O'REILLY®

5-е издание

СОЗДАЕМ ДИНАМИЧЕСКИЕ ВЕБ-САЙТЫ с помощью PHP, MySQL, JavaScript, CSS и HTML5

ПИТЕР®

Робин Никсон

FIFTH EDITION

Learning PHP, MySQL & JavaScript

With jQuery, CSS & HTML5

Robin Nixon

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Робин Никсон

СОЗДАЕМ ДИНАМИЧЕСКИЕ ВЕБ-САЙТЫ с помощью PHP, MySQL, JavaScript, CSS и HTML5

5-е издание

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2019

ББК 32.988-02-018
УДК 004.738.5
Н64

Никсон Робин

Н64 Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 5-е изд. — СПб.: Питер, 2019. — 816 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-0825-1

Новое (5-е) издание признанного бестселлера, описывающее как клиентские, так и серверные аспекты веб-разработки. Эта книга поможет вам освоить динамическое веб-программирование с применением самых современных технологий. Книга наполнена ценными практическими советами, содержит подробный теоретический материал. Для закрепления материала автор рассказывает, как создать полнофункциональный сайт, работающий по принципу социальной сети, включая рассказ о React.js.

- Изучите важнейшие аспекты языка PHP и основы объектно-ориентированного программирования.
- Познакомьтесь с базой данных MySQL.
- Управляйте cookie-файлами и сеансами, обеспечивайте высокий уровень безопасности.
- Пользуйтесь фундаментальными возможностями языка JavaScript.
- Применяйте вызовы AJAX, чтобы значительно повысить динамику вашего сайта.
- Изучите основы CSS для форматирования и оформления ваших страниц.
- Освойте продвинутые возможности HTML5: геолокацию, обработку аудио и видео, отрисовку на холсте.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988-02-018
УДК 004.738.5

ISBN 978-1491978917 англ.

Authorized Russian translation of the English edition Learning PHP, MySQL & JavaScript, 5th Edition ISBN 9781491978917 © 2018 Robin Nixon
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same

ISBN 978-5-4461-0825-1

Краткое содержание

Оглавление

Предисловие

Сочетание PHP и MySQL — один из самых удобных подходов к динамическому веб-конструированию, основанному на использовании баз данных. Этот подход удерживает свои позиции, несмотря на вызовы, брошенные интегрированными средами разработки, такими как Ruby on Rails, освоение работы с которыми дается значительно труднее. Благодаря открытости исходных кодов (в отличие от конкурирующей технологии Microsoft .NET Framework) это технологическое сочетание можно использовать совершенно бесплатно, поэтому оно очень популярно у веб-разработчиков.

Любой нацеленный на результативность разработчик, использующий платформу UNIX/Linux или даже Windows/Apache, нуждается в серьезном изучении этих технологий. В сочетании с партнерскими технологиями JavaScript, jQuery, CSS и HTML5 можно создавать сайты калибра таких промышленных стандартов, как Facebook, Twitter и Gmail.

Для кого предназначена эта книга

Книга предназначена для тех, кто хочет изучить способы создания эффективных и динамичных сайтов. Сюда можно отнести веб-мастеров или специалистов по графическому дизайну, которым уже приходилось создавать статические сайты и у которых есть желание вывести свое мастерство на следующий уровень, а также студентов вузов и колледжей, недавних выпускников этих учебных заведений и просто самоучек.

Фактически любой человек, стремящийся изучить основные принципы, заложенные в основу адаптивного веб-дизайна, сможет получить весьма обстоятельные сведения об основных технологиях: PHP, MySQL, JavaScript, CSS и HTML5, а также изучить основы библиотек jQuery и jQuery Mobile.

Предположения, допущенные в книге

При написании данной книги автор предполагал, что читатель уже имеет элементарные понятия об HTML и способен как минимум скомпоновать простой статический сайт. Но при этом не обязательно наличие у читателя каких-либо знаний в области PHP, MySQL, JavaScript, CSS и HTML5, хотя, если такие знания имеются, изучение материала будет происходить значительно быстрее.

Структура издания

Главы книги расположены в определенном порядке. Сначала идет представление всех основных технологий, рассматриваемых в книге, а затем описывается процесс их установки на сервер, предназначенный для разработки веб-приложений, для того чтобы подготовить читателя к практической работе с примерами.

В первой части книги преподносятся основы языка программирования PHP, включая основы синтаксиса, массивов, функций и объектно-ориентированного программирования.

Затем, после усвоения основ PHP, можно переходить к введению в систему управления базами данных MySQL, рассмотрение которой начинается с изучения структуры базы данных MySQL и заканчивается составлением сложных запросов.

После этого рассказывается о том, как воспользоваться сочетанием PHP и MySQL, чтобы приступить к созданию собственных динамических веб-страниц путем интегрирования в это сочетание форм и других функциональных возможностей HTML. Затем приводятся подробности практических аспектов разработки на PHP и MySQL, включая описание различных полезных функций и способов работы с cookies и сессиями, а также способов поддержания высокого уровня безопасности.

В следующих нескольких главах излагаются основы JavaScript, начиная с простых функций и обработки событий и заканчивая доступом к объектной модели документа (DOM), проверкой введенных данных и обработкой ошибок в браузере. Этот учебник подойдет для тех, кто приступает к использованию популярной библиотеки jQuery для JavaScript.

После рассмотрения основных технологий описываются способы создания фоновых AJAX-вызовов и превращения сайтов в высокодинамичную среду.

После этого вам предстоит освоить еще две главы, в которых рассматривается, как использовать CSS для стилевого оформления и подбора формата ваших веб-страниц, перед тем будут раскрыты методы существенного снижения трудозатрат по разработке приложений с помощью библиотек jQuery, и дано описание интерактивных свойств, встроенных в HTML5 (геолокация, аудио, видео, холст).

Получив все эти сведения, вы сможете создать полноценный набор программ, в совокупности представляющий собой работоспособный сайт социальной сети.

По мере изложения материала дается большое количество указаний и советов по выработке хорошего стиля программирования, а также подсказок, которые помогут читателям обнаружить и устранить скрытые ошибки программирования. Кроме того, приводится много ссылок на сайты с дополнительными материалами, относящимися к рассматриваемым темам.

Дополнительная литература

Приступив к изучению разработки программных продуктов с помощью PHP, MySQL, JavaScript, CSS и HTML5, вы наверняка будете готовы к переводу своего мастерства на новый уровень, если воспользуетесь следующими книгами, которые можно найти на сайте издательства O'Reilly или на других сайтах, где продаются книги:

- *Dynamic HTML*: *The Definitive Reference* (http://oreil.ly/dynamic_html) Денни Гудмана (Danny Goodman);
- *PHP in a Nutshell* (http://oreil.ly/PHP_nutshell) Пола Хадсона (Paul Hudson);
- *MySQL in a Nutshell* (http://oreil.ly/MySQL_nutshell) Рассела Дайера (Russell Dyer);
- *JavaScript*: *The Definitive Guide* (http://oreil.ly/JS_Defi nitive) Дэвида Фланагана (David Flanagan);
- *CSS*: *The Definitive Guide* (http://oreil.ly/CSS_Defi nitive) Эрика А. Майера (Eric A. Mayer) и Эстель Вейль (Estelle Weyl);
- *HTML5*: *The Missing Manual* Мэтью Макдональда (Matthew MacDonald).

Условные обозначения

В книге применяются следующие условные обозначения.

Шрифт для названий

Используется для обозначения URL, адресов электронной почты, названий папок, а также сочетаний клавиш и названий элементов интерфейса.

`Шрифт для команд`

Используется для имен файлов, названий путей, имен переменных и команд. К примеру, путь будет выглядеть так: `/Developer/Applications`.

`Шрифт для листингов`

Применяется для отображения примеров исходного кода и содержимого файлов.

`Шрифт для листингов полужирный`

Обозначает текст, который выводится программой. Кроме того, данный шрифт иногда применяется для создания логического ударения, например, чтобы выделить важную строку кода в большом примере.

Шрифт для листингов курсивный

Обозначает код, который должен быть заменен подходящим значением (например, *имя_пользователя*).

Вам нужно обращать особое внимание на специальные врезки, выделенные с помощью следующих рисунков.

Это подсказка, пожелание, заметка общего типа. Содержит полезную прикладную информацию по рассматриваемой теме.

Это предостережение или указание, говорящее о том, что вам необходимо быть внимательными.

Использование примеров кода

Эта книга предназначена для оказания помощи в выполнении стоящих перед вами задач. Вы можете использовать код, приведенный в ней, в своих программах и документации. Вам не нужно обращаться к нам за разрешением, до тех пор пока вы не станете копировать значительную часть кода. Например, использование при написании программы нескольких фрагментов кода, взятых из данной книги, не требует специального разрешения. Но продажа и распространение компакт-диска с примерами из книг издательства O'Reilly — требует. Ответы на вопросы, в которых упоминаются материалы этой книги, и цитирование приведенных в ней примеров не требуют разрешения. Но включение существенного объема примеров кода, приводимых в данной книге, в документацию по вашему собственному продукту *требует* получения разрешения.

У этой книги имеется сопутствующий сайт, доступный по адресу http://lpmj.net, откуда вы можете скачать все приведенные в ней примеры одним ZIP-файлом.

Ссылки на источник приветствуются, но не обязательны. В такие ссылки обычно включаются название книги, имя ее автора, название издательства и номер ISBN. Например: «Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5», пятое издание, автор Робин Никсон (Robin Nixon). Copyright 2018 Robin Nixon, 978-5-4461-0825-1.

При любых сомнениях относительно превышения разрешенного объема использования примеров кода, приведенных в данной книге, можете свободно обращаться к нам по адресу permissions@oreilly.com.

Благодарности

Хочу еще раз поблагодарить своего редактора Энди Орама и всех, кто приложил немало усилий для выхода этой книги, в том числе Джона Рейда, Майкла Шпачека и Джона Крейга, за их всеобъемлющую техническую рецензию, Мелани Ярбро — за общее руководство выпуском книги, Рейчел Хед — за редактуру, Рейчел Монаган — за корректуру, Ребекку Демарест — за иллюстрации, Джуди Макконвилл — за создание указателя, Карен Монтгомери — за великолепную сахарную сумчатую летягу на обложке книги, Рэнди Камера — за последний вариант обложки книги, а также многочисленных помощников, отправивших сведения о замеченных ошибках и высказавших свои предложения относительно этого нового издания.

1 Введение в динамическое содержимое веб-страницы

Всемирная паутина — это непрерывно развивающаяся сеть, ушедшая далеко вперед от своей концепции ранних 1990-х, когда ее создание было обусловлено решением конкретных задач. Высокотехнологичные эксперименты в ЦЕРНе (Европейском центре физики высоких энергий, известном в наши дни в качестве обладателя Большого адронного коллайдера) выдавали невероятно большой объем данных, который был слишком велик для распространения среди участвующих в экспериментах ученых, разбросанных по всему миру.

К тому времени Интернет уже существовал и к нему было подключено несколько сотен тысяч компьютеров, поэтому Тим Бернерс-Ли (специалист ЦЕРНа) придумал способ навигации между ними с использованием среды гиперссылок — так называемого протокола передачи гиперссылок (Hyper Text Transfer Protocol, HTTP). Он также создал специальный язык разметки, названный языком гипертекстовой разметки (Hyper Text Markup Language, HTML). Для того чтобы собрать все это воедино, он создал первые браузер и веб-сервер.

