числа, вы можете использовать некоторые простые математические операции для генерации целого числа от 0 до какого-либо другого числа. Например, чтобы сгенерировать число от 0 до 9 допустимо использовать такой код:

```
Math.floor(Math.random()*10);
```

Этот код состоит из двух частей. Часть внутри функции `Math.floor()` — `Math.random()*10` — генерирует случайное число от 0 до 10. Эта часть станет возвращать такие числа, как 4.190788392268892; а так как случайное число находится в диапазоне от 0 *до* 10, то оно никогда не будет равно 10. Чтобы получить целое число, получившееся случайное число передается функции `Math.floor()`, которая округляет любую десятичную дробь в меньшую сторону, так 3.4448588848 становится 3, а .1111939498984 становится 0.

Если вы хотите получить случайное число в диапазоне от 1 до любого другого числа, просто умножьте результат работы функции `Math.random()` на число, соответствующее верхнему значению диапазона, а затем используйте метод `Math.ceil`, который округляет число в большую сторону. Например, если вы хотите имитировать бросок кости для получения числа от 1 до 6, используйте такой код:

```
var roll = Math.ceil(Math.random()*6); // 1,2,3,4,5 или 6
```

Выбор элемента массива случайным образом

Вы можете использовать метод `Math.random()` для случайного выбора элемента массива. Как говорилось в разделе «Доступ к элементам

массива» главы 2, доступ к каждому элементу массива осуществляется с помощью индекса. Первый элемент массива использует индекс, равный 0, а последний элемент — порядковый номер — 1. Использование функции `Math.random()` упрощает случайный выбор элемента массива:

```
varpeople=['Иван','Саша','Ирина','Вова'];//создание массива
var random = Math.floor(Math.random() * people.length);
var rndPerson = people[random]; //
```

Первая строка данного кода создает массив, содержащий четыре имени. Вторая строка выполняет два действия: сначала она генерирует случайное число от 0 до числа элементов массива (`people.length`) — в данном примере число от 0 до 4. Затем это число округляется в меньшую сторону с помощью функции `Math.floor()`, таким образом, в результате получатся числа 0, 1, 2 или 3.

Наконец, полученное значение используется для получения доступа к одному из элементов массива и его сохранения в переменной `rndPerson`.

Функция для выбора случайного числа

Функции — это отличный способ создания полезных фрагментов кода для многократного использования (см. раздел «Функции: многократное использование кода» главы 3). Если вы часто используете случайные числа, то вам может понадобиться простая функция, помогающая выбрать случайное число в некотором диапазоне, например, число от 1 до 6 или от 100 до 1000. Следующая функция вызывается с помощью двух аргументов: первый — это наименьшее число диапазона (например, 1), а второе — наибольшее число (например, 6):

```
function rndNum(from, to) {
return Math.floor((Math.random()*(to - from + 1)) + from);
}
```

Чтобы использовать эту функцию, добавьте ее на свою страницу (как рассказано в разделе «Функции: многократное использование кода» главы 3), а затем вызовите ее так:

```
var dieRoll = rndNum(1,6); //получение числа от 1 до 6
```

Дата и время

Если вы хотите отслеживать дату и время, обратитесь к объекту JavaScript `Date`. Этот специальный объект позволяет определить год, месяц, день недели, время и многое другое. Чтобы использовать его, создайте переменную и сохраните в ней новый объект `Date`:

```
var now = new Date();
```

Команда `new Date()` создает объект `Date`, содержащий текущую дату и время. После его создания вы можете получить доступ к различным фрагментам информации относительно даты и времени, используя различные методы, перечисленные в табл. 16.3. Например, чтобы выяснить текущий год, используйте функцию `getFullYear()`:

```
var now = new Date();
var year = now.getFullYear();
```

ПРИМЕЧАНИЕ

Фрагмент `new Date()` возвращает текущее время и дату, определенную компьютером пользователя. Другими словами, если у кого-то на компьютере неправильно установлены часы, то дата и время не будут точными.

Табл. 16.3. Функции для доступа к частям объекта `Date`

Функция	Результат
`getFullYear()`	Год, например, 2014
`getMonth()`	Месяц от 0 до 11: 0 — это январь, а 11 — это декабрь
`getDate()`	Число месяца от 1 до 31
`getDay()`	День недели от 0 до 6. 0 — это воскресенье, а 6 — это суббота
`getHours()`	Час от 0 до 23
`getMinutes()`	Количество минут от 0 до 59
`getSeconds()`	Количество секунд от 0 до 59
`getTime()`	Общее количество миллисекунд, прошедших с полуночи 1 января 1970 года (см. врезку «Объект `Date`, работающий за кадром»)

Получение информации о месяце

Чтобы извлечь из объекта `Date` число, соответствующее месяцу, используйте метод `getMonth()`:

```
var now = new Date();
var month = now.getMonth();
```

Однако вместо того, чтобы возвращать число, имеющее какой-то смысл (например, 1 для обозначения января), эта функция возвращает число на единицу меньше. Например, январь — это 0, февраль — это 1 и т. д. Если вы хотите извлечь число, соответствующее нашему пониманию, просто добавьте 1:

```
var now = new Date();
var month = now.getMonth()+1;//привычное число
```

Язык JavaScript не предусматривает команды, сообщающей название месяца. К счастью, странный способ нумерации месяцев может пригодиться, когда вы хотите определить название месяца. Вы можете решить эту задачу, создав массив, содержащий названия месяцев, а затем получить доступ к названию, используя индекс этого месяца:

```
varmonths=['Январь','Февраль','Март','Апрель','Май', ↵
'Июнь','Июль','Август','Сентябрь', ↵
'Октябрь','Ноябрь','Декабрь'];
var now = new Date();
var month = months[now.getMonth()];
```

Первая строка создает массив, содержащий названия всех двенадцати месяцев по порядку. Помните, что для получения доступа к элементу массива вы используете индекс, и что элементы массива индексируются, начиная с 0 (см. раздел «Доступ к элементам массива» главы 2). Таким образом, чтобы получить доступ к первому элементу массива `months`, вы используете фрагмент кода `months[0]`. С помощью метода `getMonth()` вы извлекаете число, которое можно использовать в качестве индекса в массиве `months` и, таким образом, извлечь название этого месяца.

Получение информации о дне недели

Функция `getDay()` возвращает день недели: 0 — воскресенье, а 6 — суббота. Поскольку название дня недели обычно полезнее для ваших посетителей, вы можете использовать массив для хранения названий дней недели и использовать метод `getDay()` для получения доступа к конкретному дню:

```
var days = ['Воскресенье',' ↵
Понедельник','Вторник','Среда', ↵
'Четверг','Пятница','Суббота'];
var now = new Date();
var dayOfWeek = days[now.getDay()];
```

Получение информации о времени

Объект `Date` также содержит информацию о текущем времени, так что вы можете отобразить его на веб-странице или использовать время для определения того, в какой половине дня ее просматривает посетитель. Затем вы сможете как-то использовать эту информацию, например, отобразить фоновое изображение солнца в течение дня, или луны ночью.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Объект `Date`, работающий за кадром

Язык JavaScript позволяет вам получить доступ к таким элементам объекта `Date`, как год или число месяца. Однако интерпретатор JavaScript воспринимает дату как количество *миллисекунд*, прошедших с полночи 1 января 1970 года. Например, среда, 1 декабря 2014 года — это, на самом деле, 1417420800000 для интерпретатора JavaScript.

Это не шутка: для интерпретатора JavaScript отсчет времени идет с 1 января 1970 года. Эта дата (которая называется «Эпоха Unix») была выбрана в 70-х годах программистами, работавшими над операционной системой Unix, чтобы они могли договориться о способе отслеживания времени. С тех пор этот способ стал использоваться во многих языках программирования и платформах.

Каждый раз при использовании метода объекта `Date`, например, `getFullYear()`, интерпретатор JavaScript вычисляет текущий год (на основании количества секунд, прошедших с 1 января 1970 года). Если вы хотите увидеть количество миллисекунд, соответствующее конкретной дате, вы используете метод `getTime()`:

```
var sometime = new Date();
var msElapsed = sometime.getTime();
```

Отслеживание даты и времени в миллисекундах упрощает расчет разницы между датами. Например, вы можете определить, сколько осталось до нового года, получив количество миллисекунд, которое

должно пройти с 1.1.1970 до 1 января следующего года, а затем вычтя из получившегося значения количество миллисекунд, прошедшее с 1.1.1970 до сегодняшнего дня:

```
//количество миллисекунд с 1.1.1970 до сегодняшнего дня
var today = new Date();
// количество миллисекунд с 1.1.1970 до 1-го января следующего года
var nextYear = new Date(2015,0,1);
// количество миллисекунд с сегодняшнего дня до нового года
var timeDiff = nextYear - today;
```

Результат вычитания одной даты из другой — это разница между ними в миллисекундах. Если вы хотите преобразовать это число в нечто полезное, просто разделите его на количество миллисекунд в дне (чтобы выяснить количество дней) или на количество миллисекунд в часе (чтобы выяснить количество часов) и т. д.

```
var second = 1000; // 1000 миллисекунд в секунде
var minute = 60*second; // 60 секунд в минуте
var hour = 60*minute; // 60 минут в часе
var day = 24*hour; // 24 часа в сутках
var totalDays = timeDiff/day; // общее количество дней
```

(В данном примере вы можете заметить другой способ создания даты: `new Date(2015,0,1)`. Подробнее об этом методе сказано в начале раздела «Дата и время».)

Вы можете использовать функции `getHours()`, `getMinutes()` и `getSeconds()` для получения значений часа, минут и секунд. Чтобы отобразить время на веб-странице, добавьте следующий фрагмент в раздел HTML-кода, в котором вы хотите расположить часы:

```
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes();
var seconds = now.getSeconds();
document.write(hours + ":" + minutes + ":" + seconds);
```

Этот код возвращает значение в виде 6:35:56, что означает 6 часов утра, 35 минут и 56 секунд. Однако он также может возвратить значение,

которое вам не понравится, например, 18:4:9, для обозначения 18 часов, 4 минут и 9 секунд. Проблема в том, что большинство людей, читающих эту книгу, не используют часы с 24 часами, если только они не служат в вооруженных силах. Они не воспринимают 18 как 6 часов вечера. Еще большей проблемой является то, что время должно отображаться в формате, предусматривающем по два знака для минут и секунд (даже если число меньше 10), например, 6:04:09. К счастью, подстроить вышеприведенный сценарий для соответствия данным требованиям несложно.

Приведение показания часов к 12-часовому формату

Для изменения часового формата с 24-часового на 12-часовой вам нужно выполнить два действия. Сначала вам необходимо определить, показывают ли часы утреннее время (тогда можно добавить после значения времени «*до полудня*» или «*am*») или вечернее (тогда добавить «*после полудня*» или «*pm*»). Далее вам нужно преобразовать количество часов большее 12 в их 12-часовой эквивалент (например, поменять 14 на 2 после полудня или на 2 p.m.).

Ниже представлен код для решения этой задачи:

```
1 var now = new Date();
2 var hour = now.getHours();
3 var am_pm;
4 if (hour < 12) {
5     am_pm = 'am';
6 } else {
7     am_pm = 'pm';
8 }
9 hour = hour % 12;
10    if (hour==0) {
11          hour = 12;
12 }
13 hour = hour + ' ' + am_pm;
```

ПРИМЕЧАНИЕ

Числа слева — это просто нумерация строк, которая облегчает чтение кода. Не записывайте эти числа в ваш код!

Строки 1 и 2 получают текущую дату и время и сохраняют значение текущего часа в переменной `hour`. Строки 3–7 определяют половину дня: если значение часа меньше 12 (час после полуночи — 0), то это время до полудня (a.m.), в противном случае — это время после полудня (p.m.).

Строка 8 представляет математическую операцию, называемую *модулем* и обозначаемую знаком %. Он возвращает остаток от деления. Например, 5 делится на 2 с остатком 1. Иначе говоря, 5 % 2 = 1. Поэтому в данном случае, если значение часа равно 18, то 18 % 12 = 6 (18 делится на 12 с остатком 6). Таким образом, 18 часов равнозначно 6 часам после полудня или 6 p.m., что вам и нужно. Если делимое меньше делителя (например, при делении 8 на 12), то результатом будет исходное число. Например, 8 % 12 = 8, иначе говоря, операция модуля не меняет значение часа до полудня.

Строки 9–11 берут на себя заботу о возможных результатах применения операции модуля. Если значение часа равно 12 (полдень) или 0 (после полуночи), то операция модуля возвращает 0. В данном случае, значение часа устанавливается на 12: 12 p.m. или 12 a.m. Наконец, строка 12 объединяет отформатированное значение часа с пробелом и строками `am` или `pm`, так что результат отображается в виде `6 am` или `6 pm`.

Отступы для одиночных цифр

Как обсуждалось ранее, когда значения минут или секунд не превышают десяти, то вы можете получить странный результат вроде 7:3:2 p.m. Чтобы изменить этот результат на более привычный 7:03:02 p.m., вам нужно добавить 0 перед одиночной цифрой. Это легко сделать с помощью простой управляющей инструкции:

```
1 var minutes = now.getMinutes();
2 if (minutes < 10) {
3     minutes = '0' + minutes;
4 }
```

Строка 1 получает значение минут, которое может быть и 33, и 3. Строка 2 просто проверяет, является ли данное число меньшим 10, то есть является ли оно одиночной цифрой и нуждается ли в добавлении 0. Строка 3 немного запутана, поскольку обычно вы не можете добавить 0 перед числом: 0 + 2 = 2, а не 02. Однако вы можете складывать строки таким способом, так что `'0' + minutes` означает объединение строки `'0'` со значением переменной `minutes`. Как говорилось в разделе «Преобразование строки в число» ранее в данной главе, при сложении

строки и числа интерпретатор JavaScript преобразует число в строку, поэтому в результате вы получите строку, например, '08'.

Вы можете собрать все эти части воедино для создания простой функции, которая будет возвращать время в форматах 7:32:04 p.m. или 4:02:34 a.m., или даже опускать секунды: 7:23 p.m.:

```
function getTime(secs) {
var sep = ':'; //разделитель
var hours, minutes, seconds, time;
var now = new Date();
var am_pm;
hours = now.getHours();
if (hours < 12) {
am_pm = 'am';
} else {
am_pm = 'pm';
}
hours = hours % 12;
if (hours == 0) {
hours = 12;
}
time = hours;
minutes = now.getMinutes();
if (minutes<10) {
minutes = '0' + minutes;
}
time += sep + minutes;
if (secs) {
seconds = now.getSeconds();
if (seconds < 10) {
seconds = '0' + seconds;
}
time += sep + seconds;
}
return time + ' ' + am_pm;
}
```

Вы можете найти эту функцию в файле *getTime.js* в папке *глава16*. Посмотреть этот сценарий в действии можно, открыв файл *время.html* в папке *глава16*. Для того чтобы использовать эту функцию, либо присоедините файл *getTime.js* к веб-странице, либо скопируйте функцию на веб-страницу или в другой внешний файл JavaScript. Чтобы отобразить время, просто вызовите функцию: `printTime()`, или, если вам нужно отобразить секунды, `printTime(true)`. Функция возвратит строку, содержащую время в правильном формате.

Создание даты, отличной от сегодняшней

Вплоть до этого момента вы видели, как использовать функцию `new Date()` для получения текущей даты и времени на компьютере посетителя. Но что если вы хотите создать объект `Date` для следующего Нового года или Рождества? Язык JavaScript позволяет вам создать дату, отличную от сегодняшней, разными способами. Вы можете это сделать, рассчитав разницу между двумя датами: например, «Сколько дней осталось до Нового года?». При использовании метода `Date()` вы также можете указать дату и время в будущем или в прошлом. Вот базовый формат:

```
new Date(год,месяц,день,час,минуты,секунды, миллисекунды);
```

Например, чтобы создать объект `Date` для полудня 1-го января 2012 года, напишите следующее:

```
var ny2015 = new Date(2015,0,1,12,0,0,0);
```

Этот код переводится так: «создай новый объект `Date` для 1 января, 2015 года, 12 часов, 0 минут, 0 секунд и 0 миллисекунд». Вы должны предоставить как минимум год и месяц, однако если вам не обязательно указывать точное время, то вы можете опустить миллисекунды, секунды, минуты и т. д. Например, чтобы просто создать объект `Date` для 1 января 2015 года, используйте такой код:

```
var ny2015 = new Date(2015,0,1);
```

ПРИМЕЧАНИЕ

Помните, что язык JavaScript использует 0 для января, 1 — для февраля и т. д., как описано в разделе «Получение информации о месяце».

Создание даты, отстоящей от сегодняшней на 1 неделю

Из текста в рамке «Объект Date, работающий за кадром» вы узнали, что интерпретатор JavaScript воспринимает дату как количество миллисекунд, прошедших с 1 января 1970 года. Еще одним способом создания даты является передача значения, представляющего количество миллисекунд, соответствующее этой дате:

```
new Date(миллисекунды);
```

Поэтому другим способом создания даты для 1 января 2015 года является следующий фрагмент кода:

```
var ny2015 = new Date(1420099200000);
```

Поскольку большинство из нас не являются ходячими калькуляторами, вы, вероятно, не воспринимаете дату таким образом. Однако количество миллисекунд может пригодиться при создании новой даты, которая отстоит от другой даты на определенное время. Например, при установке cookie, используя язык JavaScript, вы должны указать дату, когда данные cookie будут удалены из браузера посетителя. Чтобы быть уверенными, что cookie исчезнут через одну неделю, вам нужно указать эту дату.

ПРИМЕЧАНИЕ

Только что приведенный код — `new Date(1420099200000);` — работает не для всех. Он применяется только к часовому поясу UTC-8 (тихоокеанское время). Именно поэтому браузеры учитывают временную зону и подстраивают свои часы, исходя из разницы с часовым поясом UTC-8, как описано далее в тексте в рамке.

Чтобы указать день через неделю, можно сделать следующее:

```
var now = new Date(); // сегодня
var nowMS = now.getTime(); // получает количество ↵
миллисекунд, соответствующее сегодняшнему дню
var week = 1000*60*60*24*7; // количество ↵
миллисекунд, соответствующее дню через 1 неделю
var oneWeekFromNow = new Date(nowMS + week);
```

Первая строка сохраняет текущую дату и время в переменной `now`. Далее, функция `getTime()` извлекает количество миллисекунд, прошедшее с 1 января 1970 года до настоящего момента. Третья строка вы-

числяет общее количество миллисекунд в одной неделе (1000 миллисекунд × 60 секунд × 60 минут × 24 часа × 7 дней). Наконец, код создает новую дату, добавляя количество миллисекунд в неделе к количеству, соответствующему сегодняшнему дню.

ПРИМЕЧАНИЕ

Объект JavaScript `Date` предусматривает множество различных функций, помогающих с подсчетом дат и работой с разными географическими положениями. Полную справку по работе с объектом `Date` вы можете найти на странице **developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Date**.

КУРС ОПЫТНОГО ПОЛЬЗОВАТЕЛЯ

Работа с различными часовыми поясами

Работа компьютеров заключается не только в отслеживании секунд, минут и дней. Им также приходится координировать свое время со временем других компьютеров, расположенных по всему миру в разных часовых поясах. В конце концов, 8:00 вечера в Москве это не 8:00 вечера в Сан-Франциско. Поскольку два этих города расположены в разных полушариях, солнце там садится и встает с разницей почти в 12 часов, то есть 8:00 вечера в Москве соответствует 8:00 утра в Сан-Франциско и наоборот. (А в летнее время 8:00 вечера в Москве соответствует 9:00 утра в Сан-Франциско.)

Чтобы помочь компьютерам (и людям) в разных часовых поясах синхронизировать свою работу, разработчики программного обеспечения используют конвенцию под названием UTC (Coordinated Universal Time, Универсальное координированное время). Этот часовой пояс раньше назывался «Время по Гринвичу» и соответствовал меридиану с географической долготой, равной 0°. Если вы живете не в Англии, Франции, Испании или странах Африки, расположенных к югу от Англии, то ваш часовой пояс смещен относительно зоны UTC.

Город Сан-Франциско расположен в часовом поясе UTC –8, что означает разницу в 8 часов по сравнению с UTC. Когда в Лондоне 9:15 вечера, в Сан-Франциско 1:15 дня. Москва находится в часовом поясе UTC +4, поэтому, когда в Лондоне 9:15 вечера, в Москве 1:15 ночи. (Опять же, эта разница зависит от того, какое время используется: летнее или зимнее.)

Объект JavaScript `Date` предусматривает функцию для определения смещения (то есть разницы во времени между вашим часовым поясом и часовым поясом UTC):

```
var now = new Date();
var offset = now.getTimezoneOffset();
```

Если ваш компьютер находится в Сан-Франциско, и вы определяете смещение, то вы получите значение либо 480, либо 420. Это число соответствует разнице в минутах между временем вашего компьютера и временем UTC, таким образом, 480 соответствует 8 часам, а 420 — 7 часам (при переходе на летнее время). Если ваш регион переходит на летнее время, то ваш компьютер и веб-браузер также это учитывают (не правда ли, компьютеры умные?)

В общем, вам не нужно беспокоиться о преобразовании часового пояса. JavaScript работает на компьютерах посетителей вашего веб-сайта, поэтому при выполнении таких действий, как отображение текущего времени, браузер адаптируется к часовому поясу посетителя и отобразит соответствующее время.

Оптимизация сценариев JavaScript

Процесс программирования — это большой труд. Поэтому разработчики всегда ищут пути ускорения работы и уменьшения количества строк кода. Несмотря на большое число советов и хитростей, приведенные ниже методы особенно полезны при работе с кодами JavaScript и библиотекой jQuery.

Сохранение параметров в переменных

Один из важных навыков, который усваивают программисты, заключается в том, как извлекать детали сценариев, чтобы упростить их изменение и обновление. Например, вы хотите, чтобы при щелчке кнопкой мыши по фрагменту текста цвет шрифта менялся бы на оранжевый. Сделать это можно с помощью библиотеки jQuery, используя функцию `css()` (см. раздел «Чтение и изменение свойств CSS» главы 4):

```
$('p').click(function() {
$(this).css('color','#F60');
});
```

В данном случае оранжевый цвет (`#F60`) закодирован непосредственно на данном этапе. Допустим, вы хотите применить этот же цвет и на других этапах (например, добавить фоновый цвет при наведении указателя мыши на ячейку таблицы). Вы можете поддаться искушению

и просто ввести фрагмент кода #F60 в соответствующие строки. Но лучше поместить его в переменную в начале сценария и затем пользоваться ею на протяжении всего сценария:

```
1  $(document).ready(function() {
2  var hColor='#F60';
3  $('p').click(function() {
4     $(this).css('color',hColor);
5  });
6  $('td').hover(
7     function() {
8           $(this).css('backgroundColor',hColor);
9     },
10    function() {
11          $(this).css('backgroundColor','transparent');
12    }
13 );
14 }); //конец ready()
```

В данном примере переменная hColor содержит шестнадцатеричное значение цвета — она используется и в событии click для элементов p, и в событии наведения для элементов td. Если в дальнейшем вы решите, что оранжевый — это не ваш цвет, то можете изменить значение, хранящееся в переменной (например, var hColor='#F33';), и сценарий будет использовать новый цвет.

Приведенный код можно сделать еще более гибким, разъединив связь между цветами, используемыми в элементах p и td. Сейчас для них задан один и тот же цвет, но если вы хотите иметь возможность назначать разные цвета каждому из элементов, то в код следует ввести дополнительную переменную:

```
1     $(document).ready(function() {
2  var pColor='#F60';
3  var tdColor=pColor;
4     $('p').click(function() {
5           $(this).css('color',pColor);
```

```
6      });
7      $('td').hover(
8      function() {
9            $(this).css('backgroundColor',tdColor);
10     },
11     function() {
12           $(this).css('backgroundColor','transparent');
13     }
14 );
15 }); //конец ready()
```

Сейчас для событий `click` и `hover` используется один и тот же цвет — #F60 (поскольку в строке 3 для переменной `tdColor` задано значение переменной `pColor`). Однако если позднее вы захотите, чтобы ячейка таблицы имела другой цвет, просто измените строку 3:

```
var tdColor='#FF3';
```

При написании программы JavaScript определите значения, которым вы дадите выразительные имена, и преобразуйте их в переменные. Возможными «кандидатами» являются: цвет, шрифт, ширина, высота, время (например, 1000 миллисекунд), имя файла (например, файла изображения), текст сообщения (например, оповещения или подтверждения) или путь к файлу (например, путь для ссылки или изображения). Например:

```
var highlightColor= '#33A';
var upArrow= 'ua.png';
var downArrow='da.png';
var imagePath='/images/';
var delay=1000;
```

Определите все эти переменные в начале сценария (или, если вы используете библиотеку jQuery, прямо внутри функции `.ready()`).

▶ СОВЕТ

Особенно полезно помещать в переменные текст, который вы планируете отобразить на странице. Например, сообщения об ошибках

типа «Введите корректный электронный адрес» или подтверждающие сообщения вроде «Спасибо за предоставление почтового адреса» могут быть преобразованы в переменные. Если сгруппировать их в начале сценария, то потом будет гораздо легче их редактировать (и переводить на другие языки, если понадобится обратиться к иностранным пользователям).

Сохранение параметров в объектах

Существует более продвинутый способ сохранения параметров с помощью объекта JavaScript. Как вы знаете из раздела «Одновременное изменение нескольких свойств CSS» главы 4, литерал объекта JavaScript является способом сохранения данных в одном объекте с помощью пар имя/значение (или ключ/значение). В приведенном выше примере используется много отдельных переменных для хранения таких распространенных параметров, как цвет, путь к изображению, названия значков и т. д. В использовании этого подхода нет ничего плохого, однако вы можете сгруппировать все эти значения в единый объект, а затем сослаться на отдельные свойства, используя точечный синтаксис (см. раздел «Небольшой урок, посвященный объектам» главы 2).

Например, список из пяти переменных в предыдущем разделе можно было бы сохранить в одном объекте:

```
var siteSettings = {
highlightColor: '#33A',
upArrow: 'ua.png',
downArrow: 'da.png',
imagePath: '/images/',
delay: 1000
}
```

При использовании точечного синтаксиса вы не используете знак равенства для присвоения значения имени, поэтому вы пишете: `upArrow: 'ua.png'`, а не `upArrow='ua.png'`, и вы добавляете запятую после каждой пары имя/значение, кроме последней. Теперь для использования этих параметров в своей программе примените точечный синтаксис, чтобы получить доступ ко всем отдельным свойствам объекта. Например, чтобы извлечь значение свойства `highlightColor`, вы напишете следующее:

```
siteSettings.highlightColor
```

У этого подхода есть несколько преимуществ. Во-первых, он позволяет сделать код более организованным, в отличие от многократного использования большого количества переменных. Вместо этого вы помещаете все свои параметры в единый объект в одном месте в коде. Во-вторых, этот подход позволяет избежать возникновения проблемы при использовании одного и того же имени для двух различных переменных. Например, имя «delay» носит довольно общий характер. Если вы создаете переменную `delay` в своей программе, а в начале программы вы определили переменную `delay` для хранения параметров, то первая версия переменной `delay` будет уничтожена. Тем не менее при сохранении общего для всего сайта параметра `delay` в объекте `siteSettings.delay` он не станет конфликтовать с другой переменной `delay`, которую вы можете добавить в вашу программу.

ПРИМЕЧАНИЕ

Из раздела «Знакомство с плагинами jQuery» главы 7 вы узнали о том, что многие плагины jQuery используют литералы объектов для передачи информации плагину. Виджеты jQuery UI (главы 10 и 11) используют литералы объектов для контроля над внешним видом и принципом работы виджета.

Тернарная операция

Часто в процессе программирования стоит задача присвоить значение переменной на основе определенного условия. Например, вам нужно создать переменную, содержащую текст со статусом авторизации пользователя. В вашем сценарии есть переменная `login`, содержащая логические значения `true`, если посетитель авторизован, или `false`, если не авторизован. Вот один из способов создать новую переменную для этой ситуации:

```
var status;
if (login) {
status='Авторизован';
} else {
status='Не авторизован';
}
```

В данном случае базовая управляющая инструкция (см. раздел «Управляющие инструкции» главы 3) задает значение переменной `status`, основываясь на том, авторизован пользователь или нет. Язык JavaScript предлагает простое решение этой задачи с помощью *тернарной операции*. Тернарная операция позволяет с помощью одной строки кода создать простую условную инструкцию. Базовый формат тернарной операции таков:

```
(условие) ? A : B
```

В зависимости от результата условия операция возвращает *A* (если условие истинно (`true`)) или *B* (если условие ложно (`false`)). Знак `?` стоит перед значением `true`, тогда как знак «:» предшествует значению `false`. Так, например, приведенный выше код можно переписать следующим образом:

```
var status=(login)?'Авторизован:'Не авторизован';
```

То, что было некогда шестью строками кода, превратилось теперь в одну. На рис. 16.6 показано, как работает данный код.

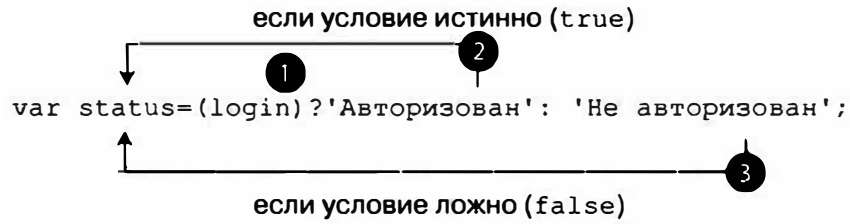

Рис. 16.6. Тернарная операция позволяет писать однострочные управляющие инструкции. В данном примере *1* — это условие. Если оно истинно, возвращается код, непосредственно следующий за знаком `?` (*2*); если условие ложно, возвращается код, непосредственно следующий за знаком `:` (*3*)

Тернарная операция — это просто короткий путь решения задачи. Вы не обязаны ее использовать, многие программисты находят ее слишком сложной для понимания, предпочитая инструкцию `if/e se`. Кроме того, лучше всего использовать тернарную операцию для задавания значения переменной в зависимости от условия. Она пригодна не для каждого типа управляющих инструкций; например, ее нельзя использовать по отношению к многострочным инструкциям, где выполнение строк кода зависит от определенного условия. Но даже если вы не применяете тернарные операции, понимание того, как они работают, поможет вам разобраться в чужих программах, в которых вы наверняка не раз с ними встретитесь.

Инструкция-переключатель

Существует много путей усовершенствования управляющей инструкции. В то время как тернарная операция удобна для присвоения значения переменной в зависимости от результатов выполнения условия, *инструкция-переключатель* или *операция выбора* предлагает более простой способ создания серий инструкций `if/else`, зависящих от значения единственной переменной.

Например, вы просите посетителей вашего сайта ввести названия любимых цветов в поле формы, а затем отображаете разные сообщения в зависимости от выбранного цвета. Вот как можно написать часть такого кода, используя обычную управляющую инструкцию:

```
if (favoriteColor == 'голубой') {
message = 'Голубой — это холодный цвет.';
} else if (favoriteColor == 'красный') {
message = 'Красный — это теплый цвет.';
} else if (favoriteColor == 'зеленый') {
message = 'Зеленый — это цвет листьев.';
} else {
message = 'Что это за цвет?';
}
```

Обратите внимание, как много в этом коде фрагментов `favoriteColor == 'некоторое значение'`. Действительно, текст `favoriteColor ==...` появляется три раза в девяти строках кода. Если все, что вам нужно сделать — это несколько раз проверить значение переменной, то инструкция-переключатель предлагает более элегантное (и легкое для чтения) решение. Базовая структура инструкции-переключателя представлена на рис. 16.7.

Первая строка операции выбора начинается с ключевого слова `switch`, за которым следуют имя переменной в круглых скобках и открывающая фигурная скобка. В сущности, код говорит: «Давайте получим значение этой переменной и посмотрим, совпадает ли оно с каким-либо другим значением». Каждый тест называется `case` (случай), и инструкция-переключатель имеет один или несколько случаев. В примере на рис. 16.7 приведены три случая, пронумерованные от 1 до 3. Базовая структура случая выглядит так:

```
case значение1:
// какое-то действие
break;
```

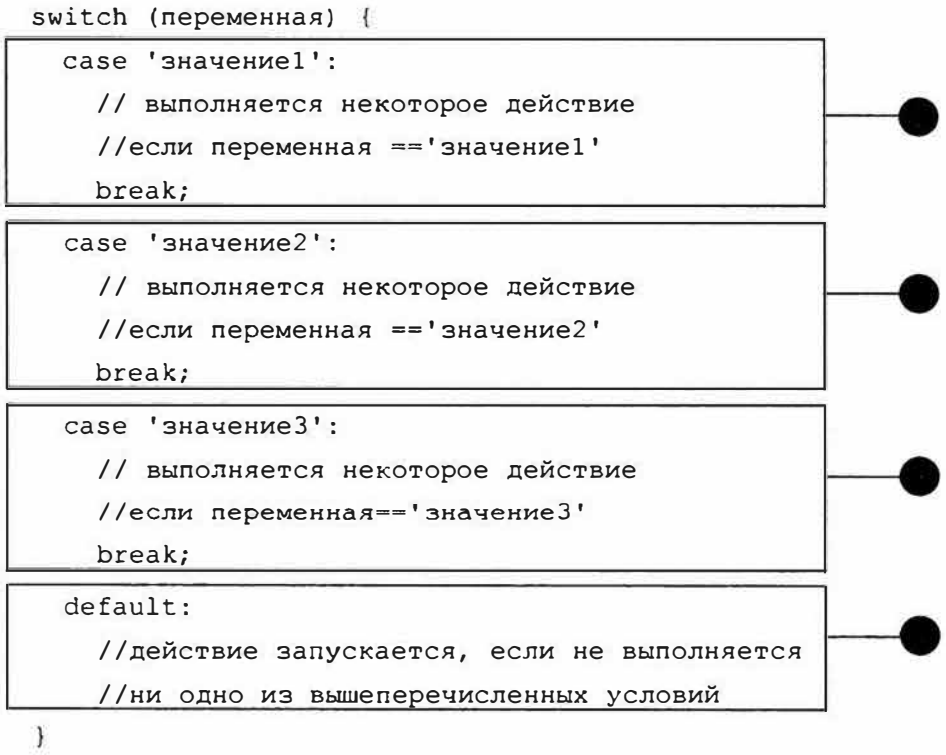

Рис. 16.7. Операция выбора предлагает простой способ выполнения различных действий на основании значения переменной. Не забывайте об операции `break`, позволяющей выйти из операции выбора

Ключевое слово `case` определяет начало случая; затем следуют некое значение и двоеточие. Эта строка является сокращенным вариантом более длинной: `if (variable=='значение1')`. Значение переменной может быть числовым, строковым или логическим, поэтому, если вы хотите проверить, равно ли значение, например, 37, тогда случай примет следующий вид:

```
case 37:
//какое-то действие
break;
```

А чтобы проверить, является значение переменной истинным или нет, вы должны написать следующее:

```
case true:
//какое-то действие
break;
```

После первой строки вы добавляете инструкции, которые должны быть выполнены, если переменная соответствует тестовому значению.

В конце добавляется операция выхода. Этот шаг важен — операция `break;` определяет выход из инструкции-переключателя. Если вы забудете ввести эту операцию, то интерпретатор JavaScript перейдет к следующему тесту и будет искать совпадения там. Другими словами, после проведения одного сравнения, если одно из условий истинно, интерпретатор JavaScript перестанет пытаться отыскать другие совпадения и выполнит оставшийся код в блоке `switch`.

Вот как инструкция-переключатель может помочь преобразовать код, приведенный в начале раздела:

```
switch (favoriteColor) {
case 'голубой':
message = 'Голубой — это холодный цвет.';
break;
case 'красный':
message = 'Красный — это теплый цвет.';
break;
case 'зеленый':
message = 'Зеленый — это цвет листьев.';
break;
default:
message = 'Что это за цвет?';
}
```

Данный код эквивалентен коду `if/else`, но более компактен, удобен для чтения и не требует многократного использования управляющих инструкций (например, `if (favoriteColor === blue)`).

В сущности, вы можете разместить несколько инструкций `case` одну за другой (и намеренно исключить ключевое слово `default`), если хотите выполнить один и тот же код для нескольких значений. Например:

```
switch (favoriteColor) {
case 'синий':
case 'голубой':
case 'индиго':
message = 'Голубой — это холодный цвет.';
```

```
break;
case 'красный':
message = 'Красный — это теплый цвет.';
break;
case 'зеленый':
message = 'Зеленый — это цвет листьев.';
break;
default:
message = 'Что это за цвет?';
}
```

Этот вариант подобен использованию конструкции

```
if (favoriteColor == 'синий' || favoriteColor == ↵
'голубой' || favoriteColor == 'индиго')
```

в инструкции `if/else`.

Объединение массивов и разбиение строк

Массив JavaScript подобен списку покупок. Это набор значений, сохраненных в одной переменной. Например, вы можете создать массив для хранения дней недели:

```
var days = ['Понедельник', 'Вторник', 'Среда', ↵
'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];
```

Каждый элемент в списке представляет собой отдельное значение, и вы можете получить к нему доступ, используя технику, описанную в разделе «Доступ к элементам массива» главы 2. Чтобы получить доступ к первому пункту списка, нужно использовать код `days[0]`. Однако иногда вам может потребоваться извлечь все элементы списка в виде строки, когда необходимо просто вывести на экран все содержимое массива. В этом случае вы можете использовать метод `.join()`, который берет все элементы массива и преобразует их в одну строку. Элементы массива разделяются либо запятой, либо другим разделителем, который вы укажете. Например:

```
var weekdays = days.join(); //Понедельник,Вторник,Среда...
```

Если вы не предоставите методу `join()` никаких аргументов, то в качестве разделителя будет использоваться запятая. Тем не менее вы можете выбрать любой другой разделитель:

```
var weekdays = days.join(':'); //Понедельник:Вторник:Среда...
```

JavaScript не добавляет пробелов между элементами списка и разделителями, поэтому, если вам нужно, чтобы за элементом следовала запятая и пробел, то вам следует передать методу `join()` строку `', '`:

```
var weekdays = days.join(', '); // Понедельник, ↵
Вторник, Среда…
```

С другой стороны, вы можете взять строку и превратить ее в массив, используя метод `split()`, если в строке используется какой-либо разделитель, показывающий, где заканчивается один элемент и начинается другой. Например, у вас есть такая строка:

```
var weekdays = 'Понедельник,Вторник,Среда,Четверг, ↵
Пятница, Суббота,Воскресенье';
```

Вы можете взять эту строку и разбить ее на массив следующим образом:

```
var dayList = weekdays.split(','); // dayList теперь ↵
является массивом, содержащим 7 элементов
```

Выводы

До сих пор, изучая материал настоящей книги, вы встречались со многими задачами, которые выполняются средствами JavaScript: проверка формы, создание сменяемых изображений и фотогалерей, улучшение пользовательского интерфейса, например, с помощью «аккордеонов», панелей вкладок и др. Но, возможно, вам интересно, как соединить все это на одном сайте? В конце концов, начав использовать язык JavaScript, вы, вероятно, захотите с его помощью улучшить каждую страницу. Вот некоторые советы по использованию нескольких сценариев на вашем сайте.

Использование внешних файлов JavaScript

Как указывалось в разделе «Внешние файлы JavaScript» главы 1, внешние файлы JavaScript обеспечивают эффективный способ использования одного и того же кода JavaScript на нескольких веб-страницах. Внешний файл облегчает обновление программы JavaScript, поскольку достаточно отредактировать лишь один файл, чтобы улучшить (или исправить) весь код JavaScript. Кроме того, после загрузки внешний файл

JavaScript хранится в кэше браузера, поэтому его повторной загрузки не требуется, что способствует более быстрому отображению и реакции веб-страниц.

При использовании библиотеки JavaScript, подобной jQuery, внешние файлы JavaScript являются необходимостью. Очевидно, что ваши веб-страницы стали бы необоснованно объемными и неудобными в обслуживании, если бы пришлось вставлять код jQuery в каждую из них. Кроме того, плагины библиотеки jQuery являются внешними файлами, поэтому их нужно присоединять к странице с помощью ссылки, чтобы затем использовать. Создать ссылку на внешний файл JavaScript совсем нетрудно:

```
<script src="js/ui.tabs.js"></script>
```

Поместив собственный код JavaScript во внешние файлы JavaScript, вы также сможете облегчить его повторное использование и ускорить работу сайта, но только если этот код актуален для других страниц. Например, вспомним случай со сценарием для проверки, созданным в разделе «Проверка формы» главы 8. Нет смысла помещать код, используемый для создания правил проверки и сообщений об ошибках, во внешний файл, поскольку все правила и сообщения специфичны только для элементов формы на данной странице и не будут работать для формы с другими элементами. В данном случае лучше использовать код JavaScript для проверки формы внутри самой страницы.

Однако файл плагина Validation, рассмотренный в разделе «Проверка формы» главы 8, можно использовать для любой формы, что делает логичным хранение его в отдельном внешнем файле. То же справедливо для любого кода, применяемого на нескольких страницах. Например, в разделе «Фокусировка на первом элементе формы» главы 8 вы узнали, как выделить первый элемент формы с помощью кода JavaScript, что вы, вероятно, хотели бы сделать по отношению к любой форме. Или обратимся к тексту в рамке «Предотвращение многократной отправки формы» в главе 8, где был представлен код JavaScript, необходимый для предотвращения многократных нажатий кнопки передачи формы (и отправки данных несколько раз), что полезно для любой страницы, содержащей форму. Итак, вы можете объединить два сценария в единый внешний файл, который можно назвать, скажем, *forms.js*. Код JavaScript будет выглядеть примерно следующим образом:

```
1  $(document).ready(function() {
2  // фокус на первом текстовом поле
3  $(":text")[0].focus();
4
5  // отключение кнопки submit после отправки
6  $('form').submit(function() {
7     var subButton = $(this).find(':submit');
8     subButton.attr('disabled',true);
9     subButton.val('...отправка информации...');
10 });
11 }); // конец ready
```

ПРИМЕЧАНИЕ

Как говорилось в разделе «Больше концепций для событий jQuery» главы 5, любой сценарий, помещенный в разделе заголовка страницы и использующий jQuery, требует применения функции `$(document).ready()`. Библиотека jQuery способна без проблем обрабатывать несколько функций `$(document).ready()`. Например, у вас может быть несколько внешних файлов JavaScript, выполняющих различные задачи на странице. Каждый из них может иметь функцию `$(document).ready()`, и вы способны включить эту функцию в элемент `script`, который появляется только на данной странице. Это вполне допустимо. Однако можно обойтись и без функции `$(document).ready()`, поместив элемент `script` в нижней части страницы перед закрывающим тегом `</body>`.

Применение одного и того же сценария на нескольких страницах требует от вас некоторого планирования. Например, строка 3 помещает текстовый курсор в первое текстовое поле формы на веб-странице. Как правило, в этом есть смысл — при загрузке формы текстовый курсор должен находиться в первом поле, чтобы посетитель сразу мог начать заполнение. Однако если страница содержит несколько форм, данный код может работать не так, как вам бы хотелось.

Например, если на вашей странице есть строка поиска в верхней части документа и отдельная форма для составления заказа, то код в строке 3 поместит текстовый курсор в строку поиска, а не в первое поле формы заказа. В данном случае вам нужно немного поразмыслить и найти способ сделать так, чтобы при загрузке страницы выделялось

нужное поле. Существуют два возможных варианта решения этой задачи:

- Присвойте имя класса полю, которое вы хотите выделить при загрузке страницы. Например, вы присвоили имя класса `focus` текстовому полю таким образом:

  ```
  <input type="text" class="focus" name="firstName">
  ```

  Тогда с помощью следующего кода JavaScript можно обеспечить помещение текстового курсора в выбранное поле:

  ```
  $('.focus').focus();
  ```

  Чтобы применить этот код, вам остается убедиться, что класс `focus` присвоен текстовому полю формы на каждой странице с формой и в том, что к каждой из этих страниц присоединен внешний файл JavaScript, содержащий данный код.

- Тот же результат можно получить путем добавления имени класса в сам элемент `form`, используя следующий код JavaScript:

  ```
  $('.focus :text')[0].focus();
  ```

Этот код автоматически выделяет первое текстовое поле формы с классом `focus`. Преимущество данного подхода заключается в том, что первое поле всегда будет содержать текстовый курсор, поэтому если вы решили реорганизовать свою форму (добавив, например, несколько полей в начало), вы знаете, что в любом случае будет выделено именно первое поле и никакое другое (пусть даже с классом `focus`).

Вы можете применять несколько сценариев JavaScript на всех (или почти всех) веб-страницах. Например, на вашей странице могли бы находиться несколько сменяемых изображений (см. раздел «Смена изображений» главы 7) и вы могли бы использовать код JavaScript, чтобы удостовериться, что внешние ссылки будут открываться в новом окне (см. раздел «Открытие внешних ссылок в новом окне» главы 7). В подобной ситуации полезно создать внешний файл JavaScript, который содержит все сценарии, используемые на вашем сайте (назвать его можно, например, *site_scripts.js* или просто *site.js*).

ПРИМЕЧАНИЕ

В библиотеке jQuery есть встроенный механизм защиты от нежелательных ошибок JavaScript. Обычно программа JavaScript сообщает об ошибке, если вы пытаетесь выполнить действие над тем, чего не существует, например, пытаетесь выбрать текстовое поле на стра-

нице, не имеющей такого поля. К счастью, библиотека jQuery просто игнорирует подобные ошибки.

Ускорение загрузки файлов JavaScript

Как только вы начнете использовать в сценариях внешние файлы JavaScript, ваши посетители почувствуют, что сайт стал работать быстрее. Благодаря кэшу браузера внешние файлы JavaScript не требуют повторной загрузки при переходе на другую страницу. Тем не менее существует еще один способ ускорить загрузку содержимого вашего сайта — сжатие внешних файлов JavaScript.

ПРИМЕЧАНИЕ

Файлы, посылаемые по протоколу безопасных соединений (SSL, Secure Socket Layer), *как правило*, не кэшируются. Поэтому, если посетители заходят на страницы вашего сайта, используя протокол *https://* (например, *https://www.sawmac.com*), то любые файлы, которые они загружают, включая и внешние файлы JavaScript, должны загружаться всякий раз, когда они нужны. (Вы можете изменить настройки сервера, чтобы разрешить кэширование файлов, посылаемых по протоколу безопасных соединений.)

Чтобы сделать сценарий более понятным, программисты обычно вставляют пустые строки, возвраты каретки и комментарии, объясняющие действия сценария. Все эти дополнения важны для программиста, но не для браузера, который прекрасно понимает язык JavaScript и без возвратов каретки, табуляции, пустых строк или комментариев. С помощью программ сжатия вы можете минимизировать пространство, занимаемое кодом JavaScript. Например, версия библиотеки jQuery, используемая в данной книге, является *минимизированной* и имеет размер файла, примерно вдвое меньший, чем размер несжатой версии.

Существуют программы для уменьшения размеров файлов JavaScript. В качестве примеров можно привести JSMin (автор Дуглас Крокфорд, **crockford.com/javascript/jsmin.html**) и Packer (автор Дин Эдвард, **dean.edwards.name/packer**). Однако мы будем работать с тем же компрессором, который используется системой Yahoo (и библиотекой jQuery), поскольку он позволяет весьма эффективно сжимать файл, не изменяя при этом его кода (некоторые компрессоры на самом деле переписывают код, что подчас может даже повредить сценарий!)

JavaScript-компрессор Yahoo (YUI) доступен по адресу: **developer.yahoo.com/yui/compressor**. К счастью, существует простой онлайн-инструмент, который позволяет вам использовать компрессор YUI, не устанавливая его на свой компьютер.

1. **Запустите веб-браузер и посетите сайт: refresh-sf.com.**

   Этот сайт посвящен онлайн-компрессору YUI.

2. **Щелкните по вкладке YUI Compressor.**

   Вы также можете просто скопировать код JavaScript из вашего текстового редактора и вставить его в большое текстовое поле на домашней странице сайта. В этом случае вы можете перейти к шагу 4.

3. **Найдите внешний файл JavaScript на своем компьютере и перетащите его в большое текстовое поле на странице сайта.**

   Файл должен содержать только код JavaScript. Например, вы не можете выбрать HTML-файл, который также содержит код на языке JavaScript.

4. **Нажмите кнопку JavaScript.**

   Веб-сайт обработает ваш код.

5. **Нажмите кнопку Save.**

   Теперь вы сохраняете сжатый файл на ваш компьютер. Затем вы можете переименовать этот файл (поскольку он всегда сохраняется под именем *min.js*) и поместить его на свой сайт для дальнейшего использования. После сжатия файла вам будет предоставлен отчет, в котором перечислены такие данные, как исходный размер файла, размер нового файла и процентное значение сжатия.

ПРЕДУПРЕЖДЕНИЕ

Удостоверьтесь, что исходный файл JavaScript находится у вас под рукой, поскольку новая сжатая версия не читается, и вы не сможете отредактировать ее в том случае, если вам понадобится внести изменения в исходный код.

Глава 17
ДИАГНОСТИКА И ОТЛАДКА

Никто не застрахован от ошибок, но ошибки в коде JavaScript приводят к некорректной работе программ или вовсе делают ее невозможной. Начав программировать на языке JavaScript, вы наверняка будете совершать немало ошибок. Выяснение причины ненадлежащего поведения сценария может оказаться нелегким делом, но это неизбежная часть процесса программирования. К счастью, опыт и практика со временем позволят вам понимать причины ошибок и находить способы их устранения.

В данной главе описаны некоторые из наиболее распространенных ошибок и, что более важно, способы их диагностики и, как говорят программисты, *отладки*. Кроме того, приведенное здесь руководство позволит вам шаг за шагом разобрать процесс отладки проблемного сценария.

Наиболее распространенные ошибки при программировании на языке JavaScript

Существует множество причин неправильного выполнения программы: от простых опечаток до трудноуловимых ошибок, проявляющихся только время от времени. Тем не менее можно выделить целый ряд ошибок, которые обычно подстерегают начинающих (и не совсем) программистов JavaScript. Ознакомьтесь с их списком в данном разделе и имейте их в виду в процессе программирования. Возможно, вы убедитесь, что знание этих простых вещей позволит вам намного легче находить неточности в ваших собственных программах и исправлять их.