Теперь эти средства воспринимаются нами как должное, но в то время концепция их применения носила революционный характер. До этого основной объем соединений приходился на пользователей домашних модемов, дозванивавшихся и подключавшихся к электронным доскам объявлений, которые базировались на отдельном компьютере и позволяли общаться и обмениваться данными только с другими пользователями этой службы. Следовательно, для эффективного электронного общения с коллегами и друзьями нужно было становиться участником многих электронных досок объявлений.

Но Бернерс-Ли изменил все это одним махом, и к середине 1990-х годов уже существовали три основных конкурирующих друг с другом графических браузера, пользовавшихся вниманием 5 млн посетителей. Однако вскоре стало очевидно,

что кое-что было упущено. Конечно, текстовые и графические страницы, имеющие гиперссылки для перехода на другие страницы, были блестящей концепцией, но результаты не отражали текущий потенциал компьютеров и Интернета по удовлетворению насущных потребностей пользователей в динамическом изменении контекста. Всемирная паутина оставляла весьма невыразительное впечатление, даже при наличии прокрутки текста и анимированных GIF-картинок. Корзины покупателей, поисковые машины и социальные сети внесли существенные коррективы в порядок использования Всемирной паутины. В этой главе будет дан краткий обзор различных компонентов, формирующих ее облик, и программного обеспечения, способствующего обогащению и оживлению наших впечатлений от ее использования.

Пришло время воспользоваться аббревиатурами. Прежде чем делать это, я старался дать им четкое объяснение. Но если сразу не удастся разобраться, какое именно понятие они замещают или что означают, переживать не стоит, поскольку все подробности прояснятся по мере чтения книги.

HTTP и HTML: основы, заложенные Бернерсом-Ли

HTTP представляет собой стандарт взаимодействия, регулирующий порядок направления запросов и получения ответов — процесса, происходящего между браузером, запущенным на компьютере конечного пользователя, и веб-сервером. Задача сервера состоит в том, чтобы принять запрос от клиента и попытаться дать на него содержательный ответ, обычно передавая ему запрошенную веб-страницу. Именно поэтому и используется термин «*сервер*» («обслуживающий»). Партнером, взаимодействующим с сервером, является *клиент*, поэтому данное понятие применяется как к браузеру, так и к компьютеру, на котором он работает.

Между клиентом и сервером может располагаться ряд других устройств, например маршрутизаторы, модули доступа, шлюзы и т. д. Они выполняют различные задачи по обеспечению безошибочного перемещения запросов и ответов между клиентом и сервером. Как правило, для отправки этой информации используется Интернет. Некоторые из этих промежуточных устройств могут также ускорить Интернет путем локального сохранения страниц или информации в так называемом кэше, обслуживая затем данное содержимое для клиентов непосредственно из кэша без постоянного извлечения его из сервера-источника.

Обычно веб-сервер может обрабатывать сразу несколько подключений, а при отсутствии связи с клиентом он находится в режиме ожидания входящего подключения. При поступлении запроса на подключение сервер подтверждает его получение отправкой ответа.

Процедура «запрос — ответ»

В наиболее общем виде процесс «запрос — ответ» состоит из просьбы браузера к веб-серверу отправить ему веб-страницу и выполнения браузером данной просьбы. После этого браузер занимается отображением страницы (рис. 1.1).

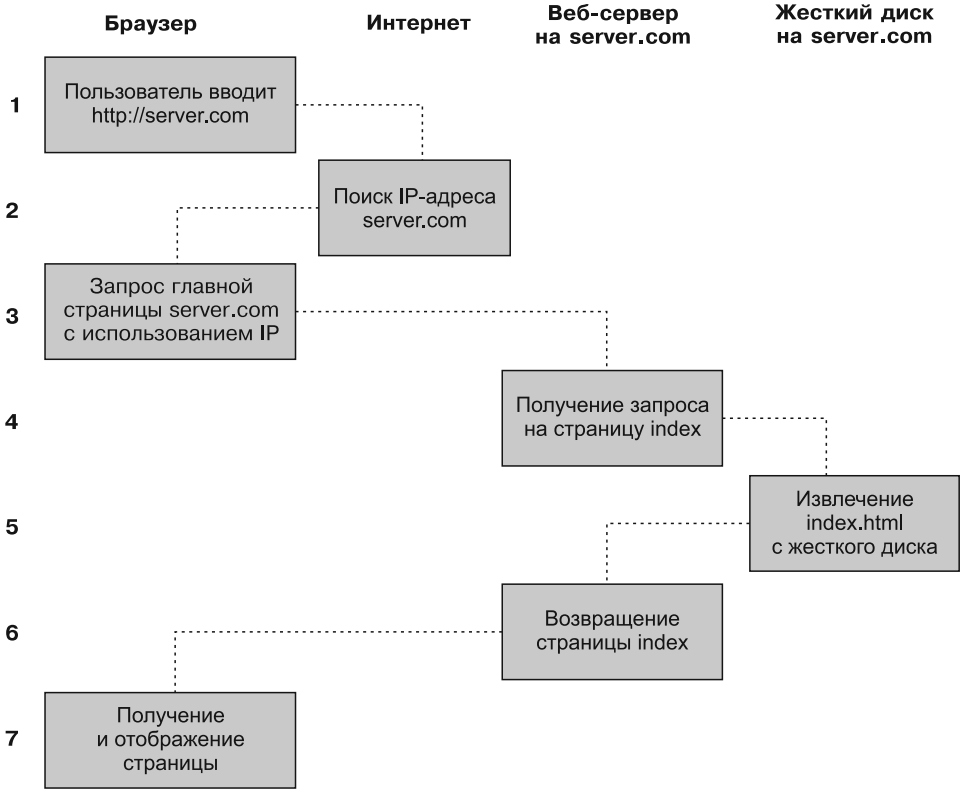

Рис. 1.1. Основная последовательность процесса «запрос — ответ» между клиентом и сервером

При этом соблюдается такая последовательность действий.

1. Вы вводите в адресную строку браузера `http://server.com`.
2. Ваш браузер ищет IP-адрес, соответствующий доменному имени server.com.
3. Браузер посылает запрос на главную страницу server.com.
4. Запрос проходит по Интернету и поступает на веб-сервер server.com.
5. Веб-сервер, получивший запрос, ищет веб-страницу на своем жестком диске. Сервер извлекает веб-страницу и отправляет ее по обратному маршруту в адрес браузера.
6. Браузер отображает веб-страницу.

При передаче типовой веб-страницы этот процесс также осуществляется для каждого имеющегося на ней объекта: элемента графики, встроенного видео- или Flash-ролика и даже шаблона CSS.

Обратите внимание на то, что на шаге 2 браузер ищет IP-адрес, принадлежащий доменному имени server.com. У каждой машины, подключенной к Интернету, включая и ваш компьютер, есть свой IP-адрес. Но, как правило, доступ к веб-серверам осуществляется по именам, таким как google.com. Вам должно быть известно, что браузер обращается к вспомогательной интернет-службе, так называемой службе доменных имен (Domain Name Service, DNS), чтобы найти связанный с сервером IP-адрес, а затем воспользоваться им для связи с компьютером.

При передаче динамических веб-страниц процедура состоит из большего количества действий, поскольку к ней могут привлекаться как PHP, так и MySQL. Например, можно щелкнуть кнопкой мыши на картинке с изображением плаща. После этого PHP составит запрос, используя стандартный язык базы данных, SQL — множество используемых для этого команд будет рассмотрено в книге, — и отправит запрос в адрес MySQL-сервера. Этот сервер возвратит информацию о выбранном вами плаще, и PHP-код заключит ее в некий код HTML, который будет отправлен сервером в адрес вашего браузера (рис. 1.2).

Выполняется такая последовательность действий.

1. Вы вводите в адресную строку браузера `http://server.com`.
2. Ваш браузер ищет IP-адрес, соответствующий доменному имени server.com.
3. Браузер посылает запрос на главную страницу server.com. Запрос проходит по Сети и поступает на веб-сервер server.com.
4. Веб-сервер, получивший запрос, ищет веб-страницу на своем жестком диске.
5. Теперь, когда главная страница размещена в его памяти, веб-сервер замечает, что она представлена файлом, включающим в себя PHP-сценарии, и передает страницу интерпретатору PHP.
6. Интерпретатор PHP выполняет PHP-код.
7. Кое-какие фрагменты кода PHP содержат MySQL-инструкции, которые интерпретатор PHP, в свою очередь, передает процессору базы данных MySQL.
8. База данных MySQL возвращает результаты выполнения инструкции интерпретатору PHP.
9. Интерпретатор PHP возвращает веб-серверу результаты выполнения кода PHP, а также результаты, полученные от базы данных MySQL.
10. Веб-сервер возвращает страницу выдавшему запрос клиенту, который отображает эту страницу на экране.

Конечно, ознакомиться с этим процессом и узнать о совместной работе трех элементов не помешает, но на практике эти подробности не понадобятся, поскольку все происходит в автоматическом режиме.

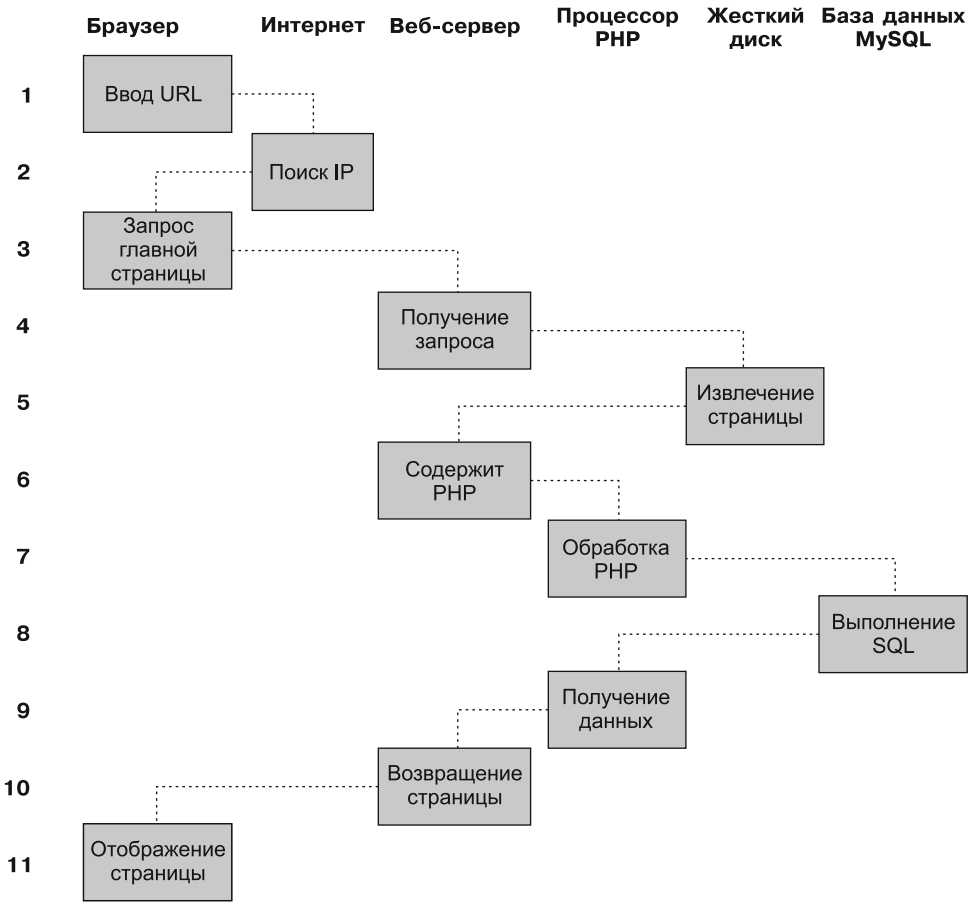

Рис. 1.2. Динамическая последовательность процесса «запрос — ответ», выполняемого клиентом и сервером

В каждом из примеров возвращенные браузеру HTML-страницы могут содержать также код JavaScript, интерпретируемый локально на машине клиента. Этот код может инициировать еще один запрос, точно так же запрос может быть инициирован встроенными объектами, например изображениями.