Незакрытые пары

Как вы уже заметили, код JavaScript содержит большое число кавычек, скобок, точек с запятой и других знаков препинания. В силу «педантичной сущности» компьютеров пропуск одного знака препинания может помешать выполнению программы. Наиболее распространенной ошибкой является отсутствие закрывающего знака пунктуации. Напри-

мер, выполнение кода `alert('привет';` приведет к ошибке, поскольку пропущена закрывающая скобка: `alert('привет');`.

Отсутствие закрывающей скобки приведет к синтаксической ошибке (см. врезку «Типы ошибок» далее в главе), которая прервет выполнение сценария. Когда вы тестируете сценарий, браузер сообщает о допущенных синтаксических ошибках, но, увы, различные браузеры описывают проблемы по-разному. В консоли ошибок браузера Firefox (см. раздел «Консоль JavaScript в браузере Firefox» главы 1) вы получаете сообщение типа *Missing) after arguments list* (*Отсутствие) после списка аргументов*); браузер Internet Explorer (см. раздел «Отображение консоли ошибок в браузере Internet Explorer» главы 1) описывает эту же ошибку как *Expected ')'* (*Ожидается ')'*); консоль ошибок браузера Chrome отобразит ошибку *SyntaxError: Unexpected token ;*; а программа Safari (см. раздел «Доступ к консоли ошибок в браузере Safari» главы 1) выдаст еще менее обнадеживающее сообщение: *Syntax Error: Expected token ')'*. Как уже говорилось, программа Firefox предоставляет наиболее понятные сообщения об ошибках, так что этот браузер лучше всего подходит для выяснения причины неработоспособности сценария (рис. 17.1).

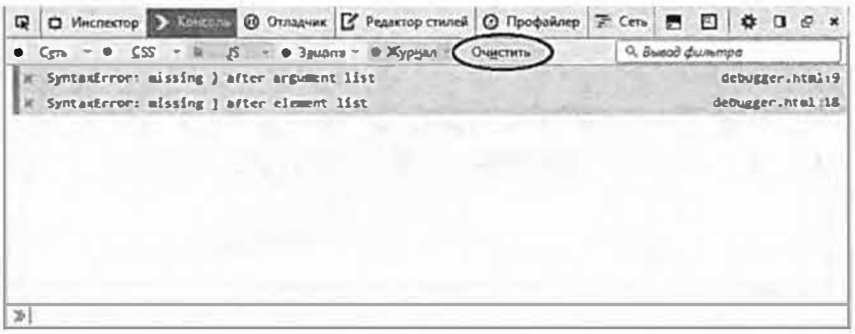

Рис. 17.1. Консоль ошибок программы Firefox перечисляет все ошибки JavaScript, которые выявил браузер. Чтобы отобразить консоль JavaScript, выберите в меню **Инструменты** ⇒ **Веб-разработка** (Tools ⇒ Web Developer) команду **Веб-консоль** (Web Console) (**Ctrl+Shift+K**) (Windows). В операционной системе OS X выберите аналогичную команду или нажмите сочетание клавиш **⌘+⌥+K** (OS X). Поскольку в консоли перечислены ошибки на всех страницах, вам часто понадобится удалять их с помощью кнопки **Очистить** (Clear) (выделена на рисунке)

Синтаксическую ошибку в коде `alert('привет';` найти несложно. А вот когда приходится использовать несколько вложенных скобок, очень легко пропустить закрывающий знак и не заметить эту оплошность. Например:

```
if ((x>0) && (y<10) {
// некое действие
}
```

Здесь пропущена последняя закрывающая круглая скобка после управляющей инструкции, идущая сразу за частью `(y<10)`. На самом деле первая строка должна выглядеть так: `if ((x>o) && (y<10)) {`. Браузер Firefox дает наиболее четкое описание проблемы: *Missing) after condition* (*Пропущена) после условия*). Табл. 17.1 содержит список сообщений консоли ошибок программы Firefox.

Табл. 17.1. Консоль ошибок браузера Firefox предоставляет наиболее ясное описание синтаксических ошибок

Сообщение об ошибке Firefox	Описание

Unterminated string literal (Незавершенный строковый литерал)	Пропущена открывающая или закрывающая кавычка:
	`var name = Жанна';`
	Ошибка будет выдана и в случае использования разных видов кавычек:
	`var name = 'Жанна";`

Missing) after arguments list (Пропущена) после списка аргументов)	Пропущена закрывающая круглая скобка при вызове функции или метода: `alert('привет' ;`
Missing) after condition (Пропущена) после условия)	Пропущена закрывающая круглая скобка внутри управляющей инструкции: `if (x==0`
Missing (before condition (Пропущена (перед условием)	Пропущена открывающая круглая скобка внутри управляющей инструкции: `if x==0)`
Missing } in compound statement (Пропущена } в составной инструкции)	Пропущена закрывающая фигурная скобка как часть управляющей инструкции: `if (score == 0) { alert('игра окончена'); //в этой строке пропущена }`
Missing } after property list (Пропущена } после списка свойств)	Пропущена закрывающая фигурная скобка для объекта JavaScript: `var x = {fName: 'иван', lName: 'иванов' // в этой строке пропущена }`
Syntax Error (Синтаксическая ошибка)	Общая проблема, мешающая интерпретатору JavaScript прочесть сценарий
Missing ; before statement (Пропущена ; перед инструкцией)	Попытка выполнить две инструкции, не разделенные точкой с запятой. Это сообщение также появляется, если вы неверно разместили вложенные кавычки: `var name ='Жанна д'Арк';`
Missing variable name (Пропущено имя переменной)	Попытка применить зарезервированное слово языка JavaScript (см. раздел «Создание переменной» главы 2) в качестве имени переменной: `var if="Syntax error.";`

Вы столкнетесь с синтаксической ошибкой, если забудете о второй кавычке. Например, код `alert('привет);` вызывает ошибку, поскольку

ку пропущена вторая кавычка: `alert('привет');`. Если вы не поставите обе кавычки, браузер Firefox сообщит о «незакрытом строковом литерале», тогда как программа Internet Explorer — о «незавершенной строковой константе»; браузеры Safari и Chrome выдадут сообщение: «Синтаксическая ошибка: Unexpected token ILLEGAL».

Фигурные скобки также используются парами и применяются в управляющих инструкциях (см. раздел «Интеллектуальная реакция программы» главы 3), циклах (см. раздел «Работа с повторяющимися задачами с использованием циклов» главы 3), при создании функций (см. раздел «Функции: многократное использование кода» главы 3) и данных в формате JSON (см. раздел «Формат JSON» главы 13):

```
if (score==0) {
alert('игра окончена');
```

Здесь пропущена закрывающая скобка `}`, поэтому возникнет синтаксическая ошибка.

Один из способов решить данную проблему — всегда набирать фигурные скобки до ввода программного кода. Например, вы хотите ввести следующий код:

```
if ((name=='вова') && (score==0)) {
alert('Вы проиграли, но у вас отличное имя');
}
```

ПРИМЕЧАНИЕ

Многие хорошие текстовые редакторы предусматривают функцию подсвечивания синтаксиса, благодаря которой выделяются пары круглых, квадратных и фигурных скобок. Некоторые редакторы даже могут указать на недостающие знаки пунктуации, что позволяет вам быстро исправлять ошибки.

Начните с ввода внешних элементов, создайте базовый каркас для управляющей инструкции:

```
if () {
}
```

Сейчас кода не много, поэтому нетрудно заметить, если пропущен какой-нибудь знак. Теперь фрагмент за фрагментом вводите код. Такой

способ подойдет при создании сложного литерала объекта JavaScript наподобие того, что был использован для настройки плагина Validation (см. раздел «Проверка формы» главы 8), или объекта JSON, описанного в разделе «Формат JSON» главы 13. Начните с основы:

```
var options = {
};
```

Теперь добавьте структуру:

```
var options = {
rules : {
},
messages : {
}
};
```

Затем завершите объект:

```
var options = {
 rules : {
name : 'required',
email: 'email'
 },
 messages : {
name : 'Введите свое имя',
email: 'Введите адрес электронной почты.'
 }
};
```

Этот подход дает вам возможность проверять работу на разных этапах и существенно облегчает распознавание любых ошибок пунктуации. Консоль браузера Firefox (см. раздел «Консоль JavaScript в браузере Firefox» главы 1) предоставляет самые ясные описания синтаксических ошибок. Если сценарий не работает, откройте его в программе Firefox и просмотрите окно консоли. В табл. 17.1 приведены некоторые из наиболее распространенных ошибок и их описания.

Кавычки

Кавычки нередко являются проблемой для начинающих программистов. Они служат для создания строк из букв и других символов

(см. раздел «Строки» главы 2), которые используются в качестве сообщений на странице или переменных в программе. Язык JavaScript, подобно другим языкам программирования, разрешает применять или *двойные*, или *одинарные* кавычки. Так, например, код

```
var name="Жанна";
```

эквивалентен коду

```
var name='Жанна';
```

Как указывалось в предыдущем разделе, вы должны включать в код открывающие и закрывающие кавычки, чтобы не получить сообщение *Unterminated string literal* (*Незавершенный строковый литерал*) в программе Firefox (остальные браузеры также откажутся выполнять ваш сценарий). Кроме того, как вы уже знаете, в каждой паре должны использоваться кавычки одинакового типа, иными словами, либо обе одинарные, либо обе двойные. Например, код `var name='Жанна"` сгенерирует ошибку.

Еще одна распространенная проблема возникает при использовании кавычек внутри строки. Например, легко сделать ошибку, набрав следующий код:

```
var name ='Жанна д'Арк';
```

Обратите внимание на кавычку в слове `д'Арк`. Интерпретатор JavaScript воспринимает ее как закрывающую, поэтому видит такую строку: `var name ='Жанна д'`, а все остальное трактует как ошибку. В консоли ошибок программы Firefox вы увидите сообщение *Missing ; before statement* (*Пропущена ; перед инструкцией*), поскольку браузер решил, что вторая кавычка завершает простую инструкцию JavaScript, а то, что следует далее, является второй инструкцией.

Чтобы исправить эту ошибку, воспользуйтесь одним из следующих способов. Во-первых, вы можете комбинировать одинарные и двойные кавычки. Иначе говоря, заключать в двойные кавычки строку, содержащую одинарные кавычки, или наоборот — заключить в одинарные кавычки строку, содержащую двойные кавычки. Например, устранить ошибку можно так:

```
var name ="Жанна д'Арк";
```

Или в случае наличия в строке двойных кавычек:

```
var message='Он сказал: "Нет проблем."';
```

Во-вторых, вы можете использовать знак переключения кода. Эта тема была подробно рассмотрена в тексте «Кавычки в строках» главы 2, но напомним: чтобы заставить интерпретатор JavaScript воспринимать кавычку в строке буквально, перед ней ставится знак \:

```
var name ="Жанна д\'Арк";
```

Интерпретатор JavaScript воспринимает сочетание символов \' как одинарную кавычку, а не как символ начала и конца строки.

УСКОРЯЕМ РАБОТУ

Типы ошибок

Выделяют три основные категории ошибок, с которыми можно столкнуться в процессе программирования на языке JavaScript. Некоторые из них обнаруживаются сразу, тогда как другие проявляют себя только при выполнении сценария.

- **Синтаксические ошибки.** В сущности, являются грамматическими ошибками, которые вынуждают интерпретатор JavaScript браузера поднять руки вверх и сказать: «Сдаюсь». К этой категории относятся любые случаи пропуска кавычек или скобок. Браузер обнаружит такую ошибку сразу же, как начнет читать сценарий, так что программа никогда не запустится. Сообщение о синтаксической ошибке всегда отображается в консоли ошибок браузера.
- **Ошибки на этапе выполнения.** Ошибки могут возникнуть и после того как браузер успешно завершил чтение сценария, а интерпретатор JavaScript обработал его — это так называемые ошибки *runtime*. Например, вы определили переменную `message` в начале сценария; позже добавили событие щелчка для изображения, чтобы после щелчка кнопкой мыши по рисунку появлялось окно оповещения. Допустим, код для подобного сообщения выглядит так: `alert(MESSAGE);`. Этот синтаксис не имеет ошибок, но он вызывает переменную `MESSAGE` вместо `message`. Как говорится в разделе «Чувствительность к регистру» далее в главе, язык JavaScript учитывает регистр, поэтому слова `MESSAGE` и `message` обозначают две разные переменные. Когда посетитель щелкает по изображению, интерпретатор JavaScript ищет переменную `MESSAGE` (которой не существует) и генерирует ошибку на этапе выполнения.

Другая распространенная ошибка случается при попытке доступа к элементу на странице, который либо не существует, либо еще не прочтен браузером. Описание jQuery-функции `$(document).ready()` см. в разделе «Больше концепций для событий jQuery» главы 5.

- **Логические ошибки.** Иногда даже выполняющийся сценарий не приносит ожидаемых результатов. Например, у вас есть инструкция *if/else* (см. раздел «Больше концепций для событий jQuery» главы 5), которая выполняет шаг A, если условие истинно, или шаг B, если оно ложно. К сожалению, программа, похоже, никогда не перейдет к шагу B, даже если вы уверены, что условие ложно. Эта ошибка происходит из-за некорректного применения операции равенства (см. раздел «Основные математические операции» главы 2). С точки зрения интерпретатора JavaScript все технически корректно, но вы допустили ошибку в логике кода, что не позволит сценарию работать по плану.

Другим примером логической ошибки является бесконечный цикл, то есть участок кода, который выполняется *вечно*, а это обычно приводит к зависанию программы и иногда даже к отказу браузера. Вот пример бесконечного цикла:

```
for (var i=1; i>0; i++) {
// данный код будет выполняться вечно
}
```

В сущности, этот цикл будет выполняться до тех пор, пока тестовое условие (`i>0`) остается истинным. Поскольку начальным значением переменной `i` является 1 (`var i=1`), и каждый раз при выполнении цикла значение переменной `i` возрастает на 1 (`i++`), значение переменной `i` всегда будет превышать 1. Иначе говоря, выполнение цикла никогда не прекратится (сведения о циклах *for* см. в разделе «Циклы `for`» главы 3).

Логические ошибки обнаружить труднее всего. Однако методы отладки, изложенные в разделе «Отладка с помощью веб-консоли» далее в главе, должны помочь вам в распознавании почти всех проблем, с которыми можно столкнуться.

Использование зарезервированных слов

В разделе «Создание переменной» главы 2 сказано, что язык JavaScript имеет много слов, которые зарезервированы для использования с целью обозначения базовых элементов самого языка. К их числу относятся слова синтаксиса, например, `if`, `do`, `for` и `while`, а также слова, относящиеся к объекту *browser*: `alert`, `location`, `window`, `document` и т. д. Эти слова не могут использоваться в качестве имен переменных.

Например, следующий код сгенерирует синтаксическую ошибку:

```
var if = "Это не сработает.";
```

Поскольку слово `if` служит для создания управляющих инструкций, как в `if(x==0)`, вы не можете назвать этим словом переменную. Некоторые браузеры, тем не менее, не генерируют ошибку, если для обозначения переменных используются слова, зарезервированные для объектной модели браузера. Например, взгляните на следующий код:

```
var document='Здесь происходит что-то странное.';
alert(document);
```

Браузеры Firefox, Safari и Opera не сгенерируют ошибки, а вместо этого отобразят окно оповещения с сообщением `[object HTMLDocument]`, которое относится к самому HTML-документу. Другими словами, эти браузеры не позволят вам заменить объект документа строкой.

Одинарные знаки равенства в управляющих инструкциях

Управляющие инструкции (см. раздел «Управляющие инструкции» главы 3) позволяют программе реагировать разными способами, в зависимости от значения переменной, статуса элемента на странице или других условий в сценарии. Например, управляющая инструкция может показать изображение, если оно скрыто, или скрыть его, если оно отображено. Однако управляющие инструкции имеют смысл, только если условие может принимать значения `true` или `false`. К сожалению, довольно легко создать управляющую инструкцию, которая является истинным всегда:

```
if (score=0) {
alert('игра окончена');
}
```

Этот код был предназначен для проверки значения, хранящегося в переменной `score`. Если значение равно 0, должно появляться сообщение `'игра окончена'`. Однако в данном случае это сообщение будет появляться *всегда*, независимо от значения переменной `score` перед управляющей инструкцией. Причина — знак равенства, воспринимаемый как операция присваивания. Таким образом, код `score=100` сохраняет значение 100 в переменной `score`. Интерпретатор JavaScript воспринимает операцию присваивания как `true`, так что наш код не только все время выводит указанное сообщение, но и изменяет любое значение переменной `score` на 100.

Во избежание этой ошибки используйте *двойной* знак равенства при сравнении двух значений:

```
if (score==100) {
alert('Вы выиграли!');
}
```

Чувствительность к регистру

Помните, что язык JavaScript учитывает регистр, то есть интерпретатор отслеживает не просто буквы в именах переменных, функций, методов и ключевых слов, а также и их регистр. Так, код `alert('эй')` для интерпретатора JavaScript не то же самое, что `ALERT('эй')`. Код `alert('эй')` вызывает встроенную команду браузера `alert()`, тогда как `ALERT('эй')` пытается вызвать пользовательскую функцию `ALERT()`.

Вы также можете столкнуться с проблемой при использовании таких методов выделения DOM, как: `getElementsByTagName()` или `getElementById()`, поскольку они включают как прописные, так и строчные буквы (еще один повод для использования библиотеки jQuery). Аналогично, включая в имена переменных и функций буквы, набранные разным регистром, вы рискуете столкнуться с теми же проблемами.

Если вы видите сообщение *"x is not defined" (x не определен)* (где *x* — это имя вашей переменной, функции или метода), то причиной может оказаться несовпадение регистров.

Некорректный путь к внешнему файлу JavaScript

Еще одной распространенной ошибкой является некорректная ссылка на внешний файл JavaScript. В разделе «Внешние файлы JavaScript» главы 1 уже обсуждалось, как присоединять внешний JavaScript-файл к веб-странице. В общем, вы используете свойство `src` элемента `script` для указания на файл. Таким образом, вы добавляете элемент `script` в раздел заголовка документа:

```
<script src="site_js.js"></script>
```

Свойство `src` работает как свойство ссылки `href` — оно определяет путь к файлу JavaScript. Как отмечалось в тексте «Типы URL-адресов»

главы 1, есть три способа указать на файл: с помощью абсолютных ссылок (*http://www.site.com/site_js.js*), ссылок относительно корневого каталога (*/site_js.js*) и ссылок относительно документа (*site_js.js*).

Путь относительно документа описывает то, как браузер добирается от текущей страницы (веб-страницы) до определенного файла. Ссылки относительно документа используются часто, поскольку позволяют тестировать веб-страницу и файл JavaScript непосредственно на локальном компьютере. При использовании ссылок относительно корневого каталога необходимо установить на компьютер веб-сервер для тестирования ваших страниц (или переместить документы на веб-сервер).

Подробнее узнать о работе ссылок можно в тексте «Типы URL-адресов» главы 1. Но, в сущности, если вы обнаружили, что сценарий не работает, а вы используете внешние файлы JavaScript, дважды проверьте, правильно ли задан путь к ним.

СОВЕТ

Если вы используете библиотеку jQuery и получаете сообщение *$ is not defined* (*$ не определен*) в консоли ошибок браузера Firefox, вы, вероятно, некорректно сослались на файл *jquery.js* (см. врезку «Типы URL-адресов» главы 1).

Некорректные пути внутри внешнего файла JavaScript

Другая проблема с путями к файлам возникает при использовании путей, относящихся к документам во внешнем файле JavaScript. Например, вы создали сценарий, отображающий изображения на странице (вроде сценария слайд-шоу или программы отображения случайно выбранной картинки). Если для указания на изображения в сценарии используются пути относительно документа, то вы можете столкнуться с проблемой, если поместили сценарий во внешний файл JavaScript, и вот почему: когда внешний файл JavaScript загружается на веб-страницу, его область видимости для путей относительно документа — это сама веб-страница. Таким образом, любые пути относительно документа, которые вы включаете в файл JavaScript, должны относиться к *веб-странице*, а не к файлу JavaScript.

Рассмотрим простой пример для иллюстрации этой проблемы. На рис. 17.2 представлена структура очень простого сайта. Он имеет две страницы (*page.html* и *about.html*), четыре папки (*libs*, *images*, *pages*

и *about*), внешний файл JavaScript (*site_js.js* внутри папки *libs*) и картинку (*photo.jpg* в папке *images*). Допустим, файл *site_js.js* ссылается на файл *photo.jpg*, например, для предварительной загрузки картинки (см. раздел «Предварительная загрузка изображений» главы 7) или для ее динамического отображения на веб-странице.

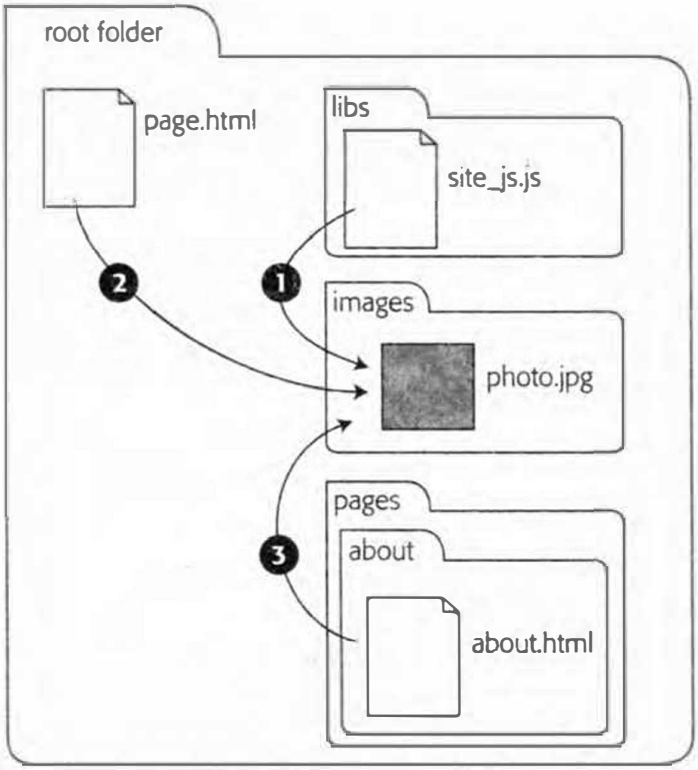

Рис. 17.2. Пути относительно документа зависят от местонахождения как файла, с которого начинается путь, так и конечного файла. Например, путь относительно документа от файла *site_js.js* к файлу *photo.jpg* (*1*) таков: *../images/photo.jpg*; путь к этому же файлу из файла *page.html* (*2*) следующий: *images/photo.jpg*; от файла *about.html* (*3*) — *../../images/photo.jpg*

С точки зрения файла *site_js.js*, путь относительно документа к файлу *photo.jpg* — это *../images/photo.jpg* (*1* на рис. 17.2). Путь приказывает браузеру выйти из папки *libs* (*../*), войти в папку *images* (*images/*) и выбрать файл *photo.jpg*. Однако, с точки зрения файла *page.html*, путь к файлу *photo.jpg* (*2* на рис. 17.2) — это просто *images/photo.jpg*. Иными словами, для этих двух файлов путь к одной и той же картинке неодинаков.

Если вы хотите использовать сценарий *site_js.js* внутри файла *page.html*, тогда вам нужно использовать путь *2* в файле *site_js.js* для ссылки на изображение *photo.jpg* (то есть вы выбираете путь относительно файла *page.html*). Вы также не можете использовать файл *site_js.js* на веб-

странице, размещенной в другой папке на вашем сайте, поскольку путь относительно документа для этого файла будет иным (*3* на рис. 17.2).

Существует несколько путей решения этой проблемы. Во-первых, вы можете никогда и не узнать о подобных затруднениях, если не станете использовать пути к другим файлам внутри ваших файлов JavaScript. Но если это необходимо, используйте пути относительно корневого каталога (см. текст «Типы URL-адресов» главы 1), которые одинаковы для любой страницы сайта. В качестве альтернативного способа можно определять путь к файлам внутри каждой веб-страницы. Например, создайте ссылку на внешний файл JavaScript (см. раздел «Внешние файлы JavaScript» главы 1), а затем на каждой веб-странице определите переменные, содержащие пути относительно документа с текущей страницы к нужному файлу.

Наконец, можно применить подход, используемый при создании слайд-шоу в разделе «Фотогалерея с эффектами на практике» главы 7. Пути внедрены в ссылки на странице — программа JavaScript просто извлекает пути из HTML-кода. Если эти пути работают в HTML-коде, они будут работать и в сценарии.

Пропадающие переменные и функции

Иногда вам может встретиться ошибка *"x is not defined"* (*x не определен*), где *x* является именем переменной или функции, которую вы пытаетесь вызвать. Иногда причина — в простой ошибке при вводе имени переменной или функции или в использовании неверного регистра. Однако если проверка ясно показывает, что переменная или функция в вашем сценарии задана, тогда, возможно, вы столкнулись с проблемой «области видимости».

Область видимости переменной или функции подробно обсуждалась в разделе «Предупреждение конфликта переменных» главы 3, но здесь следует сказать, что, если переменная определена внутри функции, она доступна только в пределах этой функции (или других функций, заданных внутри данной функции). Вот простой пример:

```
1 function sayName(name) {
2     var message = 'Ваше имя ' + name;
3 }
4 sayName();
5 alert(message); // Ошибка: переменная не определена
```

Переменная `message` задана внутри функции `sayName()`, так что существует она только внутри этой функции. Вне ее пределов переменная `message` не существует, поэтому попытка сценария обратиться к переменной вне функции генерирует ошибку (строка 5).

Вы можете также встретиться с этой же ошибкой при использовании библиотеки jQuery. В разделе «Больше концепций для событий jQuery» главы 5 вы прочли о важности функции `$(document).ready()` для библиотеки jQuery. Все компоненты внутри этой функции выполняются только после загрузки HTML-кода страницы. Вы столкнетесь с проблемой, когда попытаетесь задать переменные или функции внутри функции `$(document).ready()` и обратиться к ним за пределами этой функции:

```
$(document).ready(function() {
var msg = 'привет';
});
alert(msg); // Ошибка: переменная msg не определена
```

Таким образом, при использовании библиотеки jQuery убедитесь, что весь программный код находится внутри функции `$(document).ready()`:

```
$(document).ready(function() {
var msg = 'привет';
alert(msg); // переменная msg доступна
});
```

УСКОРЯЕМ РАБОТУ

Как уменьшить вероятность ошибок при верстке кода

Лучше всего распознавать и предотвращать программные ошибки как можно раньше. Причину ошибок бывает действительно весьма трудно отследить, если вы пишите сценарий, состоящий из 300 строк, прежде чем начать тестирование программы в веб-браузере. В этой связи есть два важных совета.

- **Создавайте сценарий небольшими частями.** Как вы, вероятно, уже убедились, программу JavaScript бывает нелегко прочесть из-за всех этих `}`, `)`, `'`, `if`, `else`, функций и т. д. Не пытайтесь написать сценарий за одни присест (разве только если у вас действительно хорошая квалификация, сценарий короток или вам

всегда сопутствует удача). Существует так много возможностей для совершения ошибки при программировании, что разумнее писать сценарий постепенно.

Скажем, вы хотите отобразить количество знаков, введенных в окно «Комментарии», рядом с полем ввода. Иначе говоря, когда посетитель вводит данные в поле формы, рядом с полем появляется общее количество набранных символов. (На некоторых сайтах такая возможность используется с целью ограничить число вводимых знаков, например, 300 символами.) Эта задача легко решается с помощью кода JavaScript, но включает ряд этапов: реакцию на событие нажатия клавиши (когда посетитель вводит в поле символ), считывание значения этого поля, подсчет числа знаков в поле ввода и вывод этого числа на страницу. Вы можете попытаться написать сценарий одним разом, но можно сначала создать код для шага 1 (реакция на событие нажатия клавиши), затем протестировать его непосредственно в браузере (с помощью команды `alert()` или функции `console.log()`, описанной далее в главе). Если все работает, можно перейти к шагу 2, затем проверить его и т. д.

По мере накопления опыта вам уже не нужно будет проверять столь короткие фрагменты. Вы сможете написать код для нескольких шагов сразу, после чего проверить его.

- **Тестируйте часто.** Проверять сценарий в браузере следует регулярно. Как минимум, это следует делать по завершении каждого раздела программы. Кроме того, сценарий желательно тестировать в разных браузерах — предпочтительно в программах Internet Explorer версии 8 и выше, в новейших версиях Safari, Firefox и Chrome, а также в любом другом браузере, которым, как вам кажется, могут пользоваться посетители вашего сайта.

Отладка с помощью веб-консоли

Если вы еще не знакомы с веб-консолью своего браузера, значит, до сих пор не имели удовольствия увидеть в действии один из самых лучших инструментов, какой только может быть в распоряжении веб-дизайнера. Все основные браузеры предусматривают встроенную веб-консоль JavaScript, которая может существенно помочь вам в отладке кодов HTML, CSS и JavaScript.

Открытие веб-консоли

Чтобы использовать консоль, откройте веб-страницу в браузере. Затем используйте один из следующих методов:

- **Google Chrome.** Чтобы открыть консоль, щелкните по значку **Настройка** (Customize) (который выделен на рис. 17.3), выберите в меню **Инструменты** (Tools) команду **Консоль JavaScript** (JavaScript Console) или используйте сочетание клавиш **Ctrl+Shift+J** (Windows) или **⌘+⌥+J** (OS X).
- **Internet Explorer.** Нажмите клавишу **F12**, чтобы открыть панель **Инструменты разработчика** (Developer Tools). Щелкните по вкладке **Консоль** (Console) для вызова консоли JavaScript.
- **Firefox.** Чтобы отобразить консоль JavaScript, выберите в меню **Инструменты ⇒ Веб-разработка** (Tools ⇒ Web Developer) команду **Веб-консоль** (Web Console) (**Ctrl+Shift+K**) (Windows). В операционной системе OS X выберите аналогичную команду или нажмите сочетание клавиш **⌘+⌥+K** (OS X).
- **Safari.** Консоль ошибок в браузере Safari доступна с помощью команды меню **Разработка ⇒ Показать консоль ошибок** (Develop ⇒ Show Error Console) (или воспользуйтесь сочетанием клавиш **⌥+⌘+C**, а в операционной системе Windows — **Ctrl+Alt+C**). Однако следует помнить, что меню **Разработка** (Develop) в браузере Safari обычно отключено, поэтому, прежде чем перейти к консоли ошибок JavaScript, необходимо выполнить некоторые операции. Чтобы отобразить меню **Разработка** (Develop), вам нужно вызвать диалоговое окно **Настройки** (Preferences). В операционной системе OS X выберите в меню **Safari** пункт **Настройки** (Preferences). В операционной системе Windows щелкните по значку основных настроек в верхнем правом углу браузера и в появившемся меню выберите пункт **Настройки** (Preferences). В открывшемся диалоговом окне **Настройки** (Preferences) щелкните по кнопке **Дополнения** (Advanced). Установите флажок **Показывать меню «Разработка» в строке меню** (Show Develop menu in menu bar) и закройте окно **Настройки** (Preferences). Когда вы перезапустите браузер Safari, пункт **Разработка** (Develop) появится между меню **Закладки** (Bookmarks) и **Окно** (Window) в верхней части экрана. Выберите команду меню **Разработка ⇒ Показать консоль ошибок** (Develop ⇒ Show Error Console), чтобы открыть консоль ошибок (см. рис. 1.9).

Обзор ошибок с помощью консоли

В консоли перечислены все ошибки, обнаруженные в вашем коде JavaScript. Сначала там указывается первая синтаксическая ошибка,

возникшая на странице. Как вы помните из раздела «Отслеживание ошибок» главы 1, синтаксические ошибки подобны грамматическим ошибкам. Когда интерпретатор JavaScript браузера встречает синтаксическую ошибку, он просто прекращает искать другие ошибки. Он сообщает вам об этой ошибке, но об остальных синтаксических ошибках на странице вы не узнаете до тех пор, пока не устраните первую.

После исправления синтаксических ошибок вы можете начать сталкиваться с ошибками на этапе выполнения (см. врезку «Типы ошибок» ранее в этой главе), то есть с ошибками, о которых браузер дает вам знать во время выполнения программы. Например, попытка получить доступ к несуществующей переменной может вызвать такую ошибку. Консоль должна стать вашей первой линией обороны при отслеживании ошибок в коде (рис. 17.3).

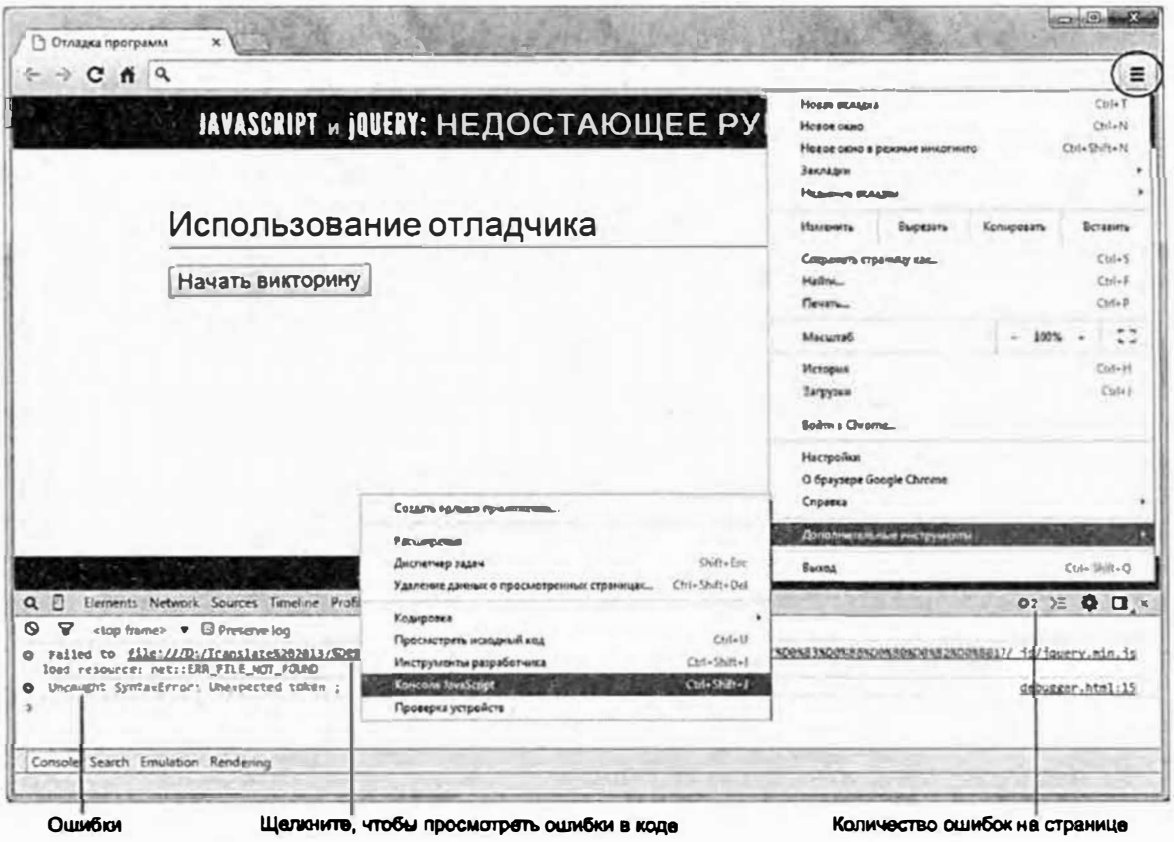

Рис. 17.3. В консоли JavaScript перечислены все ошибки JavaScript, обнаруженные браузером на текущей странице. Чтобы открыть консоль, щелкните по значку **Настройка** (Customize) (выделен на рисунке), выберите в меню **Инструменты** (Tools) команду **Консоль JavaScript** (JavaScript Console). Чтобы увидеть конкретную строку кода, в которой найдена ошибка, щелкните по фрагменту кода под ошибкой. Браузер переключится с вкладки **Консоль** (Console) на вкладку **Ресурсы** (Sources) и выделит строку, содержащую ошибку

Использование функции `console.log()` для отслеживания выполнения сценария

Когда сценарий начинает выполняться, он становится чем-то сродни «черному ящику». Вы не знаете, что происходит внутри сценария, а только видите конечный результат, например, сообщение на странице, всплывающее окно и т. д. Вы не всегда можете удостовериться, что данный цикл выполняется корректно или значение данной переменной появляется вовремя.

Разработчики JavaScript уже давно применяют метод `alert()` (см. раздел «Ваша первая программа на языке JavaScript» главы 1) для отображения окна с текущим значением переменной. Например, если вы хотите знать, какое значение хранится в переменной `elementName` во время выполнения цикла, вы можете вставить в цикл команду оповещения: `alert(elementName);`. Это один из способов заглянуть в «черный ящик» сценария. Однако окно оповещения довольно «навязчиво»: вам придется щелкать по нему, чтобы его закрыть, и если цикл должен выполняться, скажем, 20 раз, то вам придется закрыть 20 окон.

Консоль JavaScript предлагает более приемлемый способ заглянуть внутрь вашей программы. Эта консоль не только перечисляет ошибки (см. предыдущий раздел), но и может использоваться для вывода сообщений программы. Функция `console.log()` работает подобно функции `document.write()` (см. раздел «Написание текста на веб-странице» главы 1), но вместо отображения сообщений на веб-странице, она показывает их в консоли.

СОВЕТ

Все современные браузеры поддерживают функцию `console.log()`. Так что вы можете использовать ее в консолях Chrome, Safari, Internet Explorer и Opera.

Например, вы могли бы отобразить текущее значение переменной `elementName` в консоли с помощью следующего кода:

```
console.log(elementName);
```

В отличие от метода `alert()`, данный способ не прервет выполнение программы — вы просто увидите сообщение в окне консоли.

Сделать сообщение консоли более понятным можно с помощью включения строки дополнительного текста. Например, если у вас есть

переменная name, и вы хотите определить, какое значение в ней содержится на некотором этапе выполнения программы, можно воспользоваться функцией console.log() так:

```
console.log(name);
```

Но если вы хотите предварить имя сообщением, можно ввести следующее:

```
console.log('Имя пользователя: ', name);
```

Этот фрагмент кода отобразит в консоли одну строку, содержащую слова 'Имя пользователя: ' и значение, хранящееся в переменной name.

Но что, если вам необходимо поместить значения переменных внутри строки? Например, вы отслеживаете количество очков, набранных пользователем.

Вы хотите отобразить в консоли ясное сообщение, например, «Женя набрала 50 очков». Другими словами, вы хотите поместить имя в начале строки, а количество очков — в середине.

Для этого вы передаете функции console.log() строку, содержащую сочетание символов %s в качестве местозаполнителя для каждой из переменных, которые вы хотите поместить в строку. После этого сочетания должна следовать запятая, имя переменной, еще одна запятая и еще одно имя переменной. Например:

```
console.log('%s набрал %s', name, score);
```

Сочетание %s означает: «замени меня значением переменной». Другими словами, первое сочетание %s будет заменена значением переменной name, а второе сочетание %s — значением переменной score.

Функция log() предназначена просто для получения некоторой информации о вашем сценарии на стадии его *разработки*. Когда программа завершена и успешно прошла проверку, вам следует удалить весь код функции console.log() из вашего сценария.

Использование консоли на практике

В данном руководстве вы узнаете, как использовать функцию console.log(), чтобы увидеть, что происходит внутри вашей программы. Вы создадите сценарий, который покажет число знаков, введенных в текстовое поле формы.

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Для выполнения данного руководства использовался браузер Google Chrome. Вы можете воспользоваться своим любимым браузером, однако его поведение может отличаться от поведения программы Chrome.

1. **Откройте файл *17_01.html* в текстовом редакторе.**

   Этот сценарий использует библиотеку jQuery. Внешний файл jQuery уже присоединен, также присутствуют открывающий и закрывающий теги элемента `script`. Начнем с добавления jQuery-функции `$(document).ready()`.

2. **Между тегами элемента `script` в верхней части страницы добавьте следующий код, выделенный полужирным шрифтом:**

```
<script>
$(document).ready(function( ) {
}); // конец ready
</script>
```

   Функция `$(document).ready()`, с которой вы познакомились в разделе «Больше концепций для событий jQuery» главы 5, заставляет браузер загружать весь код страницы, прежде чем начать выполнение программы JavaScript. Сначала используем функцию `console.log()`, чтобы отобразить сообщение о том, что сценарий выполнил функцию `.ready()`.

3. **Добавьте в сценарий выделенный полужирным шрифтом код:**

```
<script>
$(document).ready(function() {
console.log('READY');
}); // конец ready
</script>
```

   Функция `console.log()` выполняется, в какое бы место в сценарии вы бы ее ни поместили. Другими словами, после загрузки

HTML-кода страницы (а именно этого ожидает функция `ready()`) браузер напишет *«READY»* в своей консоли. Добавление функции `ready()` является достаточно обычным ходом, поэтому вы можете не всегда использовать здесь функцию `console.log()`, но в данном руководстве мы сделаем это, чтобы посмотреть, как она работает. Собственно говоря, вам придется добавить немало сообщений на эту страницу, чтобы приобрести навык использования функции `console.log()`.

4. **Сохраните файл и откройте его в браузере Chrome. Если консоль еще не открыта, нажмите сочетание клавиш Ctrl+Shift+J (ОС Windows) или ⌘+⌥+J (OS X).**

   В окне консоли должно появиться слово READY (обведено на рис. 17.4). Создаваемый сценарий должен показывать число знаков, введенных в поле формы, каждый раз, когда посетитель печатает символ. Для этого добавим к текстовому полю событие `keyup` (см. раздел «События клавиатуры» главы 5). На каждом шаге построения сценария вы будете добавлять функцию `console.log()`, чтобы быть в курсе происходящего.

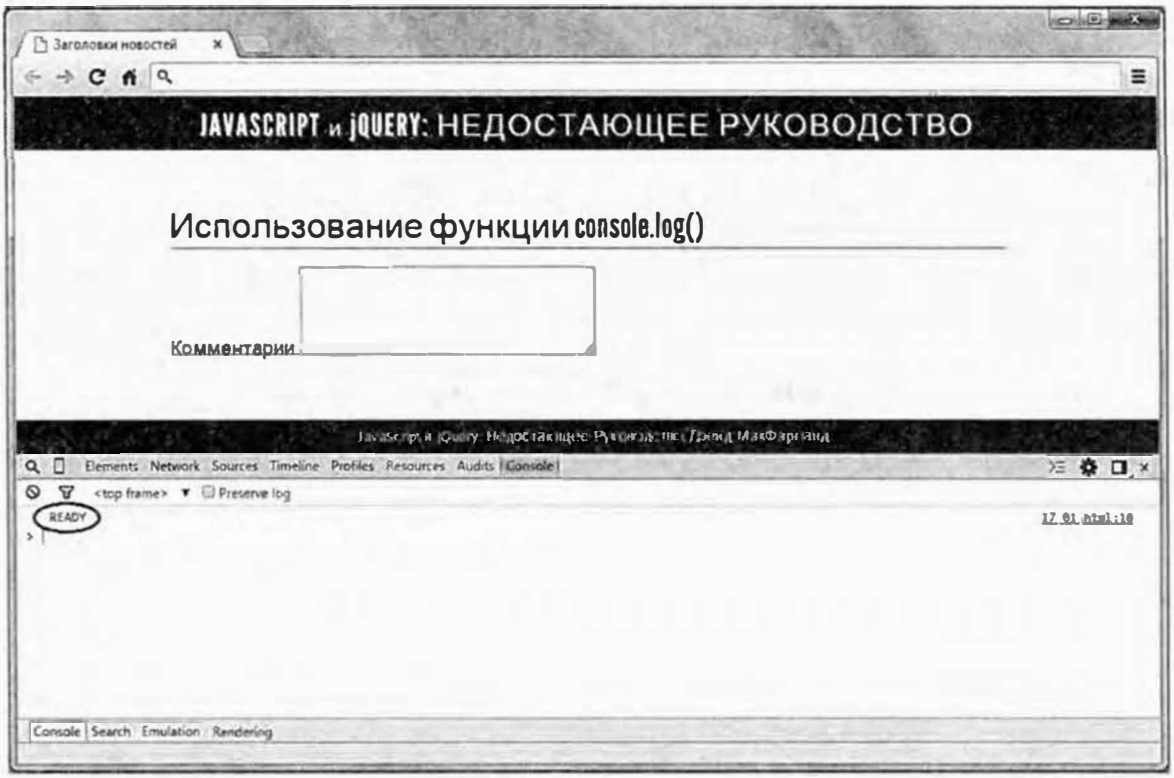

Рис. 17.4. Вы можете использовать консоль браузера для написания секретных сообщений. Однако она больше подходит для отображения заметок в процессе отладки программы. В этом случае простое сообщение «Ready» показывает, что программа успешно выполнила jQuery-функцию `$(document).ready()`

5. **После кода, введенного на шаге 3, добавьте следующий фрагмент:**

```
$('#comments').keyup(function() {
   console.log('Событие: keyup');
}); // конец keyup
```

Убедитесь в том, то вы поместили этот код внутри функции `$(document).ready()`.

Элемент `textarea` на этой странице имеет идентификатор `comments`, так что выбрать данный элемент можно с помощью селектора jQuery (`$('#comments')`), а затем добавить функцию к событию `keyup` (чтобы освежить свои знания о добавлении событий, обратитесь к разделу «Использование событий: подход jQuery» главы 5). В данном случае функция `console.log()` просто отображает сообщение о статусе в консоли JavaScript каждый раз, когда запускается событие `keyup`. Это и есть простой способ узнать, функция события действительно выполняется или что-то этому препятствует.

Сохраните страницу, обновите ее в браузере и введите несколько символов в текстовое поле формы. Убедитесь, что консоль JavaScript открыта и содержит строку *«Событие:keyup»*, а также число слева от сообщения (в некоторых случаях — справа). Это число показывает, сколько раз было отображено сообщение `console.log()`.

Теперь, когда событие `keyup` работает, извлечем содержимое текстового поля и сохраним его в переменной. Чтобы быть уверенными, что мы получаем требуемую информацию, покажем содержимое переменной в консоли.

6. **Добавьте строки 3 и 4 в код, введенный на шаге 5:**

```
1 $('#comments').keyup(function() {
2   console.log('Событие: keyup');
3   var text = $(this).val();
4   console.log('Содержимое комментария: %s',text);
5 }); // конец keyup
```

Строка 3 извлекает значение из текстового поля и сохраняет его в переменной `text` (см. раздел «Получение и установка значений элементов форм» главы 8 для получения сведений об извлечении данных из поля формы). Строка 4 отображает сообщение в консоли. В данном

случае она объединяет строку 'Содержимое комментария:' и значение, хранящееся в данный момент в текстовом поле. Когда программа работает некорректно, очень часто проводится «распечатка» значений переменных в сценарии с целью убедиться, что данная переменная содержит то значение, которое вы и ожидали увидеть.

7. **Сохраните файл, обновите его в браузере и введите какой-нибудь текст в поле комментария.**

   Теперь консоль должна отображать содержимое комментария при вводе в поле каждого символа. Вы уже приобрели некоторый навык работы с консолью, так что добавим еще одно сообщение, после чего завершим сценарий.

8. **Отредактируйте функцию события keyup, добавив еще две строки (5 и 6):**

```
1 $('#comments').keyup(function() {
2  console.log('Событие: keyup');
3  var text = $(this).val();
4  console.log(' Содержимое комментария: %s',text);
5  var chars = text.length;
6  console.log('Количество знаков:',chars);
7 }); // конец keyup
```

   Строка 5 подсчитывает количество знаков, хранящихся в переменной `text` (см. раздел «Определение длины строки» главы 16 для получения сведений о свойстве `length`), и сохраняет его в переменной `chars`. Чтобы убедиться, что сценарий правильно подсчитывает число знаков, используйте функцию `log()` (строка 6) для отображения сообщения в консоли.

   Остается сделать только одно: завершить сценарий таким образом, чтобы показать посетителю число введенных знаков.

9. **Добавьте последнюю строку в конец функции события keyup (строка 10), чтобы сценарий выглядел так:**

```
1 <script>
2 $(document).ready(function() {
3  console.log('READY');
4  $('#comments').keyup(function() {
5        console.log('Событие: keyup');
```

```
6         var text = $(this).val();
7  console.log('Содержимое комментария: %s',text);
8         var chars = text.length;
9  console.log('Количество знаков: %d' ,chars);
10 $('#count').text(chars + " знаков");
11 }); // конец keyup
12 }); // конец ready
13 </script>
```

10. Сохраните файл и просмотрите его в браузере.

Теперь консоль должна выглядеть, как показано на рис. 17.5. Готовую версию этого руководства — *готовый_ 17_01.html* — можно найти в папке *глава17*.

ПРИМЕЧАНИЕ

Получив функционирующую должным образом программу, вы должны удалить из вашего сценария весь код, включающий функцию `console.log()`, так как при работе с некоторыми браузерами она сгенерирует ошибку.

Дополнительные средства отладки

Консоль JavaScript предоставляет отличный способ отслеживания хода выполнения программы с помощью сообщений. Но иногда программа выполняется так быстро, что трудно уследить за тем, что происходит на каждом этапе. Здесь вам нужен способ замедления хода событий. К счастью, в браузерах предусмотрен мощный отладчик JavaScript, который позволяет по шагам, строка за строкой, исследовать все, что происходит при выполнении программы.

ПРИМЕЧАНИЕ

В данном разделе показан отладчик в браузере Chrome, но вы можете найти подобные отладчики JavaScript в браузерах Firefox, Opera, Safari и Internet Explorer.

Отладка — это процесс исправления неправильно функционирующей программы путем выявления и устранения ошибок. Чтобы по-

настоящему понять, как программа функционирует (или не функционирует), вам нужно увидеть, как она работает, шаг за шагом.