Преимущества использования PHP, MySQL, JavaScript и CSS

В начале этой главы был представлен мир технологии Web 1.0, но рывок к созданию технологии Web 1.1, вместе с которой были разработаны такие браузерные расширения, как Java, JavaScript, JScript (несколько иной вариант JavaScript от корпорации Microsoft) и ActiveX, не заставил себя долго ждать. На серверной стороне

прогресс был обеспечен за счет общего шлюзового интерфейса (Common Gateway Interface, CGI), использования таких языков сценариев, как Perl (альтернатива языку PHP), и выполнения *сценариев на стороне сервера* — динамической вставки содержимого одного файла (или выходных данных выполняемой локальной программы) в другой файл.

Когда ситуация окончательно прояснилась, на передовых позициях остались три основные технологии. Несмотря на то что язык сценариев Perl силами своих стойких приверженцев сохранил популярность, простота PHP и допустимость использования в нем встроенных ссылок на программу базы данных MySQL обеспечили этому языку более чем двойное превосходство по количеству пользователей. А JavaScript, ставший важнейшей составной частью уравнения, используемого для динамического манипулирования каскадными таблицами стилей (Cascading Style Sheets, CSS) и HTML, в настоящее время берет на себя наиболее трудоемкие задачи асинхронного обмена данными (осуществляемого между клиентом и сервером после загрузки веб-страницы). Используя асинхронный обмен данными, веб-страницы обрабатывают данные и отправляют запросы веб-серверу в фоновом режиме, не оповещая пользователя о происходящем.

Несомненно, своеобразный симбиоз PHP и MySQL способствует их продвижению, но что привлекает к ним разработчиков в первую очередь? На это следует дать простой ответ: та легкость, с которой эти технологии можно использовать для быстрого создания на сайтах динамических элементов. MySQL является быстродействующей и мощной, но при этом простой в использовании системой базы данных, предлагающей сайту практически все необходимое для поиска и обработки данных, которые предназначены для браузеров. Когда PHP для хранения и извлечения этих данных выступает в союзе с MySQL, вы получаете основные составляющие, необходимые для разработки сайтов социальных сетей и для перехода к технологии Web 2.0.

И когда вы также соедините вместе JavaScript и CSS, у вас появится рецепт для создания высокодинамичных и интерактивных сайтов, особенно в современных условиях доступности множества сложных функциональных сред JavaScript, вызов которых действительно позволяет ускорить веб-разрабоку. Достаточно упомянуть широко известную библиотеку JQuery, применение которой в настоящее время, наверное, является наиболее востребованным способом доступа программистов к функциям асинхронного обмена данными.

MariaDB: клон MySQL

После того как компания Oracle приобрела Sun Microsystems (владельца MySQL), возникли опасения, что полностью открытым код MySQL может не остаться, и в итоге от этой СУБД отпочковалась MariaDB, дабы код оставался открытым в соответствии с положениями лицензии GNU GPL. Разработка MariaDB шла под

руководством ряда первоначальных создателей MySQL, и эта СУБД сохранила максимальную совместимость с MySQL. Поэтому вероятность встречи на некоторых серверах MariaDB вместо MySQL весьма высока, что не должно вызывать никаких опасений, поскольку все показанное в данной книге одинаково успешно работает как под MySQL, так и под MariaDB, имеющей туже кодовую базу, что и MySQL Server 5.5. В любом случае одна СУБД может заменяться другой, и вы при этом не заметите никакой разницы.

Впрочем вышло так, что многие возникшие поначалу опасения были напрасны, поскольку код MySQL остался открытым, а компания Oracle просто сделала платной приобретение и поддержку тех выпусков, которые предоставляют дополнительные функциональные возможности, включающие георепликацию и автоматическое масштабирование. Тем не менее, в отличие от MariaDB, MySQL больше не поддерживается сообществом, но осознание того, что MariaDB никогда не лишится этой поддержки, позволит многим разработчикам спать спокойно и, вероятно, даст гарантии того, что код самой MySQL останется открытым.

PHP

Использование PHP существенно упрощает встраивание средств, придающих веб-страницам динамические свойства. Когда страницам присваивается расширение PHP, у них появляется прямой доступ к языку сценариев. Разработчику нужно лишь написать код, похожий на этот:

```
<?php
  echo " Today is " . date('T') . ". ";
?>
Here's the latest news.
```

Открывающий тег `<?php` дает веб-серверу разрешение на интерпретацию всего последующего кода вплоть до тега `?>`. Все, что находится за пределами этой конструкции, отправляется клиенту в виде простого HTML. Поэтому текст `Here's the latest news` просто выводится в браузер. А внутри PHP-тегов встроенная функция `date` отображает текущий день недели, соответствующий системному времени сервера.

В итоге на выходе из этих двух частей получается примерно следующее:

```
Today is Wednesday. Here's the latest news.
```

PHP — довольно гибкий язык, и некоторые разработчики предпочитают помещать PHP-конструкцию непосредственно рядом с кодом PHP, как в этом примере:

```
Today is <?php echo date("l"); ?>. Here's the latest news.
```

Существуют также другие способы форматирования и вывода информации, которые будут рассмотрены в главах, посвященных PHP. Важно усвоить то,

что, используя PHP, веб-разработчики получают язык сценариев, который хотя и не обладает быстротой кода, скомпилированного на C или ему подобных языках, но все же работает невероятно быстро и к тому же очень хорошо вписывается в разметку HTML.

Если вы собираетесь набирать встречающиеся в этой книге примеры на PHP в программе-редакторе, чтобы работать параллельно с моим повествованием, не забывайте предварять их тегом <?php, а в конце ставить тег ?>, для того чтобы обеспечить их обработку интерпретатором PHP. Для упрощения этой задачи можно заранее подготовить файл example.php, содержащий эти теги.

Используя PHP, вы получаете средство управления своим веб-сервером с неограниченными возможностями. Если понадобится на лету внести изменения в HTML, обработать данные кредитной карты, добавить сведения о пользователе в базу данных или извлечь информацию из стороннего сайта, все это можно будет сделать из тех же самых PHP-файлов, в которых находится и сам код HTML.

MySQL

Разумеется, без средств отслеживания информации, предоставляемой пользователем в ходе работы с вашим сайтом, нельзя в полной мере говорить о возможностях динамического изменения выходного кода HTML. На заре создания Всемирной паутины многие сайты использовали неструктурированные текстовые файлы для хранения таких данных, как имена пользователей и пароли. Но подобный подход мог вызвать ряд проблем, если файл не был надежно заблокирован от повреждений, возникающих при одновременном доступе к нему множества пользователей. К тому же неструктурированный файл мог разрастаться до таких размеров, что с ним непросто было работать, не говоря уже о трудностях, связанных с попытками объединения файлов и осуществления в них сложных поисковых операций за какое-нибудь мало-мальски приемлемое время.

Именно в таких случаях большое значение приобретает использование реляционных баз данных со структурированной системой запросов. И MySQL, будучи совершенно бесплатной и установленной на огромном количестве веб-серверов системой, оказывается как нельзя кстати. Она представляет собой надежную и исключительно быстродействующую систему управления базами данных, использующую команды, похожие на простые английские слова.

Высшим уровнем структуры MySQL является база данных, внутри которой можно иметь одну или несколько таблиц, содержащих ваши данные. Предположим, вы работаете над таблицей под названием `users` (пользователи), внутри которой были созданы столбцы для фамилий — `surname`, имен — `firstname` и адресов электронной

почты — email, и теперь нужно добавить еще одного пользователя. Одна из команд, которую можно применить для этого, выглядит следующим образом:

```
INSERT INTO users VALUES('Smith', 'John', 'jsmith@mysite.com');
```

Для создания базы данных и таблицы, а также настройки всех нужных полей понадобится сначала выдать и другие команды, но используемая здесь SQL-команда INSERT демонстрирует простоту добавления в базу данных новой информации. Structured Query Language (SQL) является языком, разработанным в начале 1970-х годов и напоминающим один из старейших языков программирования — COBOL. Тем не менее он хорошо подходит для запросов к базе данных, что и предопределило его использование в течение столь длительного времени.

Так же просто выполняется и поиск данных. Предположим, что имеется адрес электронной почты пользователя и нужно найти имя его владельца. Для этого можно ввести следующий запрос MySQL:

```
SELECT surname,firstname FROM users WHERE email='jsmith@mysite.com';
```

После этого MySQL вернет Smith, John и любые другие пары имен, которые могут быть связаны в базе данных с адресом электронной почты.

Нетрудно предположить, что возможности MySQL простираются значительно дальше выполнения простых команд вставки и выбора — INSERT и SELECT. Например, можно скомбинировать родственные наборы данных, чтобы собрать вместе взаимосвязанные части информации, запросить результаты, выбрав порядок их выдачи из множества вариантов, найти частичные совпадения, если известна только часть искомой строки, вернуть конкретно заданное количество результатов и сделать многое другое.

При использовании PHP все эти вызовы можно направлять непосредственно к MySQL без необходимости самостоятельного применения ее интерфейса командной строки. Это значит, что для того, чтобы докопаться до нужного вам элемента данных, вы можете сохранять результаты в массивах для их обработки и осуществления множества поисковых операций, каждая из которых зависит от результатов, возвращенных предыдущими операциями.

Далее будет показано, что для придания еще большей мощности прямо в MySQL встроено несколько дополнительных функций, которые можно вызвать с целью эффективного выполнения наиболее часто встречающихся в MySQL операций, избавляясь от необходимости их составления из нескольких PHP-вызовов в адрес MySQL.

JavaScript

Самая старая из трех основных технологий, рассматриваемых в данной книге, — JavaScript — была создана для получения доступа из сценариев ко всем элементам HTML-документа. Иными словами, она предоставляет средства для динамического

взаимодействия с пользователем, например для проверки приемлемости адресов электронной почты в формах ввода данных, отображения подсказок наподобие «Вы действительно подразумевали именно это?» и т. д. (хотя с точки зрения безопасности, которая всегда должна реализовываться на веб-сервере, на эту технологию положиться нельзя).

В сочетании с CSS JavaScript закладывает основу мощности динамических веб-страниц, которые изменяются буквально на глазах, в отличие от новой страницы, возвращаемой сервером.

Тем не менее с использованием JavaScript могут возникнуть осложнения, обусловленные некоторыми существенными различиями в способах реализации этого языка, выбранных разными разработчиками браузеров. В основном эти различия возникают, когда некоторые производители пытаются придать своим браузерам дополнительные функциональные возможности, не обращая внимания на совместимость с продуктами своих конкурентов.

К счастью, разработчики в большинстве своем уже взялись за ум, и теперь оптимизация вашего кода для различных браузеров утратила прежнюю актуальность. Но остаются миллионы экземпляров устаревших браузеров, которыми будут пользоваться на протяжении еще многих лет. Тем не менее и для них существуют решения проблем несовместимости, и позже в этой книге будут рассмотрены библиотеки и технологии, позволяющие без каких-либо опасений игнорировать существующие различия.

А сейчас взглянем на то, как можно воспользоваться обычным JavaScript-кодом, воспринимаемым всеми браузерами:

```
<script type="text/javascript">
  document.write("Today is " + Date() );
</script>
```

Этот фрагмент кода предписывает браузеру интерпретировать все, что находится внутри тегов `<script>`, в качестве кода JavaScript, что затем браузер и сделает, записав в текущий документ текст `"Today is "`, а также дату, полученную за счет использования принадлежащей JavaScript функции `Date`. В результате получится нечто подобное следующему:

```
Today is Sun Jan 01 2017 01:23:45
```

Если не требуется указывать конкретную версию JavaScript, то, как правило, можно опустить type="text/javascript" и использовать для начала интерпретации JavaScript тег <script>.

Ранее было упомянуто, что изначально JavaScript разрабатывался для того, чтобы получить возможность динамического управления различными элементами, нахо-

дящимися внутри HTML-документа, и это его предназначение по-прежнему является основным. Но все чаще JavaScript применяется для осуществления процесса доступа к веб-серверу в фоновом режиме с целью *асинхронного обмена данными*.

Асинхронный обмен данными позволил веб-страницам стать похожими на автономные программы, поскольку для отображения нового содержимого их уже не нужно загружать целиком. Вместо этого асинхронный вызов может использоваться для извлечения и обновления отдельно взятого элемента веб-страницы, например может быть изменена ваша фотография на сайте социальной сети или заменена кнопка, на которой нужно щелкнуть, отвечая на вопрос. Полностью эта тема будет рассмотрена в главе 17.

Затем в главе 21 мы присмотримся к среде jQuery, которую можно использовать, чтобы не изобретать колесо в случае возникновения потребностей в быстродействующем, кросс-браузерном коде для управления веб-страницами. Конечно, доступны и другие подобные среды, но jQuery является самой популярной библиотекой и, исходя из показателей, накопленных за весьма длительный срок ее использования, обладает исключительной надежностью и является основным средством в арсенале многих опытных разработчиков.

CSS

CSS является ключевым дополнением к HTML, обеспечивая соответствующую разметку HTML-текста и встроенных изображений, сообразуясь с параметрами экрана пользователя. После появления третьего стандарта (CSS3) CSS предлагает уровень динамической интерактивности, которая прежде поддерживалась только с помощью JavaScript. Например, вы можете не только придать стиль любому элементу HTML, чтобы изменить его размеры, цвета, границы, интервалы, но и, используя всего лишь несколько строк CSS, добавить своим веб-страницам анимированные переходы и преобразования.

Применение CSS может просто заключаться во вставке правил между тегами `<style>` и `</style>`, расположенными в заголовке веб-страницы:

```
<style>
  p{
    text-align:justify;
    font-family:Helvetica;
  }
</style>
```

Эти правила будут изменять исходное выравнивание текста тега `<p>`, чтобы содержащиеся в нем абзацы были полностью выровнены и для них использовался шрифт Helvetica.

В главе 18 вы увидите, что существует множество различных способов задания правил CSS и их также можно включать непосредственно в теги или сохранять во

внешнем файле, предназначенном для отдельной загрузки. Такая гибкость позволяет проводить точную настройку стиля HTML. Вы также увидите, как с помощью CSS можно, например, создать встроенную функцию `hover` для анимирования объектов при проходе над ними указателя мыши. Кроме того, вы научитесь получать доступ ко всем свойствам CSS-элемента из JavaScript и из HTML.

А теперь HTML5

Какими бы ни были полезными все эти дополнения к веб-стандартам, самым амбициозным разработчикам и их было мало. К примеру, так и не был придуман простой способ работы с графикой в браузере, не требующий обращения к таким дополнительным модулям, как Flash. То же самое происходило и в отношении аудио- и видеовставок в веб-страницы. Кроме того, можно отметить множество досадных несоответствий, вкравшихся в HTML в процессе его развития.

Итак, чтобы подчистить все эти шероховатости, перенести Интернет за пределы технологии Web 2.0 в его следующую фазу развития, был создан новый стандарт HTML, устраняющий перечисленные недостатки: *HTML5*. Он начал разрабатываться в далеком 2004 году, когда Mozilla Foundation и Opera Software (разработчики двух популярных браузеров) составили его первый проект. Но его окончательный проект был представлен World Wide Web Consortium (W3C), международной организацией, руководящей веб-стандартами, лишь в начале 2013 года.

На разработку HTML5 ушло несколько лет, но теперь мы располагаем весьма надежной и стабильной версией 5.1 (с 2016 года). Но цикл разработки никогда не заканчивается, и со временем в этот язык несомненно будут встроены дополнительные функциональные возможности. К некоторым наиболее ярким свойствам HTML5, позволяющим управлять медиаресурсами, можно отнести элементы `<audio>`, `<video>` и `<canvas>`, с помощью которых происходит добавление звука, видео и усовершенствованной графики. Все, что нужно знать об этих и всех других аспектах HTML5, весьма подробно изложено в данной книге, начиная с главы 23.

Кроме всего прочего, мне в спецификации HTML5 нравится, что для самозакрывающихся элементов больше не нужен синтаксис XHTML. Раньше перевод на новую строку можно было изобразить с помощью элемента <br>. Затем для обеспечения совместимости в будущем с XHTML (так и не состоявшейся заменой HTML) элемент был изменен на
 с добавленным символом / (поскольку ожидалось, что характерной особенностью закрывающего тега всех элементов станет именно этот символ). Но теперь все вернулось на круги своя и можно использовать любую из версий таких элементов. Итак, для большей лаконичности и меньшего объема набираемого текста в данной книге я вернулся к прежнему стилю: <br>, <hr> и т. д.

Веб-сервер Apache

В дополнение к PHP, MySQL, JavaScript, CSS и HTML5 в динамической веб-технологии фигурирует и шестой герой — веб-сервер. В нашей книге предполагается, что это веб-сервер Apache. Мы уже немного касались того, что делает веб-сервер в процессе обмена информацией между клиентом и сервером по протоколу HTTP, но на самом деле негласно он выполняет куда более масштабную работу.

Например, Apache обслуживает не только HTML-файлы — он работает с широким спектром файлов, начиная с файлов изображений и Flash-роликов и заканчивая аудиофайлами формата MP3, файлами RSS-потоков (Really Simple Syndication — простое распространение по подписке) и т. д. И эти объекты не должны быть статическими файлами, такими как изображения GIF-формата. Все они могут быть сгенерированы программами, такими как сценарии PHP. И это действительно возможно: PHP способен даже создавать для вас изображения и другие файлы либо на лету, либо заранее, в расчете на последующее обслуживание. Для этого обычно имеются модули, либо предварительно скомпилированные в Apache или PHP, либо вызываемые во время выполнения программы. Одним из таких модулей является библиотека GD (Graphics Draw — рисование графики), которую PHP использует для создания и обработки графических элементов.

Apache поддерживает также обширный арсенал собственных модулей. В дополнение к модулям PHP наиболее важными для вас как для веб-программиста будут модули, занимающиеся обеспечением безопасности. В качестве других примеров могут послужить модуль Rewrite, позволяющий веб-серверу обрабатывать широкий диапазон типов URL-адресов и перезаписывать их в соответствии с его внутренними требованиями, и модуль Proxy, который можно использовать для обслуживания часто запрашиваемых страниц из кэша, чтобы снизить нагрузку на сервер.

Далее в книге будет показано практическое применение этих модулей для улучшения свойств, предоставляемых тремя основными технологиями.

Обслуживание мобильных устройств

Мы живем в уже устоявшемся мире взаимосвязанных мобильных вычислительных устройств, и концепция разработки веб-сайтов исключительно для настольных компьютеров безнадежно устарела. Сегодня разработчики нацелены на создание адаптивных веб-сайтов и веб-приложений, перекраивающих самих себя под ту среду, в которой они оказываются запущенными на выполнение.

Поэтому в качестве нововведения я покажу вам способы рационального создания подобных программных продуктов с использованием только тех технологий, которые подробно рассматриваются в этой книге, а также весьма эффективной

библиотеки jQuery Mobile, обладающей адаптивными JavaScript-функциями. С ее помощью можно будет сконцентрировать все свое внимание на содержимом и удобстве пользования вашими веб-сайтами и веб-приложениями, заранее зная, что способы их отображения на экране будут автоматически оптимизированы под широкую номенклатуру вычислительных устройств, о чем вам меньше всего придется беспокоиться.

С целью демонстрации максимально возможной эффективности использования библиотеки jQuery Mobile в заключительной главе книги будет рассмотрен пример создания сайта простой социальной сети, обладающего полноценной адаптивностью и обеспечивающего качественное отображение на экранах любых устройств, начиная с небольших мобильных телефонов и заканчивая планшетными или настольными компьютерами.

Несколько слов о программах с открытым исходным кодом

Технологии, рассматриваемые в данной книге, основаны на использовании открытого кода: чтение кода и внесение в него изменений носит общедоступный характер. Часто спорят, обусловлена или нет популярность этих технологий тем, что они представлены программами с открытым исходным кодом, но PHP, MySQL и Apache *действительно* являются наиболее востребованными инструментами в своих категориях. Вполне определенно можно сказать, что их принадлежность к продуктам с открытым кодом означает, что они были разработаны в сообществе команд программистов, которые придавали им свойства в соответствии со своими желаниями и потребностями и хранили исходный код доступным для всеобщего просмотра и изменения. Ошибки и бреши в системе безопасности могут быть быстро распознаны и предотвращены еще до их проявления.

Есть и еще одно преимущество: все эти программы могут использоваться бесплатно. Если вы наращиваете пропускную способность своего сайта и привлекаете к его обслуживанию различные серверы, не нужно задумываться о приобретении дополнительных лицензий. Не нужно также пересматривать свой бюджет перед тем, как принять решение об обновлении системы и установке самых последних версий этих продуктов.

А теперь все это, вместе взятое

Истинная красота PHP, MySQL, JavaScript (иногда при содействии jQuery или других сред), CSS и HTML5 проявляется в том замечательном способе, благодаря которому они совместно работают над производством динамического веб-контента: PHP занят основной работой на веб-сервере, MySQL управляет данными, а CSS

и JavaScript вместе заботятся о представлении веб-страницы. JavaScript может также взаимодействовать с вашим PHP-кодом на веб-сервере, когда ему нужно что-нибудь обновить (как на сервере, так и на веб-странице). И с новыми, высокоэффективными свойствами HTML5, такими как холсты, аудио, видео и геолокация, можно придать вашим веб-страницам более высокую динамичность, интерактивность и мультимедийность.