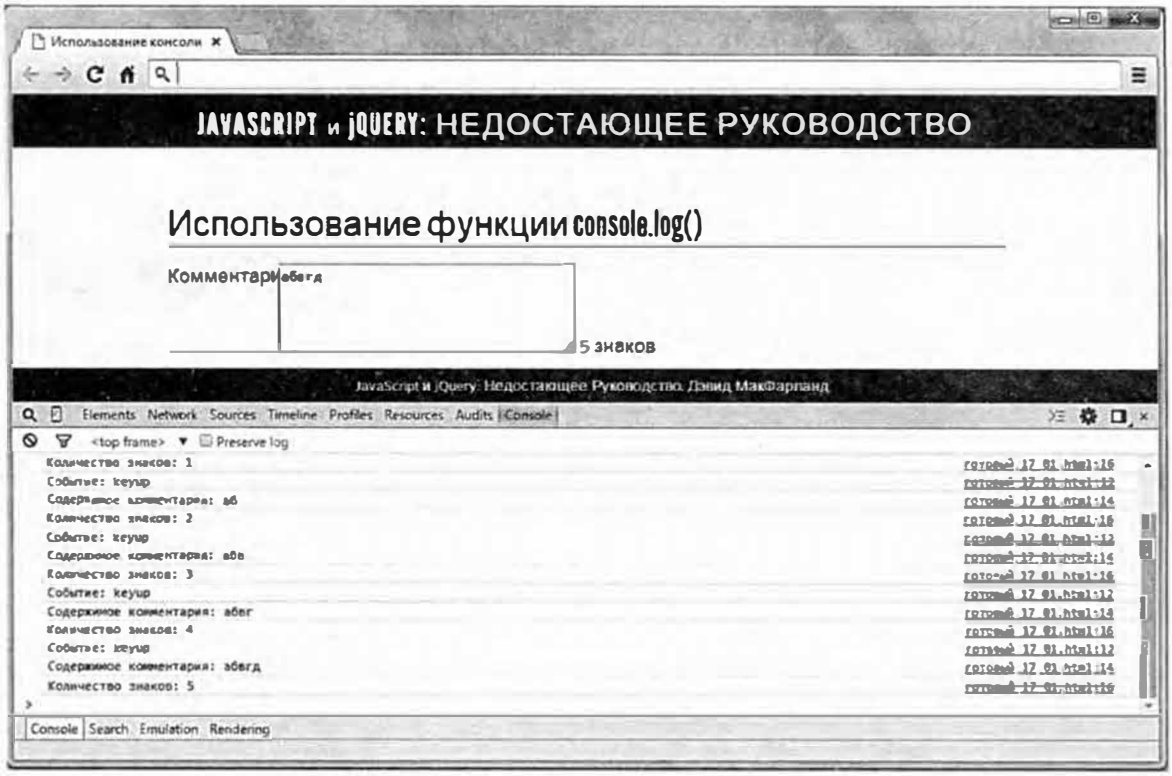

Рис. 17.5. Консоль JavaScript предоставляет отличный способ отображения диагностической информации по ходу выполнения программы. Вы можете группировать серии сообщений (например, все сообщения, выдаваемые во время выполнения цикла) путем добавления функции `console.group()` перед первым сообщением `console.log()` в группе и функции `console.groupEnd()` после последнего сообщения

Чтобы воспользоваться отладчиком, вы отмечаете определенные строки, как *точки останова* (*breakpoints*). Точка останова представляет собой место, где интерпретатор JavaScript прекращает выполнение кода и ждет. Далее вы можете использовать возможности отладчика для выполнения программы по одной строке. Так вы сможете ясно увидеть, что происходит в конкретной строке. Вот основные этапы процесса.

1. **Откройте веб-страницу в браузере.**

   Помните о том, что приведенные инструкции относятся к браузеру Chrome и его отладчику. В отладчиках других браузеров конкретные шаги и панели могут отличаться.

2. **Откройте консоль JavaScript.**

   Следуйте инструкциям в разделе «Отслеживание ошибок» главы 1, чтобы открыть средства разработчика вашего браузера. В браузере

Chrome вы можете открыть консоль JavaScript с помощью сочетания клавиш **Ctrl+Shift+J** (ОС Windows) или **⌘+⌥+J** (OS X).

3. **Щелкните по вкладке Источники (Sources) и в списке файлов выберите файл, содержащий код JavaScript, который необходимо отладить (рис. 17.6).**

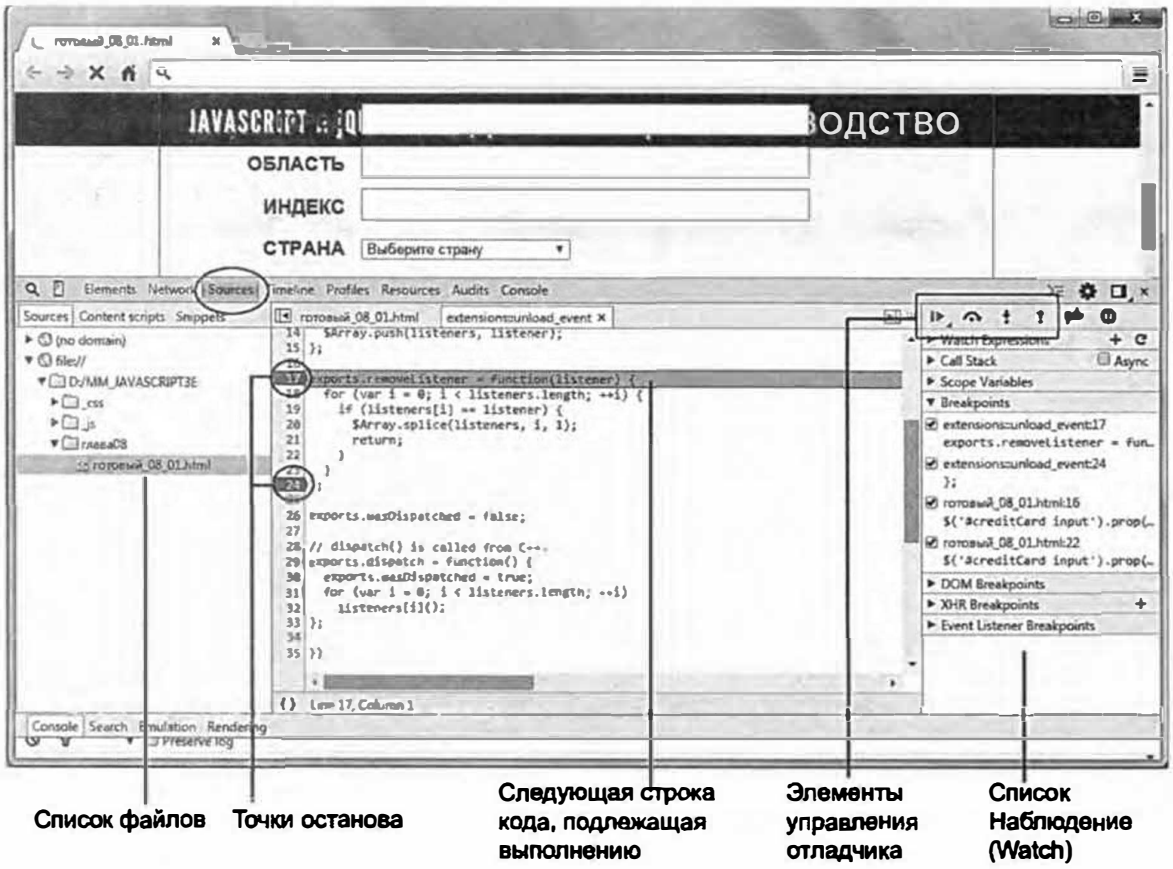

Рис. 17.6. Отладчик позволяет задавать точки останова (строки кода, где сценарий прерывает работу и ждет), управлять выполнением сценария и отслеживать переменные в списке **Watch** (Наблюдение). Когда отладчик выполняет сценарий, следующая строка, подлежащая выполнению, выделяется

В браузере Chrome вкладка **Sources** (Источники) содержит исходный код файла, подлежащего отладке. Если сценарий написан в веб-странице, вы увидите код для всей страницы (включая HTML). В случае с внешним файлом JavaScript вы увидите только код JavaScript, содержащийся в этом файле.

4. **Выберите файл с интересующим вас сценарием из меню в левой части отладчика (см. рис. 17.6).**

Как правило, сценарии хранятся в разных файлах: на самой веб-странице либо в одном или в нескольких внешних файлах JavaScript.

Если ваша страница использует сценарии из разных файлов, нужно выбрать тот файл, который содержит интересующий вас сценарий.

5. **Добавьте точки останова.**

   Для этого щелкните слева от номера строки. Появится отметка, указывающая на точку останова в этом месте.

ПРИМЕЧАНИЕ

Добавление точки останова в строку, содержащую только комментарий, не приведет к каким-либо результатам — в этом месте отладчик не прервет работы программы. Поэтому добавляйте точки останова только в строки с кодом JavaScript.

6. **Перезагрузите веб-страницу.**

   Поскольку вы должны просматривать веб-страницу в браузере, чтобы добавить точки останова, код JavaScript, подлежащий отладке, может быть уже выполнен (до установки точек останова). В этом случае следует перезагрузить страницу, чтобы запустить код JavaScript сначала.

   Если вы установили точку останова в функцию, которая отвечает на событие (например, нужно отладить код, выполняющийся при нажатии клавиши или наведении указателя мыши на ссылку), то это событие надо запустить, чтобы достичь точки останова и начать процесс отладки.

   После запуска сценария, когда будет достигнута точка останова, процесс прервется. Программа станет ожидать выполнения строки, отмеченной первой точкой останова.

7. **Используйте элементы управления отладчиком для пошагового выполнения программы.**

   Большинство отладчиков предусматривают четыре кнопки управления (рис. 17.7), определяющими ход выполнения программы после ее прерывания в точке останова. Описание этих кнопок приведено в следующем разделе.

8. **Отслеживайте состояние программы в окне Watch (Наблюдение) (см. рис. 17.7).**

   Целью пошагового режима отладки программы является выяснение того, что происходит в каждой строке сценария. Окно **Watch** (Наблюдение) предоставляет базовую информацию о состоянии про-

граммы и позволяет добавлять дополнительные переменные, которые вы хотите проверить. Например, если необходимо отслеживать значение переменной в ходе выполнения сценария, вы можете сделать это в окне **Watch** (Наблюдение). Вы узнаете, как использовать окно **Watch** (Наблюдение) в следующем разделе.

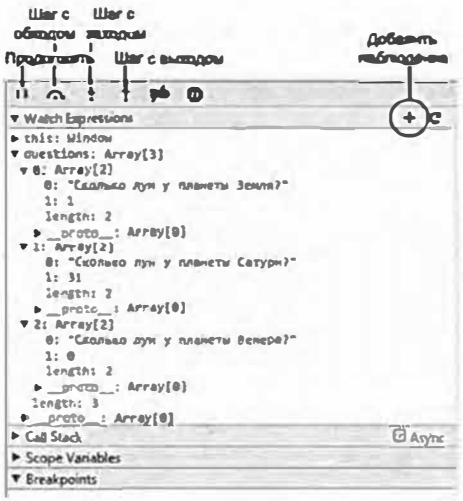

Рис. 17.7. Окно **Watch** (Наблюдение) отображает значения разных переменных в ходе выполнения программы. Вы можете добавлять в список собственные инструкции, появляющиеся на фоне серых полос в верхней части окна

9. **Исправьте сценарий в текстовом редакторе.**

   Надеемся, что пошаговая отладка сценария позволила вам понять, что пошло не так, например, почему значение переменной никогда не меняется или почему управляющая инструкция никогда не оценивается как ложное. Получив нужную информацию, вы можете перейти в текстовый редактор и изменить ваш сценарий. (Пример отладки сценария приведен в следующем руководстве.)

10. **Проверьте страницу в браузере и, если нужно, повторите все необходимые шаги для продолжения отладки.**

Управление сценарием с помощью отладчика

После расстановки точек останова и перезагрузки страницы вы будете готовы к пошаговой отладке сценария. Если вы добавили точку останова в ту часть сценария, которая выполняется при загрузке страницы, программа прервется в точке останова; если же точка останова помещена в строку, выполняющуюся только при возникновении события (типа щелчка по ссылке), вам необходимо запустить это событие, чтобы попасть в соответствующую точку останова.

Когда отладчик прерывает программу в точке останова, он не выполняет соответствующей строки кода, а останавливает программу непосредственно *перед* выполнением этой строки. После этого вы можете нажать одну из четырех кнопок, управляющих дальнейшими действиями отладчика (см. рис. 17.7):

- **Play** (Продолжить). Просто запускает сценарий. Остановка произойдет, только если интерпретатор JavaScript встретит точку останова или сценарий завершит работу. В точке останова выполнение программы прервется, и она будет ожидать нажатия одной из четырех кнопок управления отладчиком.

  Таким образом, пользуйтесь кнопкой **Play** (Продолжить), если хотите выполнить программу или перейти к следующей точке останова.

- **Step Over** (Шаг с обходом). Выполняет текущую строку кода, а затем останавливается на следующей строке сценария. Если текущая строка содержит вызов функции, входа в функцию не произойдет — отладчик «перешагнет» через нее и остановится на следующей строке. Это очень удобно, если вы точно знаете, что вызываемая функция работает безупречно. Например, если ваш сценарий содержит вызов jQuery-функции, вы захотите переступить через него — в противном случае вам пришлось бы потратить уйму времени, просматривая устрашающий программный код jQuery строку за строкой. Вы выберите кнопку **Step Over** (Шаг с обходом), если только не анализируете строку кода, содержащую созданную вами функцию. В этом случае вы захотите посмотреть, что происходит внутри этой функции, используя кнопку **Step Over** (Шаг с обходом), описанную далее.

- **Step Into** (Шаг с заходом). Предоставляет отладчику доступ в функцию. Это значит, что, если вы оказались в строке, содержащей вызов функции, нажатие данной кнопки позволит отладчику войти в функцию и остановиться на ее первой строке. Этот способ необходим, когда вы не уверены, где находится причина проблемы: в главной части сценария или внутри функции, которую вы создали.

  Не используйте данную кнопку, если уверены, что вызываемая функция работает корректно, например, если вы пользовались ею уже не один десяток раз. Также следует отказаться от использования кнопки **Step Into** (Шаг с заходом) в пользу использования кнопки **Step Into** (Шаг с заходом) при отладке строки, включающей селектор или команду jQuery. Например, селектор `$('#button')` представляет собой способ библиотеки jQuery для выделения элемента

на странице. Однако это еще и функция библиотеки jQuery, так что нажав кнопку **Step Into** (Шаг с заходом) при обнаружении jQuery-функции, вы окажетесь в сложном мире библиотеки jQuery (а о том, что это произошло, вы узнаете по вкладке **Script** (Сценарий), которая будет отображать код JavaScript для файла *jquery.js*).

Если отладчик все же заведет вас в дебри функции или библиотеки JavaScript вроде jQuery, вам поможет четвертая кнопка, о которой сейчас и пойдет речь.

- **Step Out** (Шаг с выходом). Выводит отладчик из функции. Обычно эта кнопка используется после использования кнопки **Step Out** (Шаг с выходом). Когда вы нажмете ее, функция продолжает выполняться как обычно, но остановок на каждой строке кода функции больше не будет, как это случилось бы при использовании кнопок **Step Over** (Шаг с обходом) и **Step Into** (Шаг с заходом). После нажатия кнопки **Step Out** (Шаг с выходом) отладчик вернется к строке, из которой функция была первоначально вызвана, после чего остановит выполнение программы.

Наблюдение за происходящим в сценарии

Кнопки в верхней части окна отладчика позволяют управлять выполнением сценария. Основной же сутью отладчика является возможность видеть то, что происходит внутри сценария. Окно **Watch** (Наблюдение) (см. рис. 17.7) предоставляет список переменных и функций в контексте выполняемой в данный момент строки кода. Это означает, что, поставив точку останова внутри функции, вы получите список всех переменных, которые для нее определены. Если вы поставили точку останова в основной части сценария, то получите список всех переменных, которые определены в нем. В окне **Watch** (Наблюдение) вы также найдете любую из созданных вами функций.

Вы можете добавить собственные переменные и инструкции, щелкнув по кнопке **Add watch expression...** (Добавить наблюдение...) в виде символа «+». Просто щелкните по ней, и появится текстовое поле. Введите имя переменной, которую вы хотите отследить, или инструкцию JavaScript, которую желаете выполнить. Например, поскольку отладчик не следит за переменной счетчика в цикле *for* (см. раздел «Циклы `for`» главы 3), вы можете добавить ее и, просматривая шаг за шагом цикл, наблюдать, как счетчик меняет свое значение после очередного прохождения цикла.

Окно **Watch** (Наблюдение) можно представить как нечто вроде непрерывной команды `console.log()`. Она выводит значения переменной или инструкции для определенной строки кода, которая выполняется в данный момент.

Окно **Watch** (Наблюдение) позволяет заглянуть внутрь программы с помощью эффекта стоп-кадра, что помогает вам точно определить, где именно в вашем сценарии произошла ошибка. Например, если вы знаете, что некая переменная имеет числовое значение, вы можете шаг за шагом проследить, каково это значение в момент создания переменной и как оно меняется в ходе выполнения программы (сценария). Если после нажатия кнопок **Step Over** (Шаг с обходом) или **Step Into** (Шаг с заходом) вы заметили, что значение переменной изменилось не так, как вы ожидали, то, вероятно, вы нашли строку, в которой закралась ошибка.

ПРИМЕЧАНИЕ

Имейте в виду то, что когда выполнение программы приостановлено, отладчик выделяет следующую строку кода, подлежащую исполнению. Таким образом, если эта строка устанавливает значение переменной, то вы не увидите этого нового значения в списке **Watch** (Наблюдение), пока не щелкните по кнопке **Step Over** (Шаг с обходом).

Отладка на практике

В данном руководстве вам предстоит отладить файл, содержащий всевозможные ошибки (синтаксические, логические, ошибки этапа выполнения). Веб-страница представляет собой простую программу-викторину, которая предлагает три вопроса и показывает результаты. (Откройте файл *готовый_17_02.html* из папки *глава17* в любом веб-браузере, чтобы посмотреть, как эта страница должна функционировать.)

ПРИМЕЧАНИЕ

Обратитесь к примечанию в разделе «Ваша первая программа на языке JavaScript» главы 1 для справки о копировании файлов примеров.

Как и в предыдущем руководстве, вы будете использовать браузер Chrome. Вы можете воспользоваться другой программой, однако в этом случае консоль JavaScript и отладчик могут отличаться.

1. **Запустите браузер Chrome и откройте файл *17_02.html* из папки *глава17*.**

   Откройте консоль JavaScript, нажав сочетание клавиш **Ctrl+Shift+J** (ОС Windows) или **⌘+⌥+J** (OS X).

   Консоль отображает информацию о двух ошибках. Сначала вы займетесь второй: «Uncaught SyntaxError: Unexpected token ;». Значение этой ошибки не является очевидным, но с ее помощью браузер сообщает о том, что он не ожидал встретить в данном месте программы символ `;`. Учитывая то, что точка с запятой обозначает окончание инструкции, здесь должна быть какая-то ошибка. Также обратите внимание на номер строки, в которой она возникла (обведено на рис. 17.8).

2. **Щелкните по номеру строки (выделен на рис. 17.8).**

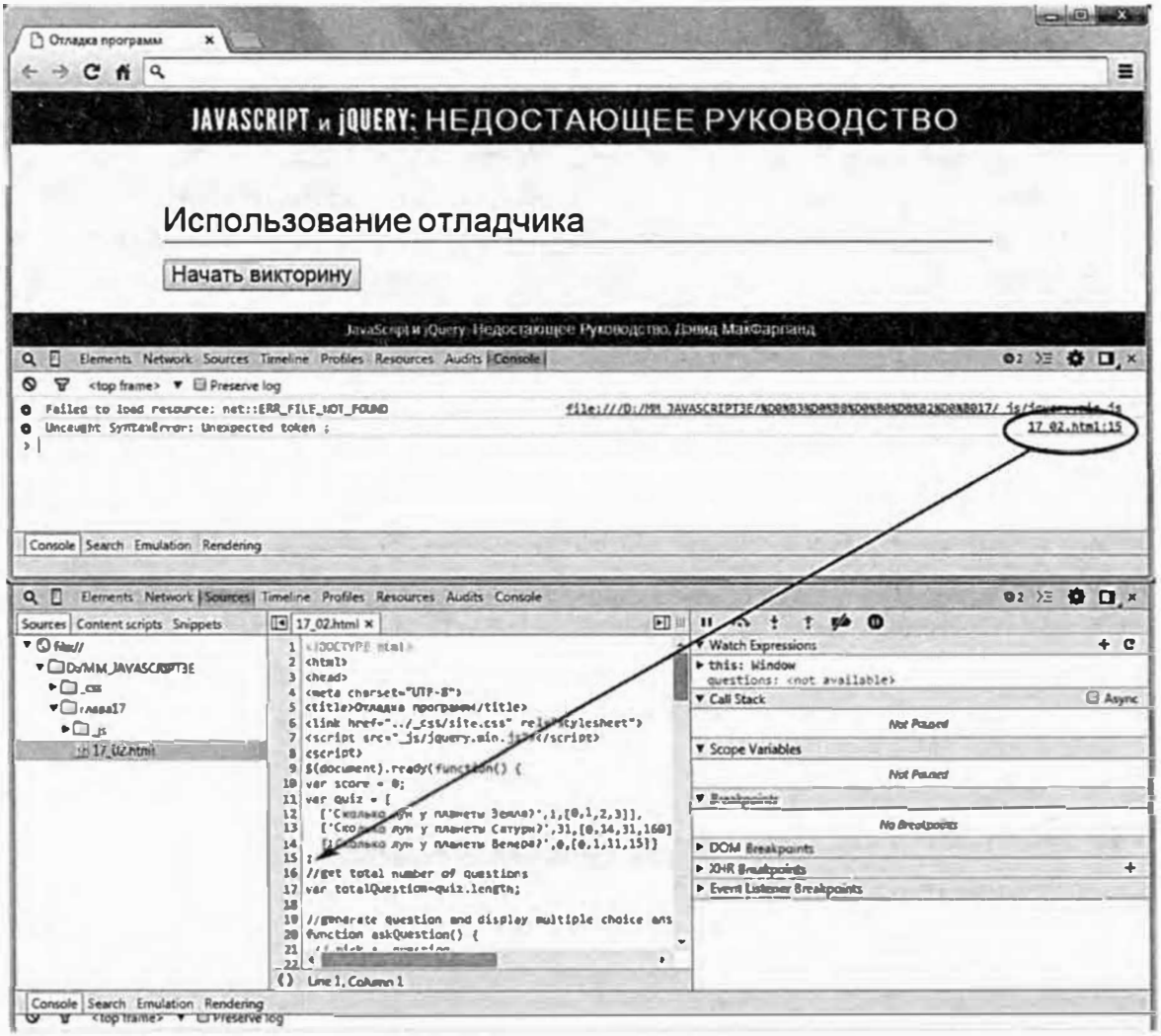

Рис. 17.8. Вы можете перейти из консоли в конкретное место в коде, где возникла ошибка, щелкнув по номеру строки рядом с ошибкой в консоли (обведено). После этого появится панель **Sources** (Источники), а строка с ошибкой будет подсвечена

Браузер Chrome откроет панель **Sources** (Источники) и выделит строку с ошибкой. Помните о том, что во многих случаях ошибки представляют собой простые опечатки. Часто ошибка заключается в отсутствующем знаке пунктуации, например, `)`, `"`, `]` или `}`, таким образом, вы можете сразу понять, в чем проблема. В данном случае у вас есть массив `var quiz = [`, начинающийся на строке 11, в котором отсутствует закрывающая квадратная скобка `]`.

3. **Откройте файл *17_02.html* в текстовом редакторе. Найдите строку 15 (в которой содержится единственный знак ;). Поставьте закрывающую квадратную скобку перед точкой с запятой:**

   ```
   ];
   ```

   Эта скобка завершает вложенный массив, содержащий все вопросы и ответы викторины.

4. **Сохраните файл и вновь загрузите страницу в браузере Chrome.**

   Теперь в коде присутствует одна из первоначальных ошибок и новая ошибка! На этот раз консоль отображает ошибку: «$ is not defined» ($ не определен) и указывает на строку 9, содержащую jQuery-функцию `$(document).ready()`. Когда браузер сообщает о том, что какой-либо элемент не определен, это значит, что код ссылается на то, чего на самом деле не существует, — это может быть имя переменной или функции, которые не были созданы. Также вероятно наличие опечатки в коде. В данном случае код выглядит правильно. Проблема возникла ранее на странице, вот здесь:

   ```
   <script src="_js/jquery.min.js"></script>
   ```

   Распространенной проблемой при работе с внешними файлами является случайная ошибка при вводе пути к файлу сценария. В данном случае файл *jquery.min.js* располагается внутри папки *_js*, которая является внешней по отношению к папке с этим файлом. Код указывает на то, что данный файл должен быть внутри папки, в которой находится данная веб-страница; а поскольку браузер не может найти файл *jquery.min.js* (где и определена специальная jQuery-функция `$()`), он выдает сообщение об ошибке.

5. **Измените элемент `script`:**

   ```
   <script src="../_js/jquery.min.js"></script>
   ```

   Код `../` указывает, что папка *js* является внешней для этой папки, и теперь путь верно указывает на файл jQuery. Что еще может быть неправильно в этой программе?

6. **Сохраните файл и вновь откройте его в браузере Chrome.**

   Ошибок нет, поскольку обе ошибки были вызваны одной и той же проблемой. Первая ошибка сообщала о том, что файл jQuery не найден. А поскольку браузер не мог найти этот файл, он не знал, что означает символ `$`. Похоже, страница исправлена... или нет?