Неплохо бы теперь подвести краткий итог всему, что изложено в данной главе, и, не пользуясь программным кодом, рассмотреть процесс, сочетающий в себе некоторые из этих технологий в повседневно использующейся многими сайтами функции асинхронного обмена данными: проверке в процессе регистрации новой учетной записи, не занято ли выбранное имя другим посетителем сайта. Хорошим примером подобного использования технологий может послужить почтовый сервер Gmail (рис. 1.3).

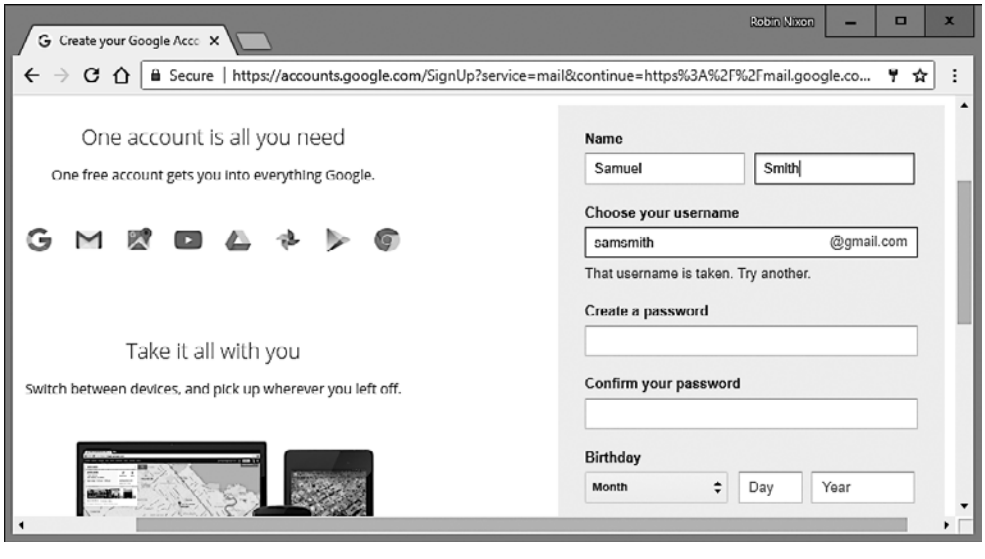

Рис. 1.3. Gmail применяет технологию асинхронного обмена данными для проверки допустимости пользовательских имен

Этот асинхронный процесс состоит из следующих шагов.

1. Сервер выдает код HTML для создания веб-формы, запрашивающей необходимые данные: имя пользователя, настоящее имя, настоящую фамилию и адрес электронной почты.
2. Одновременно с этим сервер вкладывает в HTML JavaScript-код, позволяющий отслеживать содержимое поля ввода имени пользователя и проверять два обстоятельства: введен ли в это поле какой-нибудь текст и был ли фокус ввода перемещен из этого поля по щелчку пользователя на другом поле ввода.

3. Как только будет введен текст и фокус ввода перемещен на другой элемент формы, код JavaScript в фоновом режиме передает введенное имя пользователя PHP-сценарию на веб-сервере и ждет ответной реакции.
4. Веб-сервер ищет имя пользователя и возвращает коду JavaScript ответ, в котором сообщает, было ли уже задействовано такое же имя.
5. Затем JavaScript размещает под полем ввода имени пользователя индикатор приемлемости имени пользователя, возможно, в виде зеленой галочки или красного крестика, сопровождая его текстом.
6. Если пользователь ввел неприемлемое имя, но все же пытается отправить форму, код JavaScript прерывает отправку и повторно обращает внимание пользователя (возможно, выводя более крупный графический индикатор и/или открывая окно предупреждения) на необходимость выбора другого имени.
7. Усовершенствованная версия этого процесса может даже изучить имя, запрошенное пользователем, и предложить альтернативное, доступное на данный момент.

Все это для удобства пользователя и целостности восприятия им всего происходящего делается без привлечения его внимания в фоновом режиме. Без использования асинхронного обмена данными на сервер будет отправлена вся форма, затем он вернет код HTML с подсветкой тех полей, в которых были допущены ошибки. Можно, конечно, сделать и так, но обработка поля на лету будет выглядеть намного интереснее и приятнее.

Асинхронный обмен данными может использоваться для решения куда более широкого круга задач, чем простой контроль и обработка вводимой информации. Далее в этой книге будет рассмотрено много дополнительных приемов, реализуемых с применением AJAX.

В этой главе вашему вниманию было представлено довольно полное введение в основные технологии применения PHP, MySQL, JavaScript, CSS и HTML5 (а также Apache) и рассмотрен порядок их совместной работы. В главе 2 будут описаны способы установки вашего собственного сервера, предназначенного для веб-разработок, на котором можно будет освоить на практике весь изучаемый материал.

Вопросы

1. Какие четыре компонента необходимы для создания полностью динамических сайтов?
2. Что означает аббревиатура HTML?
3. Почему в названии MySQL присутствуют буквы SQL?

4. И PHP, и JavaScript являются языками программирования, генерирующими динамическое содержимое веб-страниц. В чем состоит их главное различие и почему вы будете использовать оба этих языка?
5. Что означает аббревиатура CSS?
6. Перечислите три важнейших новых элемента, появившихся в HTML5.
7. Если вам удастся обнаружить ошибку в одном из инструментальных средств с открытым кодом (что случается довольно редко), то как, по-вашему, можно получить исправленную версию?
8. Почему такая среда, как jQuery Mobile, играет столь важную роль в разработке современных веб-сайтов и веб-приложений?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

2 Установка сервера, предназначенного для разработки

Если у вас есть желание разрабатывать интернет-приложения, но нет собственного сервера для их разработки, то прежде, чем протестировать каждую созданную модификацию приложения, вам придется загружать ее на сервер, находящийся где-нибудь в Интернете.

Даже при наличии высокоскоростного широкополосного подключения это обстоятельство может существенно замедлить разработку. А на локальном компьютере тестирование может быть не сложнее обновления программы (зачастую запускается простым щелчком на значке) с последующим нажатием кнопки Refresh (Обновить) в браузере.

Еще одно преимущество разработочного сервера заключается в том, что при написании и тестировании программ не нужно волноваться о смущающих разработчика ошибках или проблемах безопасности, однако при размещении приложения на публичном сайте следует знать о том, что люди могут увидеть, или о том, что они могут сделать с вашим приложением. Лучше решить все проблемы, пока вы работаете дома или в небольшом офисе, который, вероятнее всего, защищен межсетевыми экранами (брандмауэрами) и другими средствами обеспечения безопасности.

Получив в свое распоряжение разработочный сервер, вы удивитесь, как раньше могли обходиться без него, а также обрадуетесь легкости его настройки. Нужно лишь пройти все шаги, изложенные в следующих разделах, и выполнить соответствующие указания для обычных персональных компьютеров, Mac- или Linux-систем.

В этой главе будет рассмотрена только серверная сторона сетевого взаимодействия, о которой шла речь в главе 1. Но для тестирования результатов вашей работы, особенно потом, когда мы приступим к использованию JavaScript, CSS и HTML5, понадобится также копия каждого основного браузера, работающего под управлением удобной для вас системы. В списке браузеров должны быть по крайней мере Microsoft Edge, Mozilla Firefox, Opera, Safari и Google Chrome. Если вы хотите

убедиться, что ваши приложения также хорошо выглядят на мобильных устройствах, постарайтесь использовать для тестирования широкий диапазон устройств, работающих под управлением iOS и Android.

Что такое WAMP, MAMP и LAMP

WAMP, MAMP и LAMP — это сокращения от «Windows, Apache, MySQL и PHP», «Mac, Apache, MySQL и PHP» и «Linux, Apache, MySQL и PHP» соответственно. Такими сокращениями описываются полноценные функциональные установки, используемые для разработки динамических веб-страниц.

Системы WAMP, MAMP и LAMP поставляются в форме пакетов, связывающих упакованные программы таким образом, чтобы их не нужно было устанавливать и настраивать по отдельности. Это означает, что нужно просто загрузить и установить одну программу и следовать простым подсказкам, чтобы подготовить разработочный сервер и запустить его в кратчайшие сроки и с минимальными усилиями.

В процессе установки будут заданы исходные настройки. Конфигурация безопасности при такой установке не будет столь же строгой, как на технологическом веб-сервере, поскольку она оптимизирована для использования на локальной машине. Поэтому не следует пользоваться такими настройками при установке технологического сервера.

Но для разработки и тестирования сайтов и приложений подобная установка подойдет как нельзя лучше.

Если для создания своей системы разработки вы решили не использовать WAMP/MAMP/LAMP, следует учесть, что загрузка и самостоятельная взаимоувязка составных частей займут очень много времени и могут отнять большое количество сил на исследования для создания полноценной конфигурации всей системы. Но если все компоненты у вас уже установлены и согласованы друг с другом, они смогут работать с примерами из этой книги.

Установка AMPPS в системе Windows

Существует несколько доступных WAMP-серверов, каждый из которых предлагает свою немного отличающуюся от других конфигурацию. Среди различных бесплатных вариантов с открытым кодом самым лучшим будет AMPPS. Его можно загрузить с главной страницы сайта http://ampps.com (рис. 2.1).

Я рекомендую вам всегда загружать последний стабильный выпуск (на момент написания этих строк им был выпуск с номером 3.8 объемом около 128 Мбайт). На странице загрузки перечислены различные установщики для Windows, OS X и Linux.

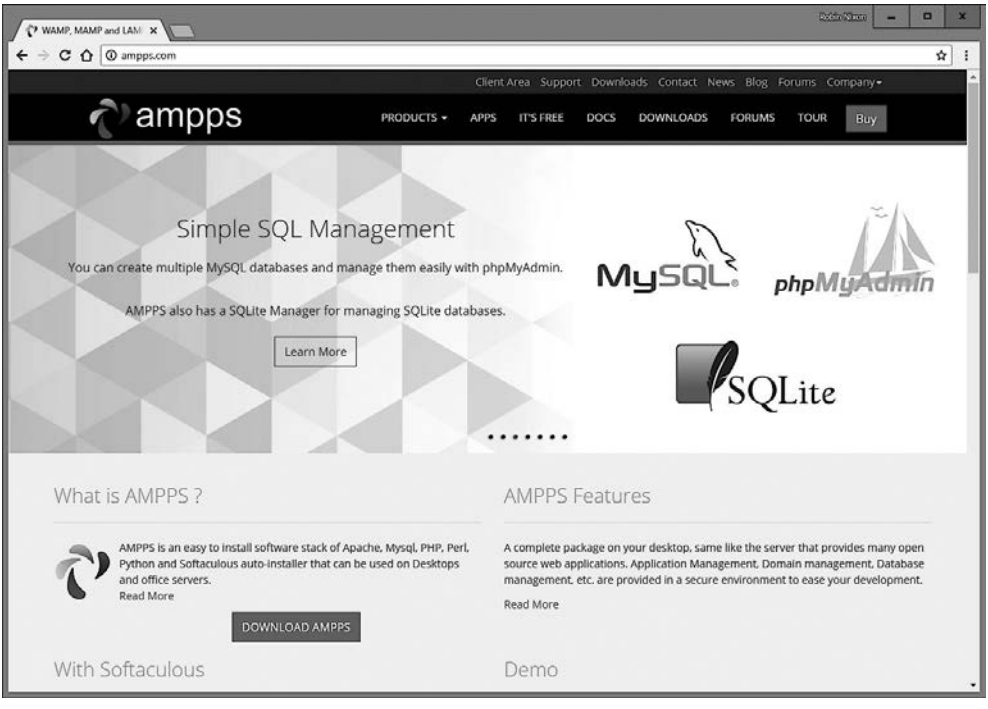

Рис. 2.1. Веб-сайт AMPPS

Когда это издание выйдет в свет, некоторые виды экранов и настройки, рассмотренные далее, наверняка изменятся. Если это произойдет, руководствуйтесь здравым смыслом и постарайтесь выполнить все действия как можно ближе к описываемой последовательности.

Запустите установщик после его загрузки, чтобы появилось окно, показанное на рис. 2.2. Если вы используете антивирусную программу или в Windows активизировано управление учетными записями (Account Control), то прежде чем вы перейдете к этому окну, вам может быть показано одно или несколько предупреждений, и для продолжения установки придется нажать кнопки Yes (Да) и (или) OK.

Нажмите кнопку Next (Далее), после чего нужно будет принять соглашения. Нажмите кнопку Next (Далее) еще раз, а затем еще раз, чтобы пройти через информационный экран. После этого понадобится подтвердить место установки. Вероятнее всего, вам будет предложено что-нибудь вроде следующего местоположения в зависимости от буквы вашего основного жесткого диска, но при желании в этот путь можно внести изменения:

```
C:\Program Files (x86)\Ampps
```

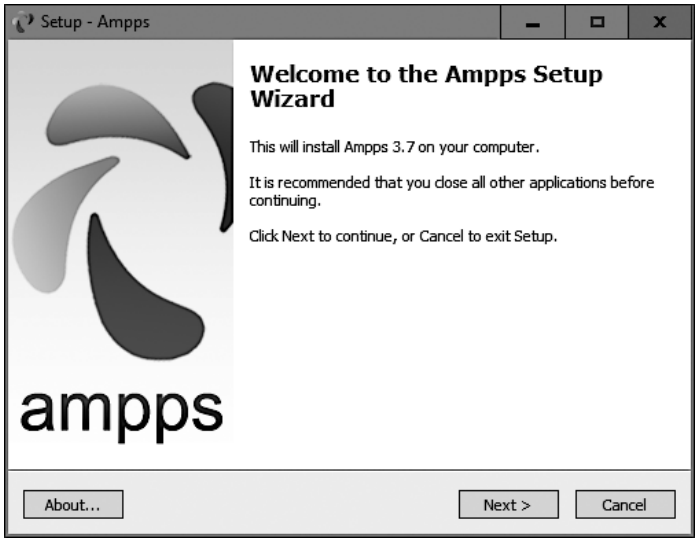

Рис. 2.2. Открытое окно установщика

Приняв решение о месте установки AMPPS, нажмите кнопку Next (Далее), выберите имя папки начального меню и снова нажмите кнопку Next (Далее). На рис. 2.3 показано, что вам предоставится возможность выбора устанавливаемых значков. Чтобы продолжить процесс установки, на следующем экране нажмите кнопку Install (Установить).

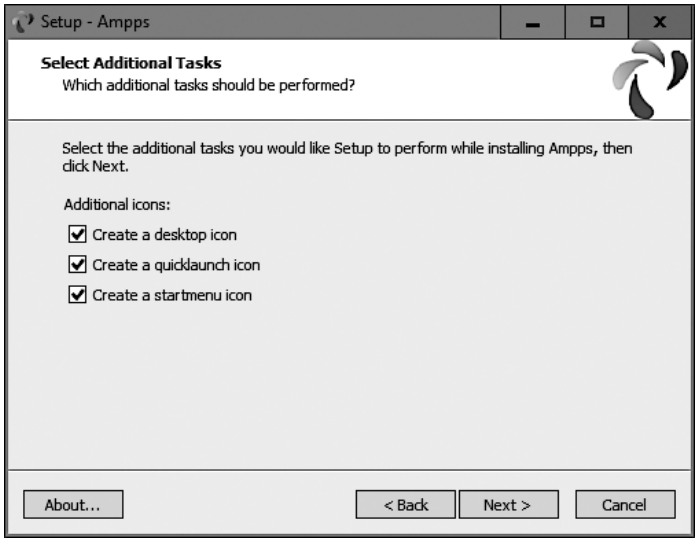

Рис. 2.3. Выбор устанавливаемых значков

Установка займет несколько минут, после чего появится окно завершения процесса, показанное на рис. 2.4, и появится возможность нажать кнопку Finish (Завершить).

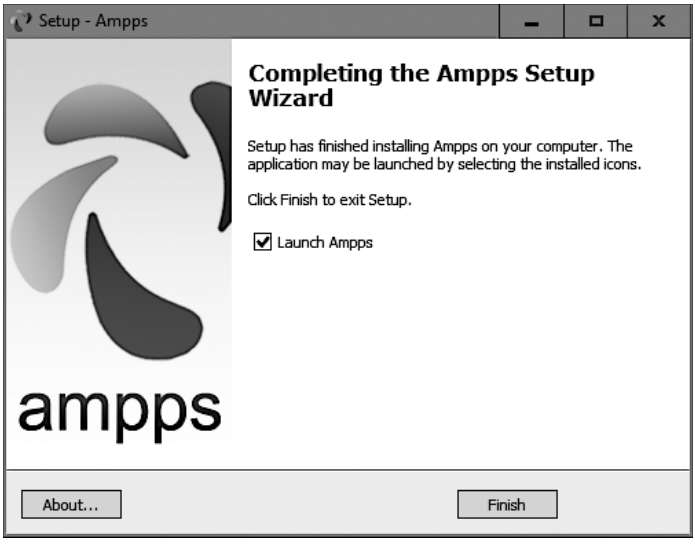

Рис. 2.4. Среда AMPPS установлена

В завершение нужно установить среду C++ Redistributable Visual Studio, если она не была установлена ранее. В этой среде будет вестись разработка программных средств. В качестве приглашения на установку появится окно, показанное на рис. 2.5. Нажмите кнопку Yes (Да) для запуска установки или кнопку No (Нет), если уверены в том, что эта среда уже установлена.

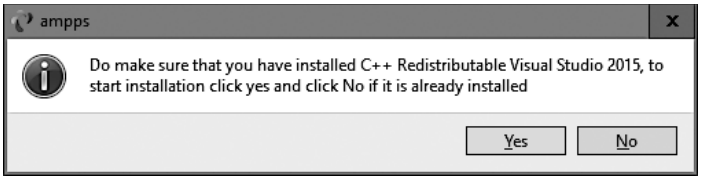

Рис. 2.5. Установите среду C++ Redistributable Visual Studio, если она еще не была установлена

Если выбрать установку среды, придется согласиться с условиями, предлагаемыми в появившемся окне, после чего нажать кнопку Install (Установить). Установка не займет много времени. Для ее завершения нужно будет нажать кнопку Close (Закрыть).

После установки среды AMPPS в правом нижнем углу экрана должно появиться окно управления, показанное на рис. 2.6. Это окно можно также вызвать, восполь-

зовавшись ярлыками приложения AMPPS в меню Пуск или на Рабочем столе, если ранее давалось разрешение на их создание.

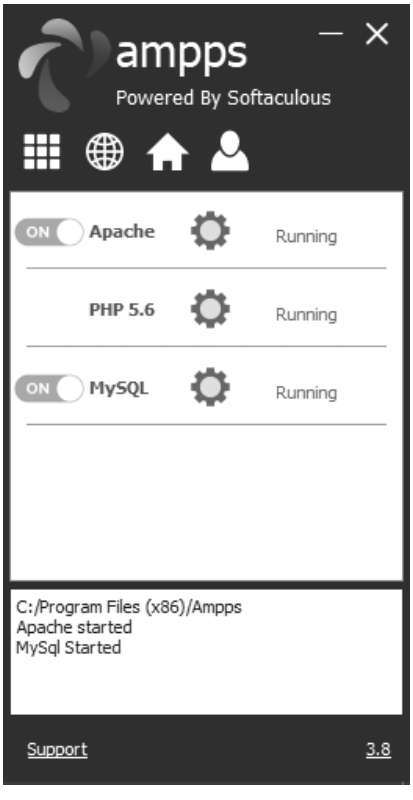

Рис. 2.6. Окно управления средой AMPPS

Прежде чем продолжить работу, рекомендуется ознакомиться с документацией AMPPS (http://ampps.com/wiki). Если после ознакомления все еще останутся вопросы, воспользуйтесь ссылкой Support (Поддержка), находящейся в нижней части окна управления. Щелкнув на ней, вы перейдете на сайт AMPPS, где можно будет открыть тематическую подсказку.

Нетрудно заметить, что в качестве исходной в AMPPS используется версия PHP с номером 5.6. В последующих разделах данной книги рассматриваются некоторые подробности наиболее важных изменений, внесенных в версию PHP 7. Если захочется испытать эти изменения на деле, нажмите в окне управления AMPPS кнопку настроек (в виде девяти белых квадратиков, образующих большой квадрат), после чего выберите пункт изменения версии, Change PHP Version, вызвав новое меню с возможностью выбора версии в диапазоне от 5.6 и до 7.1.

Тестирование установки

На данном этапе нужно проверить, что все работает должным образом. Для этого потребуется ввести любой из следующих URL-адресов в адресную строку браузера:

```
localhost
127.0.0.1
```

В результате будет вызван начальный экран, позволяющий провести настройку безопасности работы в среде AMPPS добавлением пароля (рис. 2.7). Рекомендую не устанавливать флажок безопасности и просто нажать кнопку Submit (Отправить), чтобы продолжить работу без установки пароля.