7. **Нажмите кнопку «Начать викторину».**

   Ну вот! Опять ошибка. На этот раз консоль сообщает: «askQuestions is not defined» (функция askQuestions не определена), и указывает на строку 69 ближе к концу сценария. Поскольку эта ошибка появляется только при выполнении программы, здесь мы имеем дело с ошибкой на этапе выполнения (см. врезку «Типы ошибок» ранее в главе). Проблема присутствует в управляющей инструкции:

```
if (quiz.length>0) {
askQuestions();
} else {
giveResults();
}
```

   Вы, вероятно, уже усвоили: если что-то не определено, то причиной может быть простая опечатка. В данном случае код `askQuestions()` является вызовом функции, так что потратьте некоторое время на просмотр кода в попытке найти эту функцию.

   Нашли? Функции `askQuestions()` здесь нет, зато есть функция `askQuestion()` (без буквы *s*).

8. **Вернитесь в текстовый редактор и удалите последнюю букву *s* в коде `askQuestions()` в строке 69 (ближе к концу сценария). Сохраните файл, перезагрузите страницу в браузере и вновь нажмите кнопку «Начать викторину».**

   Теперь мы видим вопрос викторины с пятью вариантами ответа. К сожалению, последний вариант помечен, как «undefined» (не определен). Похоже на ошибку. Однако консоль JavaScript пуста, так что технически код JavaScript в порядке. Вероятно, что-то не так с *логикой* программы. Чтобы проникнуть в суть проблемы, нам нужен отладчик.

9. **В браузере Chrome щелкните по вкладке Источник (Sources) и выберите файл *17_02.html* в меню (рис. 17.9).**

Вкладка **Sources** (Источник) предоставляет доступ к JavaScript-коду страницы. Если страница включает код JavaScript и связана с внешними файлами JavaScript, меню выбора источника кода позволяет указать, какой код JavaScript необходимо отладить.

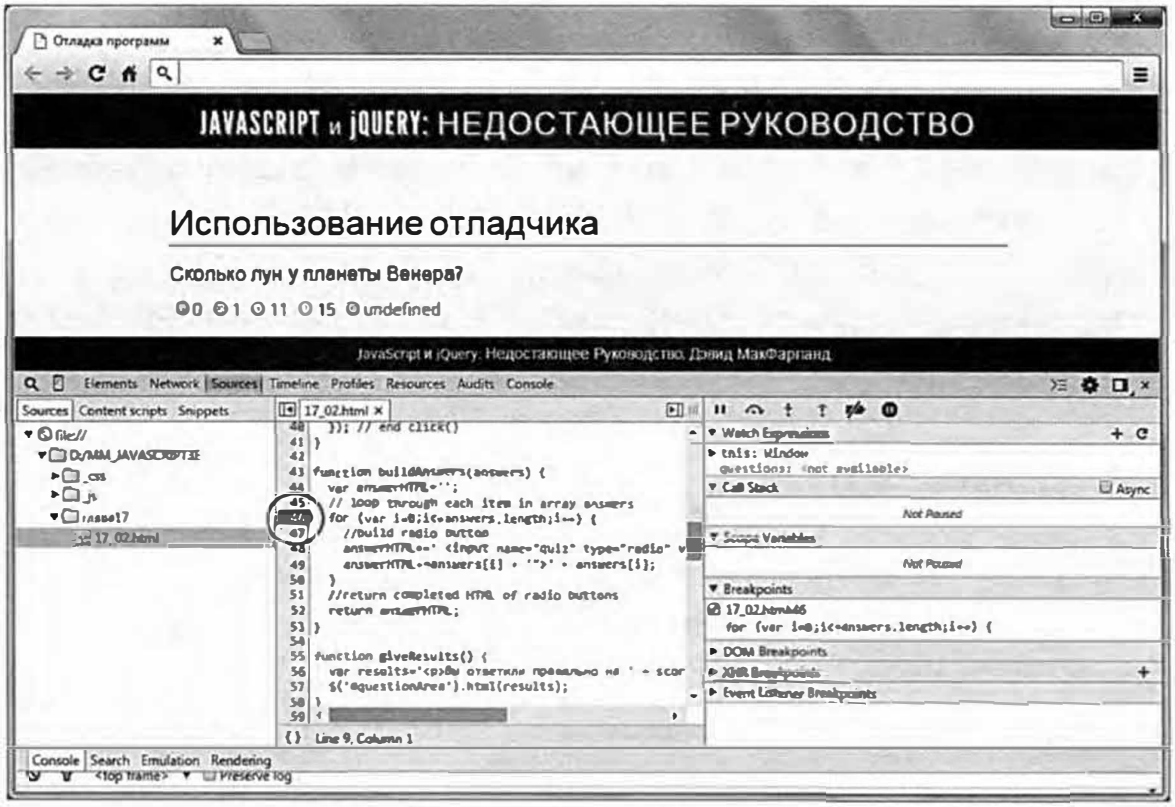

Рис. 17.9. В окне консоли вы можете отлаживать любые сценарии, используемые на текущей странице. Меню **Источник** (Sources) позволяет выбрать код JavaScript, внедренный в текущую страницу или хранящийся во внешнем файле JavaScript

Поскольку логикой программы не предусматривается создание переключателя «undefined» (не определен), хорошо бы начать искать проблему с кода, который создает переключатели. Если этот сценарий написали вы, то, вероятно, помните, где искать; если вы просто обрабатываете чужой сценарий, придется основательно потрудиться, чтобы найти нужный фрагмент.

В данном случае переключатели создаются внутри функции `buildAnswers()`, которая предназначена для построения переключателей для выбора ответов. Этой функции передается массив, включающий список значений для каждого переключателя. По завершении выполнения функция `buildAnswers()` возвращает строку, содержащую HTML-код для переключателей. Таким образом, код данной функции является правильным местом для начала отладки.

10. **В средней панели вкладки Sources (Источник), в которой отображен код HTML и JavaScript страницы, прокрутите код до строки 46. Щелкните слева от нее, чтобы вставить точку останова (обведена на рис. 17.9).**

    У этой строки появилась отметка, которая указывает на точку останова, то есть на место в коде, где интерпретатор JavaScript прерывает выполнение сценария. Другими словами, при возобновлении выполнения сценария интерпретатор JavaScript, достигнув этой строки, остановится, и вы сможете в пошаговом режиме проанализировать код, чтобы выяснить, что происходит «под капотом».

    Отладчик также дает возможность отслеживать значения переменных по ходу выполнения программы подобно тому, как это происходит при использовании функции `console.log()` в разделе «Использование функции console.log() для отслеживания выполнения сценария» данной главы. Далее вы сообщите отладчику, какие переменные вы хотите отследить.

11. **В правой части окна щелкните по кнопке в виде символа «+» (справа от слов Watch Expression (Наблюдение), введите `i` и нажмите клавишу Enter.**

    Этот шаг добавляет в список **Watch** (Наблюдение) переменную `i`. Эта переменная используется в цикле `for` в качестве счетчика количества повторений цикла (см. раздел «Циклы `for`» главы 3 для получения сведений о циклах). В ходе выполнения сценария вы будете видеть, как меняется значение этой переменной. Теперь добавьте еще одну переменную для отслеживания.

12. **Вновь щелкните по кнопке в виде символа «+», введите код `answers.length` и нажмите клавишу Enter.**

    Пусть вас не смущает значение, которое показывает в данный момент отладчик (вероятно, вы видите что-то вроде «answers.length: undefined»). Вы не можете реально отслеживать значения переменных, пока не запустили отладчик и не вошли в функцию, в которой определена переменная. Итак, пришло время заглянуть в сценарий.

13. **Нажмите кнопку Обновить (Reload) в браузере или воспользуйтесь сочетанием клавиш Ctrl+R (⌘+R). После загрузки страницы щелкните по кнопке «Начать викторину».**

    Начинается выполнение сценария, и на странице появляется первый вопрос. Но когда приходит время создать переключатели, от-

ладчик замирает на строке 46 (верхняя часть рис. 17.10). Заметьте, что в окне **Watch** (Наблюдение) значение переменной *i* недоступно (not available). Это связано с тем, что точка останова прервала программу прямо перед началом выполнения строки.

Иначе говоря, цикл не начался, и переменная *i* еще не была создана.

Однако значение `answers.length` установлено на 4. Массив `answers` представляет собой массив ответов, переданный функции. Свойство массива `length` указывает на число его элементов; в данном случае их четыре, так что по завершении выполнения функции вы должны получить четыре переключателя.

14. Нажмите кнопку Step Over (Шаг с обходом) (см. рис. 17.10).

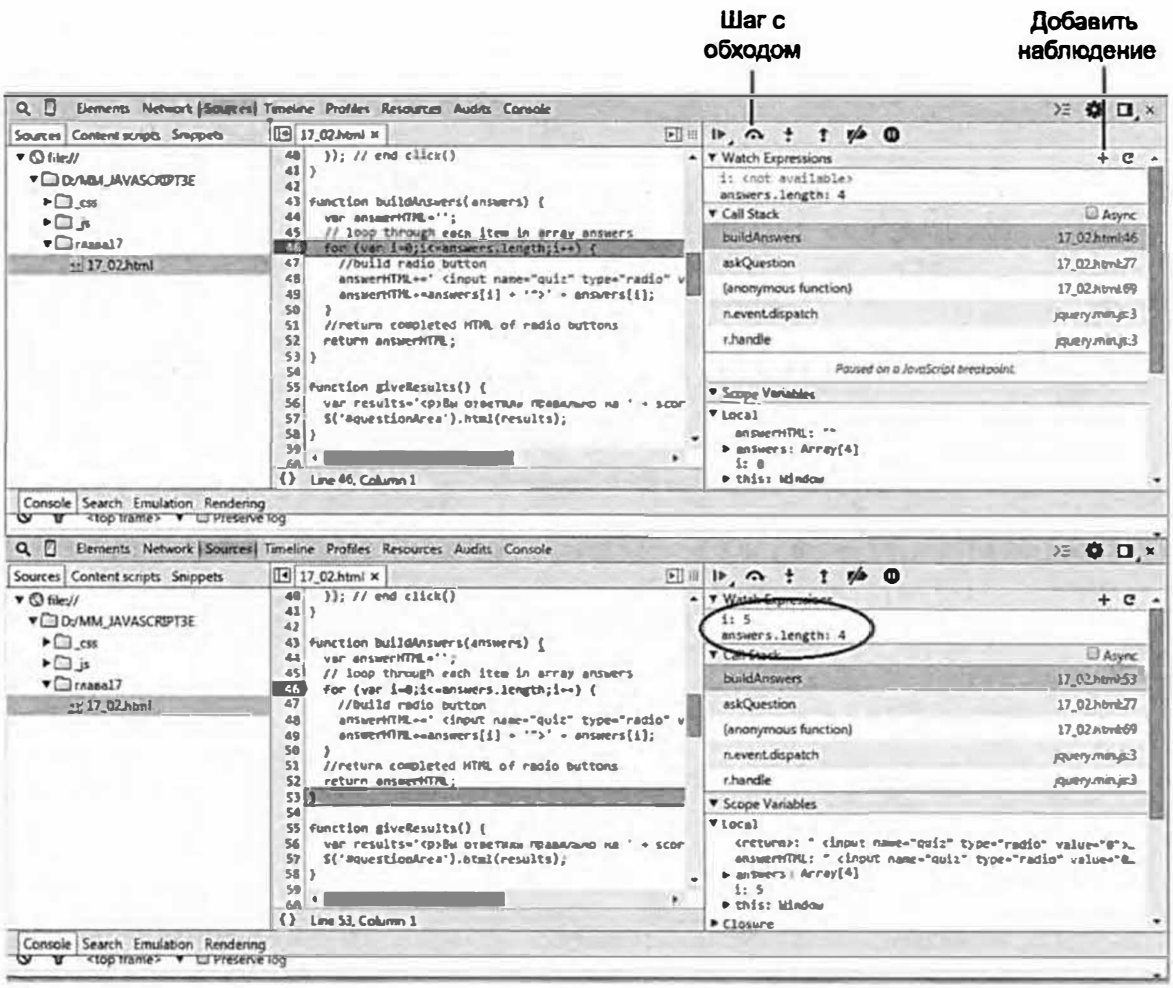

Рис. 17.10. В списке **Watch** (Наблюдение) может отображаться не только переменные или инструкции, которые вы добавили. Например, в разделе **Scope Variables** (Переменные области видимости) перечислены все переменные, принадлежащие «области видимости» текущей функции. В данном примере (верхняя инструкция) видно, что в функции `buildAnswers()` доступны обе переменные: и `answerHTML`, и `answers`

Она переведет вас на следующую строку программы. Теперь вы видите, что переменная *i* имеет значение 0. Продолжайте щелкать по этой кнопке, пока цикл не завершится.

15. **Нажимайте кнопку Step Over (Шаг с обходом), пока не увидите, что переменная `i` приняла значение 5 в списке Watch (Наблюдение) (нижняя часть рис. 17.10).**

    Хотя массив `answers` содержит лишь четыре элемента, можно заметить, что цикл `for` в действительности выполняется пять раз (на что указывает значение переменной `i`). С циклом происходит что-то странное. Вспомните, что средняя инструкция в цикле `for` представляет собой условие, которое должно быть истинным для того, чтобы цикл выполнялся (см. раздел «Циклы `for`» главы 3). В данном случае условие имеет вид `i<=answers.length;`. Другими словами, цикл начинается со значением переменной `i`, равным 0, и повторяется до тех пор, пока значение `i` остается меньшим или равным количеству элементов в массиве `answers`. Иначе говоря, `i` будет принимать значения 0, 1, 2, 3 и 4, пока цикл не завершится, то есть пять раз. Однако, поскольку в массиве `answers` содержится только четыре элемента, при пятом выполнении цикла из-за отсутствия пятого элемента массива отображается слово «undefined».

16. **Вернитесь в текстовый редактор и измените строку 46 следующим образом:**

    ```
    for (i=0;i<answers.length;i++) {
    ```

    Теперь цикл будет выполняться столько раз, сколько элементов содержит массив, создавая один переключатель для каждого из вариантов ответа.

17. **Сохраните файл и просмотрите его в браузере.**

    Вы можете убрать точку останова, щелкнув по метке около номера строки в панели **Sources** (Источник), чтобы код выполнялся без прерывания.

    Файл *готовый_17_02.html* содержит полную версию данного руководства. Как вы убедились, поиск программных ошибок может отнять много сил. Но инструменты отладки сильно упрощают задачу изучения «внутренностей» программы и выявления ошибок.

Часть VI
ПРИЛОЖЕНИЕ

Приложение А. Источники знаний по JavaScript

Приложение А
ИСТОЧНИКИ ЗНАНИЙ ПО JAVASCRIPT

Данная книга содержит достаточно информации, чтобы вы смогли начать уверенно программировать на языке JavaScript. Однако ни одно издание не способно ответить на все вопросы, касающиеся JavaScript и jQuery, которые могут возникнуть. О программировании на языке JavaScript можно узнать очень многое и это приложение предоставит вам отправные точки для дальнейшего исследования и обучения.

Справочные материалы

Иногда для чтения книги может понадобиться словарь. В процессе программирования на языке JavaScript полезно иметь под рукой полную справочную информацию о ключевых словах, терминах, методах и других компонентах синтаксиса JavaScript. Эти материалы доступны как в Интернете, так и в печатных изданиях.

Веб-сайты

- Сайт ECMAScript (**www.ecmascript.org/**) содержит документацию и информацию по языку ECMAScript (официальное название JavaScript). На этом ресурсе вы можете узнать о текущем (и будущем) положении вещей, связанном с языком JavaScript.
- Mozillia Developer Center Core JavaScript Reference (**developer.mozilla.org/ru/docs/Web/JavaScript/Reference**) — предоставляет полную справку по теме JavaScript, ориентированную на профессионалов.
- Сайт WebPlatform.org (**www.webplatform.org**) содержит описание языка JavaScript, DOM и таблиц CSS с указаниями того, какие свойства поддерживаются конкретным браузером. Это своеобразная энциклопедия веб-разработчика.

- MSDN JavaScript Language Reference (**msdn.microsoft.com/ru-RU/library/yek4tbz0(v=VS.94).aspx**) — превосходный ресурс для разработчиков, использующих программу Internet Explorer. Хотя этот сайт предоставляет техническую информацию о том, как язык JavaScript используется в других браузерах, данный ресурс в первую очередь посвящен программе Internet Explorer.

Книги

- «JavaScript. Подробное руководство», автор Дэвид Флэнаган (издательство «Символ-Плюс») — самая полная энциклопедия по теме JavaScript. Это увесистая книга, в которой можно найти все детали, необходимые для исчерпывающего понимания языка JavaScript.

Основы языка JavaScript

Освоение языка JavaScript является нелегким делом, поэтому использование широкого спектра источников никогда не повредит. Представленные ниже ресурсы помогают постичь основы языка JavaScript (которые подчас могут быть сложными для понимания).

Веб-сайты

- The W3 Schools JavaScript (**www.w3schools.com/js**) — подробная (хотя и не всегда понятная) инструкция, освещающая большинство аспектов программирования на языке JavaScript.
- An Introduction to JavaScript (**www.howtocreate.co.uk/tutorials/javascript/introduction**) предоставляет детальное описание языка JavaScript. Однако поскольку вы используете библиотеку jQuery, вам вряд ли понадобится этот ресурс, так как здесь в основном обсуждаются традиционные способы выбора и манипулирования элементами объектной модели документа.

Книги

- «Изучаем JavaScript», автор Майкл Моррисон (издательство «Питер») — замечательное, хорошо иллюстрированное введение в тему

JavaScript. Содержит большое количество сведений о программировании на языке JavaScript, однако не предоставляет достаточно практически полезных примеров.

Библиотека jQuery

Большая часть материала данной книги касается библиотеки jQuery. Однако вы можете узнать еще очень многое об этом мощном, увлекательном инструменте, способном сэкономить ваше время.

Веб-сайты

- Блог jQuery (**blog.jquery.com**) позволяет быть в курсе относительно того, что происходит с библиотекой jQuery.
- Документация по jQuery (**api.jquery.com**) ресурс, посвященный всем аспектам библиотеки jQuery. Там вы можете найти рабочие примеры, демонстрирующие как использовать каждую функцию библиотеки.
- Основы jQuery (**jqfundamentals.com**) — предусматривает уникальный подход к изучению библиотеки jQuery. Этот сайт не только содержит объяснения базовых концепций jQuery, но и предоставляет «песочницу» JavaScript, встроенную в каждую страницу сайта, что позволяет вам просматривать изменения кода в реальном времени.

Книги

- «jQuery. Подробное руководство по продвинутому JavaScript», авторы Бер Бибо и Иегуда Кац (издательство «Символ-Плюс») — подробно описывает библиотеку jQuery, материал издания дополнен большим количеством примеров. От читателя требуется некоторое представление о языке JavaScript и программировании.
- «Изучаем jQuery», авторы Эрл Каслдайн и Крэйг Шарки (издательство «Питер») содержит множество инструкций по использованию библиотеки jQuery в виде прикладных примеров, показывающих способы решения распространенных задач пользовательского интерфейса.

Расширенные возможности языка JavaScript

О да, язык JavaScript *сложнее*, чем вы могли подумать, прочитав эту книгу. Приобретя опыт программирования на JavaScript, вы, возможно, захотите расширить свои знания об этом языке.

Статьи и презентации

- JS-Must-Watch (**github.com/bolshchikov/js-must-watch/**) содержит список лучших презентаций и видео, посвященных JavaScript.

Веб-сайты

- Eloquent JavaScript (**eloquentjavascript.net**) предлагает руководства по использованию языка JavaScript. Отличается хорошей организацией и творческим подходом к созданию структуры уроков. Хотя сайт ориентирован на начинающего разработчика JavaScript, стиль изложения подходит скорее специалистам, так что это не лучшее место для тех, кто прежде не был знаком с языком JavaScript или программированием вообще. Этот ресурс также доступен в виде книги.
- The JavaScript section of Douglas Crockfords' World Wide Web (**javascript.crockford.com**) предоставляет большое количество (сложной) информации о языке JavaScript, подчас требующей для понимания ученой степени в области компьютерных технологий.
- Mozilla Developers Network JavaScript section (**developer.mozilla.org/ru/docs/Web/JavaScript**) содержит большой объем информации о языке JavaScript, включая справочник по JavaScript, о котором говорилось в начале этого приложения. Данный ресурс также содержит детальное руководство (**developer.mozilla.org/ru/docs/Web/JavaScript/Guide**), посвященное различным версиям JavaScript, а также примеры концепций этого языка в действии.

Книги

- «The Principles of Object-Oriented JavaScript», автор Николас Закас — это небольшая книга (менее 100 страниц), посвященная расширенным методам организации кода, которую стоит прочитать.

- «JavaScript. Шаблоны», автор Стоян Стефанов (издательство «Символ-Плюс»). Это издание несколько устарело, но по-прежнему содержит информацию, позволяющую вам продвинуться в области программирования на языке JavaScript. Она предоставляет вам «шаблоны» программ для решения распространенных задач, включая задачи повышения эффективности работы с объектными литералами, данными в формате JSON и массивами. Данная книга рассчитана на профессионалов.
- «JavaScript: сильные стороны», автор Дуглас Крокфорд (издательство «Питер») раскрывает наиболее ценные аспекты языка JavaScript, обходя неэффективные методы программирования. Автор знает, о чем пишет, поскольку является старшим архитектором JavaScript в компании Yahoo и изобретателем формата JSON. Книга короткая и лаконичная, но при этом содержит много мудрых советов о том, как лучше использовать язык JavaScript.

Язык CSS

Если вы взяли в руки эту книгу, то вы, вероятно, уже освоились с таблицами CSS. Язык JavaScript действительно позволяет использовать возможности форматирования CSS не только для настройки вида элементов, но даже для их анимации на экране. Чтобы освежить знания в области CSS, предлагаем некоторые полезные ресурсы.

Веб-сайты

- The Complete CSS Guide от WestCiv (**www.westciv.com/style_master/academy/css_tutorial**) освещает все аспекты каскадных таблиц стилей. Вы не найдете здесь большого количества техник, но сможете понять, что такое CSS и как создавать стили и таблицы стилей.
- The Mozilla Developer Network CSS Reference (**developer.mozilla.org/en-US/docs/Web/CSS/Reference**) — исчерпывающее руководство по использованию свойств CSS.
- Selectutorial (**css.maxdesign.com.au/selectutorial**) — отличный ресурс для изучения синтаксиса селекторов CSS. Поскольку библиотека jQuery в значительной степени базируется на идее использования селекторов CSS для манипулирования HTML-кодом страницы, очень важно хорошо понимать эту концепцию.

Книги

- «Большая книга CSS3», автор Дэвид Макфарланд (издательство «Питер») — полная, основанная на руководствах, книга о каскадных таблицах стилей. Наряду с основательным освещением темы CSS представлены практические примеры, а также советы по диагностике, призванные адаптировать таблицы CSS для работы в различных браузерах.
- «CSS. Каскадные таблицы стилей. Подробное руководство», автор Эрик А. Мейер (издательство «Символ-Плюс»). Название говорит само за себя: эта книга описывает таблицы CSS настолько детально, что вы повредитесь в рассудке, если решите осилить ее за один присест.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

$(document).ready(), функция
  в анимированной панели навигации, 297
  виджет Datepicker и, 504
  виджет Draggable и, 566
  виджет Droppable и, 584
  виджет Selectmenu и, 515
  виджет Sortable и, 606
  добавление html() в подсказку, 470
  добавление виджета Tabs, 478
  добавление комментария в, 182
  использование с функцией console.log(), 839
  обработка нескольких экземпляров, 815
  помещение в тег <head>, 247
  помещение программного кода в, 832
  при создании виджета Accordion, 492
  при создании диалогового окна, 444
  при стилизации кнопок, 522
  с функцией автозаполнения, 529
$.getJSON() функция, 674
$(this), переменная, 218
$ (символ) в именах переменных, 75, 271, 709
$() функция, jQuery() и, 728
<body>, теги, 24
  помещение шаблонов перед закрывающим, 469
<button>, элемент, 522, 523, 540
, тег
  в качестве родительского элемента, 744
  вставка пустого, 647
  использование для сортировки элементов, 604
  как блочный элемент, 284
  превращение в область бросания, 590
  превращение в перетаскиваемый элемент, 567, 580
  скрывание полей формы, 389
<h2> (заголовок 2) теги, 25
<head> (теги)
  в DOM, 187
  описание, 23
  помещение функции $(document).ready() в, 247
<html> (теги)
  описание, 23
<img>, теги, 194, 215, 314, 318
!important, правило, 561
<input>, теги, 369
.js, расширение файла, 48
<link>, теги, 362
, теги, 358, 494, 706
<message>, теги, 665
.min (минимизированный), 438
, теги, 604
<option>, теги, 375
<p>, теги
  как блочный элемент, 284
  открывающий тег абзаца, 24
<script>, теги
  в разделах страницы head и body, 136
  описание, 45
  создание шаблонов между, 468
<select>, теги, 369, 515
, теги, 706
<style>, теги, 429
<textarea>, теги, 369
, теги, 358, 604

A

accept, параметр виджета Droppable, 586
accordion(), функция jQuery, 356, 489, 491
activate, событие
  виджет Sortable, 614
  виджета Droppable, 592
activeClass, параметр виджета Droppable, 586
active, псевдокласс, 304
active, свойство
  виджета Аккордеон, 489
  панели с вкладками, 475
addClass()
  функция jQuery, 208, 336
after(), функция jQuery, 204, 752
Ajax, 632
  значения слова асинхронный, 641
  компоненты, 635
  обработка данных с сервера, 658
  обработка ошибок, 663
  основы, 635
  получение данных с сервера, 535
  серверы и клиенты, 47
  средства jQuery и, 642
  форматирование данных для отправки на сервер, 653
alert(), функция JavaScript, 53, 54, 69, 241, 259
AMP (Apache, PHP и MySQL), 638
AngularJS, библиотека, 172
animate, параметр виджета Аккордеон, 489
animate(), функция jQuery, 287, 288, 290, 295, 365, 398, 433, 613, 738

API ключ, 682
appendTo(), функция jQuery, 427
append(), функция jQuery, 203, 695, 732, 733, 752
Aptana Studio, программа, 30
attr(), функция jQuery, 215, 313, 314, 318, 323, 337, 341, 649
Autocomplete, виджет
  использование массивов с, 530
  описание, 527
  параметры, 538
  получение данных с сервера, 534
autocomplete(), функция jQuery, 529
axis, параметр
  виджета Sortable, 607

B

BBEdit, программа, 31
beforestop, событие виджета Sortable, 615
before(), функция jQuery, 204, 752
bind(), функция jQuery, 257
blind, эффект jQueryUI, 623
blur(), метод для объекта окна, 348
blur, событие, 382
blur, событие формы, 234
bounce, эффект jQueryUI, 451, 623
Brackets, программа, 30
buttonset(), функция jQuery, 525–527
Button, виджет, 522
button, селектор, 373
button(), функция jQuery, 524, 525, 527, 703

C

cancel, параметр
  виджета Draggable, 569
  виджета Sortable, 608
cancel(), функция jQuery, 617
CDN (Сеть доставки (и дистрибуции) контента), 173
ceil(), функция JavaScript, 789
changeMonth, параметр виджета Datepicker, 505
changeYear, параметр виджета Datepicker, 505
change, событие
  виджета Sortable, 615
  формы, 233, 384
checkbox, селектор, 373
checked, атрибут, 377
checked, фильтр, 374
children(), функция jQuery, 745
Chrome, браузер
  консоль, 298, 821
  отслеживание ошибок в, 61
  просмотр визуализированного HTML-кода, 207
click, событие, 229, 230, 244, 261–263, 330, 338, 383
click(), функция jQuery, 257, 263, 286, 306, 335, 339, 383, 598, 648, 649, 714
clip, эффект jQueryUI, 623
clone(), функция jQuery, 206
closest(), функция jQuery, 747
close(), метод для объекта окна, 347
CoffeeCup Free HTML Editor, программа, 30
collapsible, свойство
  виджета Accordion, 489
  панели с вкладками, 475
connectToSortable, параметр
  виджета Draggable, 570
connectWith, параметр виджета Sortable, 608
console.log(), функция, 836
containment, параметр
  виджета Draggable, 570
  виджета Sortable, 609
contains(), фильтр, 198
CouchDB, база данных, 47
create, событие
  виджета Draggable, 578
  виджета Sortable, 613
CSS3
  анимация jQuery и, 276, 303
  книга, 21
  переходы jQuery и, 303
CSS (Каскадные таблицы стилей), 26
  блок объявления, 27
  добавление стилей на веб-страницы, 25
  обеспечение презентационного слоя веб-страницы, 21
  селектор, 27, 189
  селекторы, 192
css(), функция jQuery, 210, 212, 215, 310, 395, 581, 598, 739, 741
cursorAt, параметр
  виджета Draggable, 571
  виджета Sortable, 610
cursor, параметр
  виджета Draggable, 571
  виджета Sortable, 609
События, остановка, 256

D

data, свойство, 253
data(), функция jQuery, 738
dateFormat, параметр виджета Datepicker, 506
Datepicker, виджет, 183, 354, 356, 433, 502
  использование, 503, 510
  настройка свойств, 504
datepicker(), функция jQuery, 505, 511
Date, объект, 795
dblclick, событие мыши, 230
dblclick(), функция jQuery, 240
deactivate, событие
  виджета Droppable, 593
  виджета Sortable, 615
delay, параметр
  виджета Autocomplete , 539
  виджета Sortable, 610

delegate(), функция jQuery, 257
destroy(), функция jQuery, 616, 617
detach(), функция jQuery, 717, 721
Dialog, виджет, 442, 445, 465, 554, 556, 562, 566, 705
  добавление кнопок, 453
  описание, 432
  передача параметров, 449
  создание модальных диалоговых окон, 555
dialog(), функция jQuery, 444, 445, 449, 451–453, 457
disabled, параметр
  виджета Draggable, 572
  виджета Droppable, 587
disable(), функция jQuery, 617
distance, параметр
  виджета Sortable, 610
  эффекта bounce, 623
document.getElementById(), функция JavaScript, 187
document.getElementsByClassName(), функция JavaScript, 188
document.getElementsByTagName(), функция JavaScript, 188
document.write(), функция JavaScript, 56, 89, 107, 136
document, значение
  параметра containment виджета Draggable, 571
  параметра containment виджета Sortable, 609
Dojo Toolkit, библиотека, 172
DOM (Объектная модель документа), 186
  манипулирование веб-страницей, 737
  методы jQuery для навигации по, 271
  обход, 742
  функции обхода, 737
do/while, циклы, 146
Draggable, виджет
  добавление на веб-страницу, 566
  описание, 565
  параметры, 569
  применение, 567
  события, 578
draggable, свойство диалогового окна, 445
draggable(), функция jQuery, 567, 569, 572, 580, 591
drag, событие виджета Draggable, 582
Dreamweaver, программа, 31
drop
  событие виджета Droppable, 590
  эффект jQueryUI, 624
Droppable, виджет
  использование, 584
  описание, 583
  параметры, 586
  руководство, 595
  события, 589
droppable(), функция jQuery, 585

E

each(), функция jQuery, 216, 218, 322, 336, 337, 678, 680, 693, 726, 738, 753
easing, параметр, 448, 490, 626
Eclipse, программа, 30
ECMAScript, 19
EditPlus, программа, 31
effect(), функция jQuery, 621–625, 628, 712, 721
Ember.js, библиотека, 172
empty(), функция jQuery, 755
em, единица измерения, 27
enable(), функция jQuery, 617
encodeURIComponent(), функция JavaScript, 656
end(), функция jQuery, 750, 751
errorPlacement, объект, 427
error(), функция jQuery, 664
event, свойство
  виджета Accordion, 490
  настройка панели с вкладками, 476
even, фильтр, 196
explode, эффект jQueryUI, 624

F

fadeIn(), функция jQuery, 200, 275, 279, 283, 289, 333, 365, 398
fadeOut(), функция jQuery, 216, 275, 279, 294, 306, 334, 398
fadeToggle(), функция jQuery, 279, 284, 286
fadeTo(), функция jQuery, 279, 292, 743
fade, эффект jQueryUI, 624
file, селектор, 373
find(), функция jQuery, 486, 613, 666, 710, 734, 737, 743, 745
Firebug, плагин, 65
Firefox, браузер
  консоль ошибок, 65
  просмотр визуализированного HTML-кода, 207
  сообщения в консоли ошибок, 820
first, фильтр, 197
fit, значение параметра tolerance виджета Droppable, 588
Flash, технология, 47
Flickr, сервис
  JSON-фид, 686
  добавление фида на сайт, 681
  использование нескольких ID, 684
  метки для уточнения поиска, 685
  описание, 681
  построение URL-адреса, 683
  руководство, 689
floor(), функция JavaScript, 791
focus()
  метод для объекта окна, 348
  функция jQuery, 386
focus, событие формы, 233, 380

fold, эффект jQueryUI, 624
Form, виджет, 540
for, циклы, 144

G

getData(), функция JavaScript, 726
getDate(), функция JavaScript, 793
getDay(), функция JavaScript, 793, 794
getFullYear(), функция JavaScript, 793
getHours(), функция JavaScript, 793
getMinutes(), функция JavaScript, 793
getMonth(), функция JavaScript, 793
getSeconds(), функция JavaScript, 793
getTime(), функция JavaScript, 793, 795
get(), функция jQuery, 652, 664
grid, параметр
  виджета Draggable, 572
  виджета Sortable, 610

H

handle, параметр
  виджета Draggable, 573
  виджета Sortable, 610
has(), фильтр, 197
heightStyle, свойство
  виджета Accordion, 490
  настройка панели с вкладками, 476
height, свойство диалогового окна, 445
helper, параметр виджета Draggable, 573
hidden, селектор, 272, 373
hidden, фильтр, 198
hide, свойство
  всплывающей подсказки, 466
  диалогового окна, 447
  панели с вкладками, 475
hide(), функция jQuery, 278, 333, 468, 602, 621
highlight, эффект jQueryUI, 624
hoverClass, параметр виджета Droppable, 587
hover, псевдокласс, 231
hover, событие, 296
hover(), функция jQuery, 250, 297, 298, 319, 326
HTML5
  атрибут placeholder, 381
  добавление атрибута checked в форму, 377
  проверка форм, 400
  тип документа, 22
HTML-Kit, программа, 30
HTML-атрибуты, применение jQuery к, 215
HTML-валидатор, 25
HTML-редакторы, 104
HTML-теги, 22, 23
  body, 24
  br, 140
  h2, 25
  head, 23
  p, 24
  script, 44
  select, 369
  textarea, 369
  закрывающий, 23
  открывающий, 23
  установка и чтение атрибутов, 206
html(), функция jQuery, 201, 486, 662, 731, 749
HTML (Язык разметки гипертекста), 22
  вложенные теги, 187
  добавление в подсказку, 467
  обеспечение структурного слоя веб-страницы, 21
  проверка, 24
  просмотр визуализированного кода, 206
  создание шаблонов внутри тегов <script>, 468
  теги, 22
  функции для работы с кодом, 749

I

icons, свойство
  виджета Accordion, 490
  виджета Button, 524
  виджета Selectmenu, 517
ID (идентификатор), 734
IIS, веб-сервер Microsoft, 638
image, селектор, 373
indexOf(), функция jQuery, 760, 764
input, селектор, 373
Internet Explorer, браузер
  JavaScript и, 18
  консоль ошибок, 64
  обработка событий, 235
  открытие консоли ошибок, 834
  отображение ошибок, 820
  проверка кода на наличие опечаток, 298
  просмотр визуализированного HTML-кода, 207
intersect, значение параметра tolerance виджета Droppable, 588
isNaN(), функция JavaScript, 69, 382, 788
is(), функция jQuery, 272
items, параметр виджета Sortable, 611

J

JavaScript
  Ajax и, 632
  ECMAScript, 19
  библиотека jQuery, 20
  библиотеки, 170, 172
  внешние файлы, 47
  встроенные функции, 69
  выражения-переключатели, 809
  грамматика, 68
  добавление на страницу, 44
  интерпретатор, 44
  история, 17
  источники знаний по, 858

ключевые слова, 77
комментарии, 110
компрессор, 817
литерал объекта, 212
массивы, 93
математические операторы, 79
методы, 108
минимизация ошибок при программировании, 832
минимизированный файл, 177
модифицирование веб-страниц, 182
написание программы, 40
объединение массивов и разбиение строк, 812
объект, 108
оператор, 79
описание, 16
отслеживание ошибок, 60
отступы в коде, 59
переменные, 74
пробелы, табуляция и возврат каретки в языке, 79
программное обеспечение, 29
событийно управляемый язык, 228
сохранение параметров в объектах, 806
сохранение параметров в переменных, 803
тернарный оператор, 807
типы данных, 70
ускорение загрузки файлов, 817
условные выражения, 117
функции, 148
этапы написания программы, 161
JavaScript Object Notation (JSON)
доступ к данным, 675
сложные объекты, 677
формат, 673
JavaScript-компрессор Yahoo, 818
Java, язык программирования, 18
join(), функция JavaScript, 143
jPanel, плагин, 368
jQuery, 20
Ajax и, 642, 738
CSS3 и, 303
CSS и, 737
анимация, 287
версии, 175, 178
выборки, 198
выбор элементов страницы, 188
выбор элементов формы, 372
добавление броских цитат с помощью, 220
добавление данных в элементы страницы, 738
добавление на страницу, 179
добавление содержимого на страницу, 201, 731
документация, 735
загрузка файла, 177
обход дерева DOM, 742
объект, 189
описание, 170
определение высоты и ширины объектов, 739
отложенный объект, 739
плагины, 350
полезные сведения о библиотеке, 728
получение и установка значений атрибутов, 736
преимущества, 171
продвинутые селекторы, 193
работа с каждым элементом выборки, 216
решаемые задачи, 312
связывание функций, 199
селекторы, 188, 733
селекторы для полей формы, 373
служебные функции, 739
смена изображений, 312
события, 235, 245
сохранение выборок в переменных, 729
способы добавления на веб-страницу, 173
управление поведением ссылок, 335
установка и чтение атрибутов тега, 206
утилиты, 738
фильтры, 196
чтение и изменение свойств CSS, 210
эффекты, 276
jQuery Foundation, 434
jQuery UI
CSS-файлы, 557
ThemeRoller, приложение, 548
альтернатива для, 435
взаимодействия, 565
виджет Accordion, 486
виджет Autocomplete, 527
виджет Button, 522
виджет Datepicker, 503
виджет Draggable, 565
виджет Droppable, 583
виджет Form, 540
виджет Selectmenu, 514
виджет Sortable, 604
добавление на веб-страницу, 439
добавление новой темы на сайт, 556
использование, 436
компоненты, 432
настройка внешнего вида, 548
описание, 432
переопределение стилей, 558
применение, 434
работа с формами, 502
стилизация виджетов, 562
эффекты, 565, 620
jQuery() и $(), функция jQuery, 729
jQWidgets, набор плагинов, 436
jScript, язык, 18
JSMin, программа, 817
JSONP, формат, 681
JSON, формат
доступ к данным, 675
описание, 673
сложные объекты, 677

K

Kendo UI, набор плагинов, 436
keydown, событие клавиатуры, 235
keypress, событие клавиатуры, 234
keyup, событие клавиатуры, 235

L

lastIndexOf(), функция jQuery, 761
live(), функция jQuery, 257
load, событие документа/окна, 231
load(), функция jQuery, 642, 643, 645, 646

M

Mac OS X Yosemite (OC), использование JavaScript, 19
MAMP, инсталлятор, 639
match(), функция jQuery, 779, 781
maxDate, параметр виджета Datepicker, 507
menu(), функция jQuery, 497
minDate, параметр виджета Datepicker, 508
minLength, параметр виджета Autocomplete , 539
modal, свойство диалогового окна, 447
MongoDB, база данных, 47
monthNames, параметр виджета Datepicker, 506
Mootools, библиотека, 172
mousedown, событие мыши, 230
mouseEnter, событие, 296
mouseLeave, событие, 296
mousemove, событие мыши , 231
mouseout, событие мыши , 231
mouseover, событие мыши , 231
mouseover(), функция jQuery, 257, 263, 598
mouseup, событие мыши , 231
moveBy(), метод для объекта окна, 348
moveTo(), метод для объекта окна, 348
Multi-level Push Menu, плагин, 368

N

Netscape Navigator, браузер, 18
new Date(), функция JavaScript, 800
next(), функция jQuery, 270, 748, 749
noConflict(), функция jQuery, 729
node.js, веб-сервер, 47
Notepad (Блокнот), программа, 29
Notepad++, программа, 30
not(), функция jQuery, 342
null, значение, 780
numberOfMonths, параметр виджета Datepicker, 506
Number(), функция JavaScript, 85, 786

O

off(), функция jQuery, 254, 255
on(), функция jQuery, 257, 721
opacity, параметр
  виджета Draggable, 574
  виджета Sortable, 611
open(), функция JavaScript, 344
Opera, браузер
  просмотр визуализированного HTML-кода, 208
out, событие
  виджета Droppable, 594
  виджета Sortable, 615
over, событие
  виджета Droppable, 593
  виджета Sortable, 616

P

Packer, программа, 817
pageX, свойство, 253
pageY, свойство, 253
parent, значение
  параметра containment виджета Draggable, 570
  параметра containment виджета Sortable, 609
parent(), функция jQuery, 746, 747
parseFloat(), функция JavaScript, 787
parseInt(), функция JavaScript, 137, 786
password, селектор, 373
placeholder
  атрибут, 381
  параметр виджета Sortable, 611
pointer, значение параметра tolerance виджета Droppable, 589
pop(), функция JavaScript, 101
position, свойство
  всплывающей подсказки, 467
  диалогового окна, 448
post(), функция jQuery, 652, 657
prepend(), функция jQuery, 203, 670, 718, 752
preventDefault(), функция jQuery, 254, 338, 339, 452
prev(), функция jQuery, 749
processContacts(), функция JavaScript, 675
processData(), функция JavaScript, 669
prompt(), функция JavaScript, 91, 92, 785
Prototype, библиотека, 729
puff, эффект jQueryUI, 625
pulsate, эффект jQueryUI, 625
push(), функция JavaScript, 99, 101

Q

querySelectorAll(),функция JavaScript, 188

R

radio, селектор, 373
random(), функция JavaScript, 791
Raphaлl, библиотека, 173
ready(), функция jQuery, 246
receive, событие виджета Sortable, 616
removeAttr(), функция jQuery, 215

removeClass(), функция jQuery, 209, 629
remove, событие виджета Sortable, 616
remove(), функция jQuery, 205, 721, 753
replaceWith(), функция jQuery, 315, 752
replace(), функция JavaScript, 696
reset, селектор, 373
reset, событие формы, 233
resizable, свойство диалогового окна, 445
resizeBy(), метод для объекта окна, 349
resizeTo(), метод для объекта окна, 349
resize, событие документа/окна, 232
revertDuration, параметр виджета Draggable, 575
revert, параметр виджета Draggable, 574
Rollover (сменяемое изображение), 318
round(), функция JavaScript, 789
RSS-лента, 682

S

Safari, браузер
  консоль ошибок, 66, 834
  просмотр визуализированного HTML-кода, 207
scale, эффект jQueryUI, 625
scope, параметр
  виджета Draggable, 576
  виджета Droppable, 588
screenX, свойство, 253
screenY, свойство, 253
scrollBy(), метод для объекта окна, 349
scrollTo(), метод для объекта окна, 349
scroll, событие документа/окна, 232
search(), функция jQuery, 764
Selectmenu, виджет, 514
selectmenu(), функция jQuery, 515, 518
selector, значение
  параметра containment виджета Draggable, 570
  параметра containment виджета Sortable, 609
send(), функция jQuery, 641
serialize(), функция jQuery, 617, 658
shake, эффект jQueryUI, 625
shiftKey, свойство, 253
shift(), функция JavaScript, 101
show, свойство
  всплывающей подсказки, 466
  диалогового окна, 447
  панели с вкладками, 475
show(), функция jQuery, 277, 621
siblings(), функция jQuery, 748
size, эффект jQueryUI, 626
slice(), функция jQuery, 761
slideDown(), функция jQuery, 273, 280, 398, 718, 738
slideToggle(), функция jQuery, 280, 284
slideUp(), функция jQuery, 275, 280, 289, 398, 616, 716
slide, эффект jQueryUI, 626
SmartMenu, плагин, 357, 360, 366
snapMode, параметр виджета Draggable, 577
snapTolerance, параметр виджета Draggable, 577
snap, параметр виджета Draggable, 576
Sortable, виджет, 604
  параметры, 607
sortable(), функция jQuery, 612
sort, событие виджета Sortable, 614
source, параметр виджета Autocomplete , 539
split(), функция jQuery, 813
src, атрибут
  изменение для изображения, 313
  тега <script>, 48
start, событие
  виджета Draggable, 579
  виджета Sortable, 613
stopPropagation(), функция jQuery, 256
stop, событие
  виджета Draggable, 583
  виджета Sortable, 615
stop(), функция jQuery, 302
SublimeText, программа, 31
submit, селектор, 373
submit, событие формы, 233
submit(), функция jQuery, 378
switchClass(), функция jQuery, 629

T

Tabs, виджет, 470
tabs(), функция jQuery, 472, 475, 482, 485
target, свойство, 253
TextEdit, программа, 29
TextWrangler, программа, 30
text, селектор, 373
text(), функция jQuery, 200, 202, 659, 710, 731, 749
ThemeRoller, приложение
  категории, 550
  описание, 548
this, ключевое слово, 218
times, параметр эффекта bounce, 623
toArray(), функция jQuery, 619
toFixed(), функция jQuery, 376, 790
toggleClass(), функция jQuery, 209, 285, 306
toggle(), функция jQuery, 252, 278, 284, 621
tolerance, параметр виджета Droppable, 588
toLowerCase(), функция JavaScript, 759
tooltipClass, свойство
  всплывающей подсказки, 467
Tooltip, виджет
  добавление HTML в, 467
  описание, 463
  параметры, 465
tooltip(), функция jQuery, 464, 465, 467
touch, значение параметра tolerance виджета Droppable, 589
track, свойство всплывающей подсказки, 466

U

ui, параметр, 519, 590
unbind(), функция jQuery, 257
Underscore.js, библиотека, 173
unload, событие документа/окна, 232
unshift(), функция JavaScript, 99
unwrap(), функция jQuery, 754, 755
update, событие виджета Sortable, 615
URL-адрес
  абсолютный, 50
  относительно документа, 51, 829
  относительно корневого каталога, 51
  относительно протокола, 176
  типы, 50
UTC (Универсальное координированное время), 802

V

validate(), функция jQuery, 402, 403, 408, 413, 418
Validation, плагин, 400
  добавление сообщений об ошибках, 405
  продвинутая верификация, 406
  продвинутые правила верификации, 408
  продвинутые сообщения об ошибках, 413
  простая верификация, 403
val(), функция jQuery, 244, 374, 375, 377, 399, 706, 738

W

WAMP, программа, 638
which, свойство, 253
width, свойство диалогового окна, 445
Wijmo UI, набор виджетов, 436
window, значение
  параметра containment виджета Draggable, 571
  параметра containment виджета Sortable, 609
window, объект, 109
  внешние ссылки для открытия нового окна, 340
  свойства, 345
  события, 231
  создание нового, 344
wrapInner(), функция jQuery, 754
wrap(), функция jQuery, 753

X

XHR (XMLHttpRequest), 635
XHTML, 22
XML, 632
  корневой элемент, 665
  формат, 665

Y

Yahoo JavaScript-компрессор, 818
Yahoo User Interface Library, библиотека, 172
yearRange, параметр виджета Datepicker, 509

Z

zIndex, параметр виджета Draggable, 578

А

Абсолютное позиционирование, 281
Абсолютный URL-адрес, 50
Автоматические циклы, 199
Адрес электронной почты, регулярное выражение для, 774
Айк, Брендан, 17
Аккордеон, виджет, 486
  компоненты, 488
  параметры, 489
  создание, 491
Активное состояние поля формы, 380
& (амперсанд) в составе строки запроса, 656
Амперсанд (&) в составе строки запроса, 656
Анимация jQuery
  CSS3 и, 303
  выполнение действия после завершения эффекта, 291
  изменения класса, 627
  описание, 287
  параметр easing, 289, 626
  руководство, 295
  цвет, 287
Анонимная функция, 216, 218, 219, 237, 244
  ключевое слово "this", 218
Аргумента передача, 153
Атрибута селекторы, 194
Атрибут тега, 24
  alt, 314
  height, 313
  href, 24
  src, 48, 313
  target, 340
  type, 45
  width, 313

Б

Базы данных, 46
  сервер, 637
Бесконечный цикл, 141
Бесплатные программы
  Aptana Studio, 30
  Brackets, 30
  CoffeeCup Free HTML Editor, 30
  Eclipse, 30
  HTML-Kit, 30
  Notepad++, 30
  TextWrangler, 30
Библиотеки JavaScript, 57, 170, 435
Блокнот (Notepad), программа, 29
Блочный элемент как часть виджета Accordion, 488
Браузер
  Ajax и, 635
  Ajax и объект XMLHttpRequest, 635
  JavaScript и, 18

анимация свойств CSS, 304
веб-консоль, 833
движок, 44
добавление новых CSS-свойств, 306
использование функции load(), 648
консоль ошибок, 60
локальное хранилище, 725
обработка событий в, 235
ожидание загрузки HTML-кода, 245
открытие внешних ссылок в новом окне, 340
просмотр визуализированного HTML-кода, 207
свойства окна, 345
создание нового окна, 344
тестирование всплывающих окон, 346
Браузерный движок, 44
Бросания, область, 587
Броская цитата, 220
Булево значение
истина, 73
ложь, 73
Булев тип данных
выключение поля формы, 387
использование, 121
описание, 73
флаг, 122