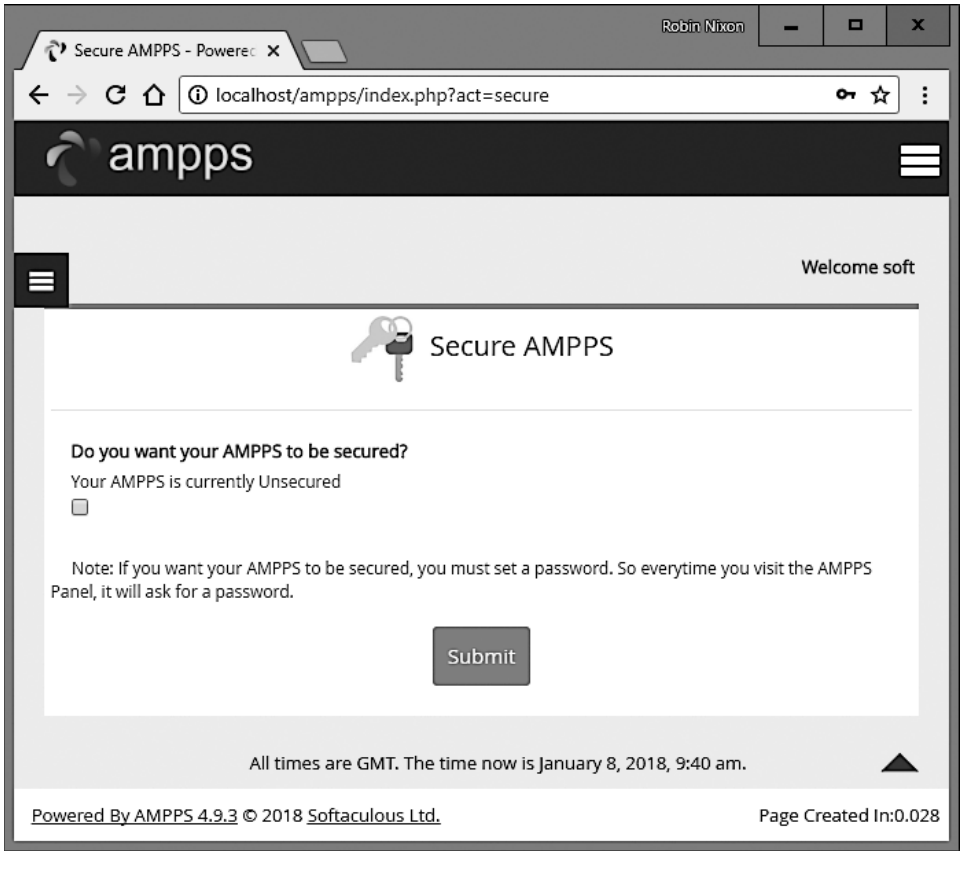

Рис. 2.7. Начальный экран настройки безопасности

Совершив это действие, вы попадете на главную страницу управления localhost/ampps/ (с этого момента предполагается, что обращение к AMPPS происходило через адрес localhost, а не через 127.0.0.1). На ней можно настраивать все аспекты

компонентов AMPPS и управлять ими, что следует отметить для будущих ссылок или, может быть, установить закладку в вашем браузере.

Далее, чтобы увидеть корневой каталог документа вашего нового веб-сервера Apache (рассматриваемый в следующем разделе), наберите такой адрес:

```
localhost
```

На этот раз вместо перехода к начальному экрану настройки безопасности на экран будет выведена страница, похожая на ту, что показана на рис. 2.8.

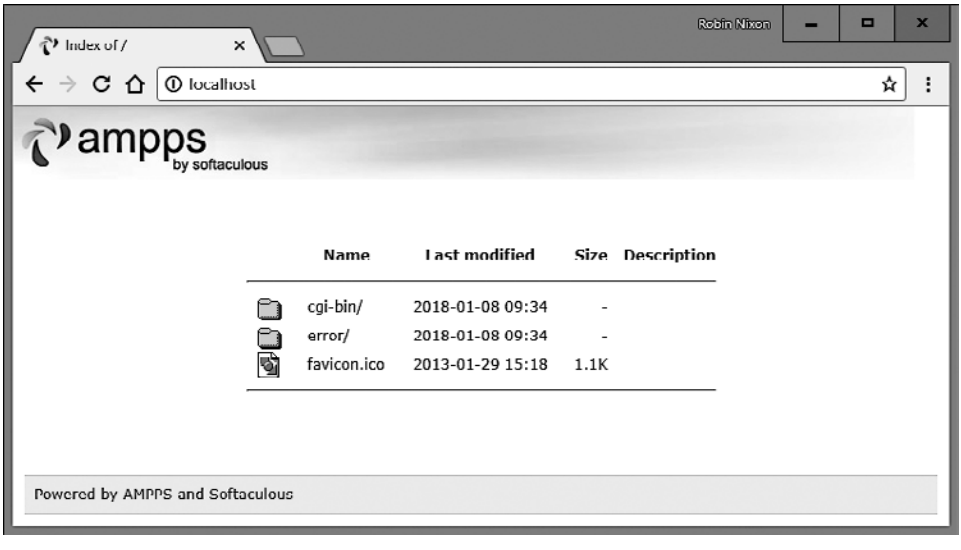

Рис. 2.8. Просмотр корневого каталога документа

Обращение к исходному источнику документов (Windows)

Исходным источником документов является каталог, в котором содержатся главные веб-документы домена. Именно он используется сервером, когда в браузере набирается базовый URL без пути, например http://yahoo.com или — на локальном сервере — http://localhost.

По умолчанию XAMPP использует для этого каталога следующее место: C:/xampp/htdocs.

Теперь, чтобы убедиться в том, что все сконфигурировано должным образом, напишем тестовую программу Hello World. Создайте небольшой HTML-файл со следующими строками, воспользовавшись Блокнотом или любым другим приложением либо текстовым редактором (при использовании текстовых процессоров,

таких как Microsoft Word, которые создают форматированный текст, файл нужно сохранять в виде простого текста):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>A quick test</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

После набора сохраните файл под именем `test.htm` в корневом каталоге документов. Если вы создаете файл в Блокноте, убедитесь, что в окне Сохранить как в списке Тип файла выбран вариант Все файлы, а не Текстовые документы (*.txt).

Теперь эту страницу можно вызвать в браузере, введя в адресной строке следующий URL-адрес (рис. 2.9):

```
http://localhost/test.html
```

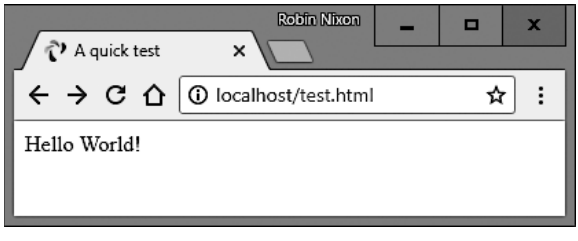

Рис. 2.9. Ваша первая веб-страница

Следует понимать, что переход на веб-страницу из корневого каталога документа (или из находящейся в нем папки) отличается от загрузки этой же страницы из файловой системы вашего компьютера. В первом случае будет гарантирован доступ к PHP, MySQL и ко всем свойствам веб-сервера, тогда как во втором файл будет просто загружен в браузер, который попытается создать его экранное представление, но не будет иметь возможности обработки кода PHP или других серверных инструкций. Поэтому примеры должны неизменно запускаться из адресной строки браузера с указанием перед их именами пути localhost, если только вы не уверены в том, что файл не зависит от использования функциональных возможностей веб-сервера.

Другие системы WAMP

При обновлении программы иногда работают неожиданным образом и в них даже могут проявиться какие-нибудь ошибки. Поэтому, столкнувшись с непреодолимыми в AMPPS трудностями, вы можете остановить свой выбор на одном из многих других решений, доступных в Интернете.

Можно будет точно так же воспользоваться всеми примерами, приводимыми в данной книге, но при этом придется следовать инструкциям, которые предоставляются с каждым WAMP-сервером и могут оказаться сложнее ранее упомянутого руководства.

Вот наиболее подходящие, на мой взгляд, серверы:

- EasyPHP (http://easyphp.org/);
- XAMPP (http://apachefriends.org/);
- WAMPServer (http://wampserver.com/en);
- Glossword WAMP (http://glossword.biz/glosswordwamp).

К вопросу об обновлениях AMPPS

Вполне вероятно, что пока данное издание будет самым последним, разработчики AMPPS могут внести в свой продукт ряд усовершенствований, в силу чего экраны и методы установки со временем могут претерпеть изменения, а кроме того, изменениям могут подвергнуться и версии Apache, PHP или MySQL. Поэтому не подумайте, что что-то не в порядке с выводимой на экран информацией и операции выглядят как-то по-другому. Разработчики AMPPS прилагают все усилия для того, чтобы их продукт оставался легок в использовании, поэтому следуйте любым выводимым на экран приглашениям и обращайтесь к документации, размещаемой на сайте http://ampps.com/.

Установка AMPPS в системе macOS

Среда AMPPS может использоваться и в macOS, а загрузить ее можно со страницы http://apachefriends.org, которая была показана на рис. 2.1 (как уже говорилось, текущая версия имеет номер 3.8).

Если ваш браузер после загрузки не откроет среду в автоматическом режиме, дважды щелкните на файле с расширением `.dmg`, после чего перетащите папку AMPPS на свою папку Applications (Приложения) (рис. 2.10).

Теперь откройте свою папку Applications (Приложения) и дважды щелкните на программе AMPPS. Если ваши настройки безопасности воспрепятствуют открытию файла, нажмите и удерживайте клавишу Control, однократно щелкнув на значке. Будет выведено новое окно с запросом подтверждения на открытие файла программы. Чтобы открыть файл, нажмите кнопку Open (Открыть). Возможно, после запуска приложения для продолжения работы придется ввести ваш пароль на macOS.

После того как среда AMPPS войдет в рабочий режим, в левом нижнем углу вашего рабочего стола появится окно управления, похожее на то, что изображено на рис. 2.6.

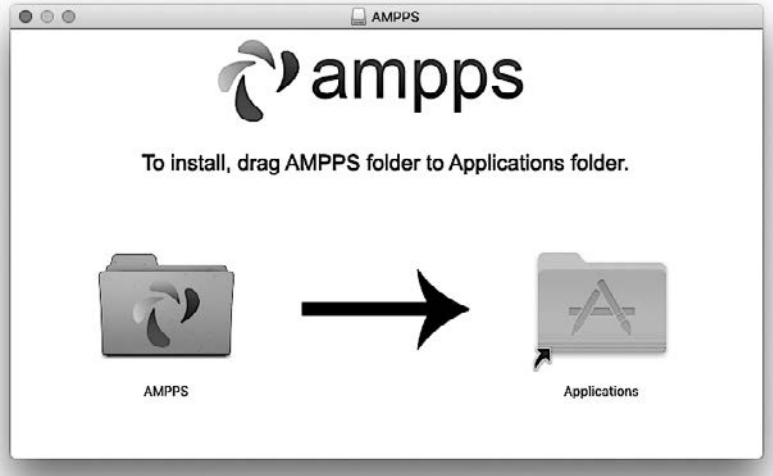

Рис. 2.10. Перетащите папку AMPPS на папку Applications (Приложения)

Нетрудно заметить, что в качестве исходной в AMPPS используется версия PHP с номером 5.6. В последующих разделах данной книги рассматриваются некоторые подробности наиболее важных изменений, внесенных в версию PHP 7. Если захочется испытать эти изменения на деле, нажмите в окне управления AMPPS кнопку настроек (в виде девяти белых квадратиков, образующих большой квадрат), после чего выберите пункт изменения версии, Change PHP Version, вызвав новое меню с возможностью выбора версии в диапазоне от 5.6 и до 7.1.

Обращение к исходному источнику документов (macOS)

По умолчанию корневой каталог документов AMPPS будет размещаться по следующему адресу:

```
/Applications/Ampps/www
```

Теперь, чтобы убедиться в правильности всех настроек, следует создать общепринятый файл `Hello World`. Поэтому воспользуйтесь программой простого текстового редактора, но не солидным текстовым процессором вроде Microsoft Word (если только не станете сохранять проделанную в нем работу в качестве простого текста), и создайте небольшой HTML-файл, состоящий из следующих строк:

```
<html>
  <head>
    <title>A quick test</title>
  </head>
  <body>
```

```
    Hello World!
  </body>
</html>
```

После набора данного текста сохраните файл в корневом каталоге документа, назвав его `test.html`.