В

Валидатор HTML-кода, 25
Валюта, форматирование значений, 789
Веб-адрес, регулярное выражение для, 777
Веб-страницы
выбор элементов, 188
добавление JavaScript на, 44
добавление виджета Draggable на, 566
добавление виджета Droppable на, 584
добавление виджета Sortable на, 604
добавление меню на, 493
добавление содержимого на, 201
добавление сообщений с помощью диалоговых окон, 440
добавление фида сервиса Flickr на, 681
заголовок, 23
модель DOM, 186
модифицирование, 182
написание текста на, 55, 103
проверка, 24
слои, 21
тело, 24
Верификация
добавление правил, 403
правила, 401
продвинутая, 406
простая, 403
формы, 399
Взаимодействия jQuery UI
виджет Draggable, 565
виджет Droppable, 589
виджет Sortable, 612
описание, 433
Виджеты
Accordion, 486
Autocomplete, 527
Button, 522
Datepicker, 183, 354, 356, 433, 502
Draggable, 565
Droppable, 583
Form, 540
Selectmenu, 493
Sortable, 604
Tooltip, 463
Визуализированный HTML-код, просмотр, 206
Вложенные
массивы, 163
списки, 358
условные выражения, 131
Внешние файлы JavaScript, 47, 173
прикрепление, 57
Возврат каретки в языке JavaScript, 79
Восьмеричное число, 787
Всплытие события, 256
Встроенные функции JavaScript, 69
Выборки jQuery, 198
замена и перемещение, 205
работа с каждым элементом, 216
Вызов функции, 150
Выражение-переключатель, 809
Высота и ширина объектов, функции для определения, 739

Г

Генерация случайных чисел, 791
Гиперссылка, 24
Глобальная переменная в функциях, 159
Глобальный контекст функций, 159

Д

Данные, функции для внесения в элементы страницы, 738
Даты
отслеживание, 793
регулярное выражение для, 776
Делегирование событий, 262, 267, 714
- (дефисы) использование в свойствах CSS, 288
Диалоговые окна, 451
добавление кнопок в, 453
настройка свойств, 444
открытие, 451
создание, 442
Динков, Васил, плагин SmartMenus , 360
// для создания комментария, 110
Документ, URL-адрес относительно, 51, 829
Дочерние селекторы, 194

З

Заголовок, содержащий текст как часть виджета Accordion, 488
Закрывающие HTML-теги, 23
, (запятая) для отделения пар имя/значение в литерале объекта, 366
Запятая (,) для отделения пар имя/значение в литерале объекта, 366
Зарезервированные слова, использование, 826
Знак переключения кода, 73
Изображения
  добавление в подсказку, 468
  добавление фида сервиса Flickr, 681
  изменение атрибута src, 313
  предварительная загрузка, 316
  смена, 312
  смена с помощью jQuery, 315
  сменяемое (rollover), 318
  фотогалерея с эффектами, 327
  заключение строк в формате JSON, 674
  заключение строки, 63
  отладка, 821
  создание пустой строки, 91
  экранирование, 823

И

Интерактивные элементы, 16
Интерпретатор JavaScript, 43, 44
Интерфейс программирования приложений (API), 681
И, оператор (&&), 128

К

Кавычки в строке, 72
" " или ' ' (кавычки)
Каскадные таблицы стилей (CSS), 26
  абсолютное позиционирование, 281
  анимация, 307
  блок объявления, 27
  выбор файлов jQueryUI, 438
  добавление стилей на веб-страницы, 25
  значение, 28
  классы, 206
  на сайте с документацией по jQuery, 737
  обеспечение презентационного слоя веб-страницы, 21
  объявление, 27
  описание, 192
  переходы, 628
  преобразования, 581
  селектор, 27, 189
  создание навигационной панели, 360
  специфичность, 559
  чтение и изменение свойств CSS, 210
  эффект смены изображения, 327
[ ] (квадратные скобки), создание массива, 95
Квадратные скобки ([ ]), создание массива, 95
Класс, 206
  анимация изменения, 627
  селектор, 191
Клиенты и серверы, 46
Ключевые слова JavaScript, 77
Кнопки
  добавление в диалоговое окно, 453
  добавление в приложение Список дел, 699
  настройка, 523
  проверка наличия меток, 377
  стилизация, 521
  форматирование в приложении Список дел, 700
Кодирование (экранирование) знаков, 656
Комбинации клавиш, 36
Комментарий, 110
/*...*/ комментарий на нескольких строках, 111
Коммерческие программы
  BBEdit, 31
  Dreamweaver, 31
  EditPlus, 31
  SublimeText, 31
Компилирование, 43
Компилируемые и скриптовые языки, 42
Компрессор, 817
Компьютерная программа, 43
Конкатенация
  автоматическое преобразование типов данных, 84
  оператор, 83
Консоль ошибок
  в браузере Chrome, 61
  в браузере IE9, 64
  в браузере Safari, 66, 834
  открытие в браузере, 833
  отладка с помощью, 833
  отслеживание ошибок, 60
Консорциум Всемирной паутины (W3C), 25
Корневой каталог, URL-адрес относительно, 51
/ косая черта
  в URL-адресе, 51
  в закрывающих HTML-тегах, 23
  как оператор деления, 79
Крокфорд, Дуглас
  JSMin, программа, 817
  книга, 862
() круглые скобки
  вызов функции, 150
  для группировки операторов, 82
  отсутствие закрывающей, 820

Л

Линейный метод easing, 289
Литерал массива, 95
Литерал объекта, 212
Логические ошибки, 826
Логический (булев) тип данных, 73

Логический оператор
  и, 128
  или, 129
Локальная переменная в функциях, 159
Локальное хранилище, 725

M

Манипулирование моделью DOM страницы, 737
Маркированный список
  создание, 358
  теги, 26
Маски
  группировка частей, 770
  нахождение в строках, 763
  нахождение соответствия, 779
Массив, 94
  вложенный, 163
  выбор элемента случайным образом, 791
  добавление элементов в, 98
  доступ к элементам, 96
  индекс, 97
  использование с виджетом Autocomplete, 530
  использование функции toArray(), 619
  литерал массива, 95
  многомерный, 163
  написание текста на веб-странице, 103
  объединение, 812
  описание, 93
  разбиение строк и объединение, 812
  свойство length, 98
  создание, 95
  создание очереди, 102
  удаление элементов из, 101
  циклы и, 142
Математические операторы, 79
Меню, 36
Меню навигации
  создание горизонтального, 498
  создание отзывчивого, 357
Меню эпизодов. См. Меню глав
Метод easing
  linear, 289
  swing, 289
Метод отправки данных
  GET, 639
  POST, 640
Методы для объекта окна
  blur(), 348
  close(), 347
  focus(), 348
  moveBy(), 348
  moveTo(), 348
  resizeBy(), 349
  resizeTo(), 349
  scrollBy(), 349
  scrollTo(), 349
(минус)
  как оператор вычитания, 79
  на веб-странице, 273
Многократная отправка формы, предотвращение, 389
Модальное диалоговое окно, 447
Моноширинный шрифт, 104

H

Неактивное состояние поля формы, 382
Незакрытые пары, ошибки, 819
Не число (NaN), 84
Нижнего подчеркивания, символ (_)
  в имени переменной, 77
  перед именем папки, 239

O

Область бросания, 587
Область видимости функции, 158
Обход дерева DOM, 742
Общение. См. Обсуждение
Объект Date, 793
Объект jQuery, 189
Объект XMLHttpRequest (XHR), 635, 639
  метод open(), 639
  метод send(), 641
  свойство responseText, 642
Объектная модель документа (DOM), 186
  манипулирование веб-страницей, 737
  методы jQuery для навигации по, 271
  обход, 742
  функции обхода, 737
Объект события, 252
Объекты, 107, 108
  window, 109
  методы, 108
  свойства, 108
  сохранение параметров в, 806
  экземпляры, 109
Окно оповещения JavaScript, 54, 259, 440
= (оператор), 77
Оператор, 79
  break, 811
  выбора, 809
  модуля (%), 798
!= (оператор) в условных выражениях, 118
!== (оператор) в условных выражениях, 119
< (оператор) в условных выражениях, 119
<= (оператор) в условных выражениях, 119
== (оператор) в условных выражениях, 118, 120, 828
=== (оператор) в условных выражениях, 118
> (оператор) в условных выражениях, 119
>= (оператор) в условных выражениях, 119
&& (оператор "И"), 128
|| (оператор "ИЛИ"), 129
% (оператор модуля), 798
*, оператор умножения, 79

Операторы сравнения, 118
  оператор неравенства, 120
  оператор равенства, 120
-- , оператор, 87
-= , оператор, 87
*= , оператор, 87
/= , оператор, 87
++ , оператор, 87, 141, 144
+= , оператор, 87
Остановка события, 256
Открывающие HTML-теги, 23
Отладка, 819, 842
  описание ошибок, 823
  ошибки, 819
  с помощью веб-консоли, 833
  типы ошибок, 825
Отладка с помощью веб-консоли, 833, 837
  наблюдение за происходящим в сценарии, 848
  обзор ошибок, 834
  открытие консоли, 833
  отладчик, 846
  отслеживание выполнения сценария, 836
  руководство по отладке, 849
  точки останова, 843
Отложенный объект, 739
Отступы в коде, 59
Очередь, создание, 102
Ошибки
  виды, 63
  зарезервированные слова, 826
  знаки равенства, 827
  кавычки, 823
  логические, 826
  настройка стилей для сообщений об, 415
  на этапе выполнения, 825
  область видимости, 831
  обработка, 663
  отслеживание, 60
  пути во внешнем файле, 829
  путь к внешнему файлу, 828
  распространенные, 819
  типы, 825
  учет регистра, 828

П

Панель с вкладками (Tabs), 470
Параметр easing, 289
Переключатель
  верификация, 424
  выбор, 396
  отключение поля формы, 394
  улучшение, 525
Переменная, 74
  глобальная, 159
  изменение значений, 86
  использование, 77
  комбинированные операторы, 87
  локальная, 159
  оператор присваивания, 77
  правила присвоения имен, 75
  создание, 75
  сохранение параметров в, 803
Переходы, 303
Плагин jQuery, 351
  Color, 287
  Navigation, 360, 363
  ScrollTo, 349
  основы работы с, 354
+ (плюс)
  как оператор конкатенации, 83
  как оператор сложения, 79
  на веб-странице, 273
Подмаски, использование для замены текста, 783
Поиск в строке, 759
Полный цикл разработки на JavaScript, 20
Помощник события, 235
Порядок операций, 82
Потомков селекторы, 193
Почтовый индекс, регулярное выражение для, 772
Правила верификации
  equalTo, 411
  max, 411
  maxlength, 411
  min, 411
  minlength, 411
  range, 411
  rangelength, 411
Предварительная загрузка изображения, 316
Пробел в языке JavaScript, 79
Прогрессивное улучшение, 648
Продвинутые селекторы
  дочерние селекторы, 194
  селекторы атрибута, 194
  селекторы потомков, 193
  соседние родственные селекторы, 194
Просмотр визуализированного HTML-кода, 206
Протокол, URL-адрес относительно, 176
Пустая строка, 91

Р

Разработки, сервер, 638
Регулярное выражение, 325, 763
  адрес электронной почты, 774
  веб-адрес, 777
  дата, 776
  построение, 765
  почтовый индекс, 772
  регекс, 764
  свойство global, 781
  создание, 764
  телефонный номер, 773
(решетка) в селекторе ID, 191

C

Свойства окна
height, 346
left, 346
location, 346
menubar, 347
scrollbars, 346
status, 346
toolbar, 346
top, 346
width, 346
Связывание функций, 199, 214, 730, 751
Селекторы CSS, 188
ID-селекторы, 190
селекторы классов, 191
селекторы элементов, 191
Сервер, 637
базы данных, 637
веб-сервер, 637
верификация формы с помощью, 413
клиент и, 46
общение с, 639
приложения, 637
разработки, 638
установка, 638
Сети. См. Социальные сети
Сеть доставки (и дистрибуции) контента (CDN), 173
Символ границы слова, 767
_ (символ нижнего подчеркивания)
в имени переменной, 77
перед именем папки, 239
Синтаксис, 42
Скорость эффекта, 276
fast, 277
normal, 277
slow, 277
Скриптовые и компилируемые языки, 42
Скрипты, 43
в Интренете, 18
Слайдеры, программирование, 350
Служебные функции, 739
Смена изображений, 312
с помощью jQuery, 315
Сменяемое изображение, 318
добавление, 320
Событийно управляемый язык, 228
События
виджета Draggable, 578
виджета Droppable, 589
виджета Sortable, 612
всплытие, 256
делегирование, 262, 267, 714
запуск, 229
использование, способ jQuery, 235
объект , 252
определение, 228
открывающие новое окно, 349
отмена обычного поведения, 254
помощник , 235
удаление, 254
События jQuery
hover(), 249
События документа/окна
load, 231
resize, 232
scroll, 232
unload, 232
События клавиатуры
keydown, 235
keypress, 234
keyup, 235
События мыши
click, 230, 383
dblclick, 230
mousedown, 230
mousemove, 231
mouseout, 231
mouseover, 231
mouseup, 231
События форм
blur, 234, 382
change, 233, 384
focus, 233, 380
reset, 233
submit, 233, 378
Создание нового окна браузера, 344
Соседние родственные селекторы, 194
Специфичность в CSS, 559
Список дел, создание приложения, 697
Ссылки
выборка с помощью языка JavaScript, 336
использование ссылки на окно, 347
определение направления, 337
открытие внешних, 340
предотвращение перехода по, 338
управление поведением, 335
Строка запроса, 654
Строки, 71
извлечение части, 761
изменение регистра, 758
конкатенация, 83
нахождение масок в, 763
определение длины, 756
поиск в, 759
пустая строка, 91
работа с, 756
разбиение, 812

T

Табуляция в языке JavaScript, 79
Теги-предки, 256
Текстовый редактор, 52

Телефонный номер, регулярное выражение для, 773
Тема, добавление новой на сайт, 555
Тернарный оператор, 808
Тип данных, 70
  логический (булев), 73
  строка, 71
  число, 70
Тип документа (doctype), 22
Типы HTML, 22
Точечный синтаксис, 108, 688
Точка останова, 843
; (точка с запятой)
  в блоке объявления, 27
  в утверждениях, 68
  для связывания функций jQuery, 200
Точка с запятой (;)
  в блоке объявления, 27
  в утверждениях, 68
  для связывания функций jQuery, 200
Триггер, 458

У

<> угловые скобки, 23
Угловые скобки (<>), 23
Узлы в DOM, 187
Универсальное координированное время (UTC), 802
Унифицированный указатель ресурсов (URL), 22
Управление поведением ссылок, 335
Уровень защищенных сокетов (SSL), 817
Условные выражения, 115
  else if, 124
  if, 117
  булевы значения, 121
  вложение, 131
  использование, 133
  оператор И (&&) в, 128
  оператор ИЛИ (||) в, 128
  оператор НЕ (!) в, 130
  операторы-переключатели и, 127
  операторы сравнения, 118
  оценка условия, 122
  решение проблем, 827
  советы по написанию, 132
  условие, 117
  части, 117

Ф

{} фигурные скобки
  в блоке объявления стиля CSS, 27
  в условном выражении, 117
  в функциях, 149
  в циклах, 139
  пропущенная закрывающая скобка, 823
Фигурные скобки ({})
  в блоке объявления стиля CSS, 27
  в пользовательских функциях, 149
  в условном выражении, 117
  в формате JSON, 673
  отсутствие закрывающей скобки, 821
Фид, 682
Фильтры jQuery, 196
  checked, 374
  contains(), 198
  even, 196
  first, 197
  has(), 197
  hidden, 198
  last, 197
  not(), 197
  odd, 196
  selected, 374
  visible, 198
Флаг, булево значение, 122
Флажки
  верификация, 424
  проверка наличия меток, 377
  улучшение, 525
Форма, 369
  автозаполнение, 527
  активное состояние поля, 380
  верификация, 399
  верификация с помощью сервера, 413
  включение полей, 387
  выбор элементов, 372
  выключение полей, 392
  настройка кнопок, 523
  неактивное состояние поля, 382
  отправка, 378
  плагин sValidation, 400
  получение и установка значений элементов, 375
  предотвращение многократной отправки, 389
  скрывание и отображение параметров, 388
  скрывание полей, 396
  события, 378
  стилизация кнопок, 521
  структура, 369
  улучшение переключателей и флажков, 525
  усовершенствование, 385
  фокус на первом поле, 385
Фотогалерея с эффектами, руководство, 327
Функция
  вызов, 150
  область видимости, 158
  обратного вызова, 640
  оператор возврата, 155, 339
  определение, 148
  параметр, 152
  передача аргумента, 153
Функция JavaScript
  alert(), 53, 54, 69, 241, 259
  ceil(), 789

charAt(), 113
document.getElementById(), 187
document.getElementsByClassName(), 188
document.getElementsByTagName(), 188
document.write(), 56, 89, 107, 136
encodeURIComponent(), 656
floor(), 791
getData(), 726
getDate(), 793
getDay(), 793, 794
getFullYear(), 793
getHours(), 793
getMinutes(), 793
getMonth(), 793
getSeconds(), 793
getTime(), 793, 795
isNaN(), 69, 382, 788
join(), 143
new Date(), 800
Number(), 85, 786
open(), 344
parseFloat(), 787
parseInt(), 137, 786
pop(), 101
processContacts(), 675
processData(), 669
prompt(), 91, 92, 785
push(), 99, 101
querySelectorAll(), 188
random(), 791
replace(), 696
round(), 789
shift(), 101
toLowerCase(), 759
unshift(), 99
встроенная, 69
глобальная переменная в, 159
глобальный контекст, 159
локальная переменная в, 159
Функция jQuery
$(document).ready(), 180
$.getJSON(), 681
$(), jQuery() и, 729
accordion(), 356, 489, 491
addClass(), 208, 581
after(), 204, 752
animate(), 287, 288, 290, 295, 365, 398, 433, 613, 738
append(), 203, 695, 732, 733, 752
appendTo(), 427
attr(), 215, 313, 314, 318, 323, 337, 341, 649
autocomplete(), 529
before(), 204, 752
bind(), 257
button(), 524, 525, 527, 703
buttonset(), 525–527
cancel(), 617
change(), 385
children(), 745
click(), 257, 263, 286, 306, 335, 339, 383, 598, 648, 649, 714
clone(), 206
closest(), 747
css(), 210, 212, 215, 310, 395, 581, 598, 739, 741
data(), 738
datepicker(), 505, 511
dblclick(), 240
delay(), 295
delegate(), 257
destroy(), 616, 617
detach(), 717, 721
dialog(), 444, 445, 449, 451–453, 457
disable(), 617
draggable(), 567, 569, 572, 580, 591
droppable(), 585
each(), 216, 218, 322, 336, 337, 678, 680, 693, 726, 738, 753
effect(), 621–625, 628, 712, 721
empty(), 755
enable(), 617
end(), 751
error(), 664
fadeIn(), 200, 275, 279, 283, 289, 333, 365, 398
fadeOut(), 216, 279
fadeTo(), 279, 292, 743
fadeToggle(), 279, 284, 286
find(), 486, 613, 666, 710, 734, 737, 743, 745
focus(), 386
get(), 652, 664
getJSON(), 674
hide(), 278, 333, 468, 602, 621
hover(), 250, 297, 298, 319, 326
html(), 201, 486, 662, 731, 749
indexOf(), 760, 764
is(), 272
lastIndexOf(), 761
live(), 257
load(), 642, 643, 645, 646
match(), 779, 781
menu(), 497
mouseover(), 257, 263, 598
next(), 270, 748, 749
noConflict(), 729
not(), 342
off(), 254, 255
on(), 257, 721
parent(), 746, 747
post(), 652, 657
prepend(), 203, 670, 718, 752
prev(), 749
preventDefault(), 254, 338, 339, 452
ready(), 246
remove(), 205, 721, 753
removeAttr(), 215

removeClass(), 209, 629
replaceWith(), 315, 752
search(), 764
selectmenu(), 515, 518
send(), 641
serialize(), 617, 658
show(), 277, 621
siblings(), 748
slice(), 761
slideDown(), 273, 280, 398, 718, 738
slideToggle(), 280, 284
slideUp(), 275, 280, 289, 398, 616, 716
sortable(), 612
split(), 813
stop(), 302
stopPropagation(), 256
submit(), 378
switchClass(), 629
tabs(), 472, 475, 482, 485
text(), 200, 202, 659, 710, 731, 749
toArray(), 619
toFixed(), 376, 790
toggle(), 252, 278, 284, 621
toggleClass(), 209, 285, 306
tooltip(), 464, 465, 467
unbind(), 257
unwrap(), 754, 755
val(), 244, 374, 375, 377, 399, 706, 738
validate(), 402, 403, 408, 413, 418
wrap(), 753
wrapInner(), 754
Функция обратного вызова, 276
JavaScript, 641
processContacts(), 679
processResponse(), 661
в приложении Списокдел, 717
статус ответа, 659
Функция объекта Math
ceil(), 789
random(), 791
round(), 789

Ц

Цикл, 139
do/while, 146
for, 144
while, 139
бесконечный, 141

Ч

Числа, 70
восьмеричные, 787
генерация случайных, 791
округление, 789
преобразование строки в, 785
шестнадцатеричные, 787
Чувствительность к регистру
имен переменных, 76
решение проблем, 828

Ш

Шестнадцатеричное число, 787

Э

Эдвард, Дин, автор программы Packer, 817
Экземпляры объекта, 109
Экранирование (кодирование) знаков, 656
Элемент, 184
в DOM, 187
действия над, 185
селектор, 191
установка атрибутов для, 736
Элемент-триггер, 458
Эффекты jQuery
абсолютное позиционирование, 281
выполнение действия после завершения, 291
ключевые слова для задавания скорости эффекта, 277
описание, 276
отображение и скрывание элементов, 277
постепенное появление и исчезновение элементов, 279
руководство, 282
скользящие элементы, 280
фотогалерея с, 327
Эффекты jQuery UI
easing, параметр, 626
анимация изменения класса, 627
описание, 620, 622

Я

Язык программирования
клиентский, 46
компилируемый, 43
серверный, 46
синтаксис, 42
скриптовый, 43
Язык разметки гипертекста (HTML), 22

Все права защищены. Книга или любая ее часть не может быть скопирована, воспроизведена в электронной или механической форме, в виде фотокопии, записи в память ЭВМ, репродукции или каким-либо иным способом, а также использована в любой информационной системе без получения разрешения от издателя. Копирование, воспроизведение и иное использование книги или ее части без согласия издателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Дэвид Макфарланд

JAVASCRIPT И JQUERY
ИСЧЕРПЫВАЮЩЕЕ РУКОВОДСТВО

3-е издание
(орыс тілінде)

Директор редакции *Е. Капьёв*
Ответственный редактор *В. Обручев*
Художественный редактор *Е. Мишина*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86, 8 (495) 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86, 8 (495) 956-39-21
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8(727) 2 51 59 89,90,91,92, факс: 8 (727) 251 58 12 вн. 107; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Оптовая торговля книгами «Эксмо»:
ООО «ТД «Эксмо». 142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел. 411-50-74.
E-mail: **reception@eksmo-sale.ru**
По вопросам приобретения книг «Эксмо» зарубежными оптовыми покупателями *обращаться в отдел зарубежных продаж ТД «Эксмо»*
E-mail: **international@eksmo-sale.ru**
International Sales: International wholesale customers should contact Foreign Sales Department of Trading House «Eksmo» for their orders.
international@eksmo-sale.ru
По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, *обращаться по тел. +7(495) 411-68-59, доб. 2261, 1257.*
E-mail: **ivanova.ey@eksmo.ru**

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить по адресу: http://eksmo.ru/certification/

Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Подписано в печать 07.07.2015.
Формат 70x100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 71,3.
Тираж 2000 экз. Заказ 9374.
Отпечатано в ОАО «Можайский полиграфический комбинат»
143200, г. Можайск, ул. Мира, 93.
www.oaompk.ru, www.оломпк.рф тел.: (495) 745-84-28, (49638) 20-685

ИНТЕРНЕТ-МАГАЗИН
shop.eksmo.ru
shop.eksmo.ru
ЭКСМО

ISBN 978-5-699-79119-4

ЛитРес:

Все ответы здесь!

Язык программирования JavaScript позволяет усовершенствовать ваши веб-страницы с помощью анимации, интерактивных элементов и визуальных эффектов, но его не так просто изучить и освоить. Новое, обновленное и расширенное издание уже ставшей классикой книги доступно объясняет основы языка JavaScript и показывает, как можно экономить время и силы с помощью библиотеки jQuery, содержащей готовые фрагменты кода JavaScript, и новейшего плагина jQuery UI.

Дэвид МакФарланд – президент компании McFarland Media, около 20 лет занимается созданием сайтов и их управлением. Он преподавал в Калифорнийском университете в Беркли и в Портлендском государственном университете. Дэвид написал более 15 книг по компьютерной тематике, в том числе по CSS и JavaScript.

Прочитав эту книгу, вы сможете:

Сделать свои страницы интерактивными. Используйте библиотеку jQuery для создания интерактивных элементов, реагирующих на действия посетителя.

Освоить новейший плагин jQuery UI. Улучшайте интерфейс, используя панели с вкладками, диалоговые окна, панели для выбора дат и другие виджеты.

Создавать удобные формы. Собирайте данные посетителей, помогайте покупателям осуществлять покупки и позволяйте участникам оставлять комментарии.

Применять технологию Ajax. Организуйте обмен данными между веб-страницами и веб-сервером без необходимости перезагрузки страниц.

Углубить свои знания. Используйте редактор ThemeRoller для настройки виджетов. Избегайте типичных ошибок, свойственных начинающим программистам.

Дополнительные материалы к книге можно скачать по адресу: http://eksmo.ru/Primers_js_jq.zip.

ISBN 978-5-699-79119-4

O'REILLY®