Теперь можно вызвать созданную вами веб-страницу в браузере путем ввода в его адресной строке следующего URL (чтобы увидеть тот же результат, который был показан на рис. 2.9):

```
localhost/test.html
```

Следует понимать, что переход на веб-страницу из корневого каталога документа (или из находящейся в нем папки) отличается от загрузки этой же страницы из файловой системы вашего компьютера. В первом случае будет гарантирован доступ к PHP, MySQL и всем свойствам веб-сервера, тогда как во втором файл будет просто загружен в браузер, который попытается создать его экранное представление, но не будет иметь возможности обработки кода PHP или других серверных инструкций. Поэтому примеры должны неизменно запускаться из адресной строки браузера с указанием перед их именами пути localhost, если только вы не уверены в том, что файл не зависит от использования функциональных возможностей веб-сервера.

Установка LAMP в Linux

Эта книга ориентирована в основном на пользователей PC и MAC, но содержащийся в ней код будет хорошо работать и на компьютере с Linux. Существуют десятки популярных разновидностей Linux, на каждую из которых LAMP может устанавливаться со своими особенностями. Но многие версии Linux поступают с предустановленным веб-сервером и MySQL, поэтому есть вероятность, что у вас уже все готово к работе. Чтобы понять, так ли это, попробуйте ввести в браузер следующий адрес и посмотрите, получите ли вы веб-страницу, используемую по умолчанию в исходном источнике документов:

```
localhost
```

Если все заработает, то у вас, наверное, установлен сервер Apache, а также может быть установлена и запущена база данных MySQL, но, чтобы окончательно удостовериться в этом, проверьте факт их установки вместе со своим системным администратором.

Если веб-сервер не установлен, можете воспользоваться версией AMPPS, которую можно загрузить со страницы http://apachefriends.org.

Последовательность установки похожа на ту, которая рассматривалась в предыдущем разделе. Если нужна дополнительная подсказка по использованию программного продукта, обратитесь к документации, размещенной по адресу http://ampps.com/wiki.

Работа в удаленном режиме

Если есть доступ к веб-серверу, на котором уже имеются сконфигурированные PHP и MySQL, то вы всегда можете применить его для веб-разработок. Но это не самый лучший вариант, если только вы не пользуетесь высокоскоростным подключением. Разработка, выполняемая на локальной машине, позволяет протестировать новые модификации практически без задержки или с небольшой задержкой на загрузку.

Может вызвать трудности и удаленный доступ к MySQL. Чтобы зарегистрироваться на сервере для создания баз данных вручную и установки прав доступа из командной строки, нужно воспользоваться безопасным SSH-протоколом. Компания, предоставляющая веб-хостинг, посоветует, как это можно сделать наилучшим образом, и предоставит пароль для доступа к MySQL (а в первую очередь, разумеется, для доступа к самому серверу). Если выбирать не из чего, я все же не советую пользоваться для удаленного входа на любой сервер небезопасным протоколом Telnet.

Вход в систему

Я рекомендую пользователям Windows для доступа к Telnet и SSH (следует помнить, что уровень безопасности у SSH значительно выше, чем у Telnet) как минимум установить программу PuTTY.

На компьютере Mac система SSH будет доступна изначально. Нужно только выбрать папку Applications, перейти к папке Utilities, а затем запустить программу Terminal. В окне терминала необходимо войти на сервер, используя SSH в следующей команде:

```
ssh mylogin@server.com
```

где *server.com* — это имя сервера, на который необходимо войти, а *mylogin* — имя пользователя, под которым нужно войти в систему. Затем система запросит у вас пароль для данного имени пользователя, и, если вы введете правильный пароль, вход состоится.

Использование FTP

Для переноса файлов на веб-сервер и обратно понадобится FTP-программа. Если искать подходящую в Интернете, можно найти так много программ, что на выбор нужной именно вам уйдет уйма времени.

Я предпочитаю пользоваться FTP-программой FileZilla (http://filezilla-project.org/), имеющей открытый код в версиях для Windows, Linux и macOS 10.5 и выше (рис. 2.11). Подробные инструкции по работе с FileZilla доступны по адресу http://wiki.filezilla-project. Разумеется, если у вас уже есть FTP-программа, вы можете использовать ее.

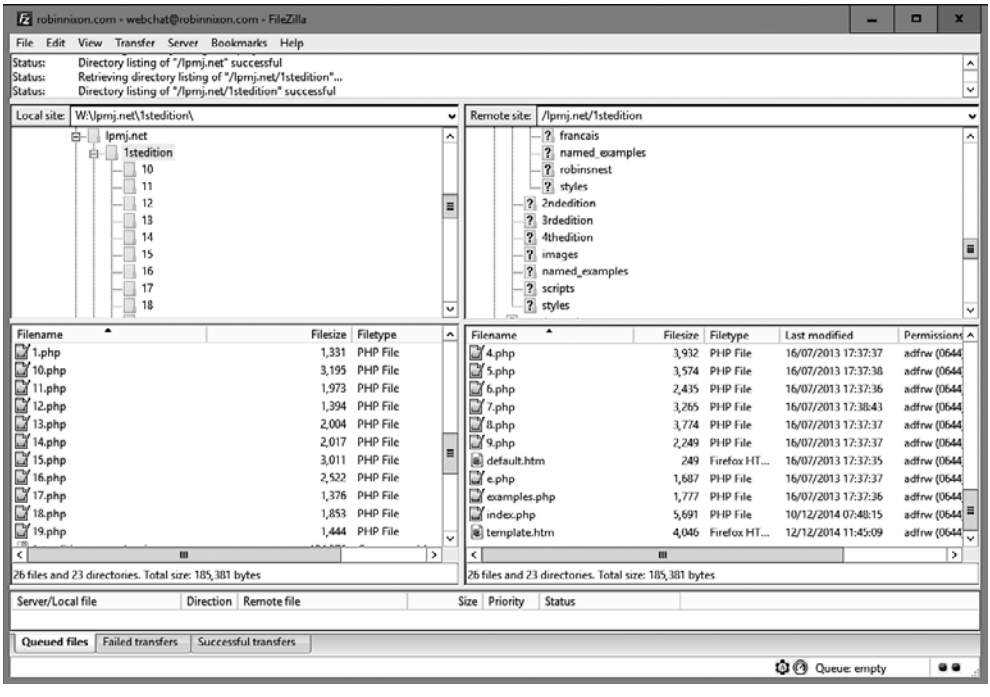

Рис. 2.11. FileZilla является полнофункциональной FTP-программой

Использование редактора программ

Хотя для редактирования HTML, PHP и JavaScript подходит любой текстовый редактор, существуют очень удобные приложения, специально предназначенные для редактирования текста программ. В них имеются весьма полезные возможности, например цветная подсветка синтаксиса. Современные редакторы программ хорошо продуманы и могут еще до запуска программы на выполнение показывать места, в которых допущены синтаксические ошибки. Перейдя к использованию современного редактора, вы будете удивлены тому, что раньше обходились без него.

Есть множество хороших и доступных программ, но я остановил свой выбор на программе Editra (рис. 2.12), поскольку она распространяется бесплатно и доступна для macOS, Windows и Linux/UNIX. Программы можно загрузить, перейдя на сайт http://editra.org и выбрав ссылку Download (Скачать) в верхней части страницы, где также можно найти и ссылку на документацию, сопровождающую редактор. Конечно, у каждого свой стиль и предпочтения в программировании, поэтому, если данный редактор вас не устраивает, можно выбрать что-либо из множества других доступных программ-редакторов или же пожелать перейти непосредственно в интегрированную среду разработки (IDE) в соответствии с описанием, приведенным в следующем разделе.

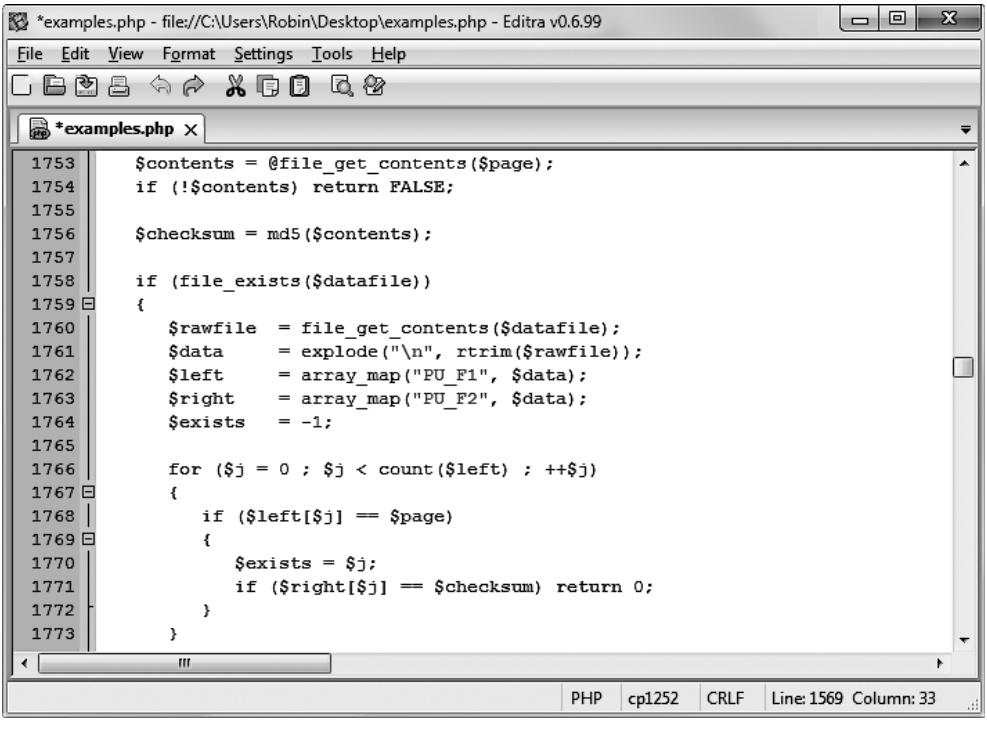

Рис. 2.12. Программы-редакторы (подобные показанной здесь Editra) превосходят по своим возможностям обычные текстовые редакторы

Editra, как показано на рис. 2.12, выделяет синтаксис, используя соответствующие цвета, что очень удобно. Можно поместить курсор после квадратной или фигурной скобки, и Editra подсветит соответствующую парную скобку, давая возможность определить лишние или недостающие скобки. В Editra предлагается также множество других свойств, облегчающих работу с текстом программы.

Опять же, если вы предпочитаете какую-нибудь другую программу-редактор, воспользуйтесь ею, поскольку всегда лучше работать с теми программами, с которыми вы уже знакомы.

Использование IDE

При всех достоинствах специализированных редакторов программ, позволяющих повысить производительность труда программиста, они не могут сравниться с интегрированными средами разработки, предлагающими множество дополнительных возможностей, например проведение отладки и тестирования программ прямо в редакторе, а также отображение описаний функций и многое другое.

На рис. 2.13 показана популярная интегрированная среда разработки phpDesigner с программой PHP, которая загружена в главное окно, и с расположенным в правой части анализатором кода Code Explorer, в котором перечислены различные классы, функции и переменные, используемые в этой программе.

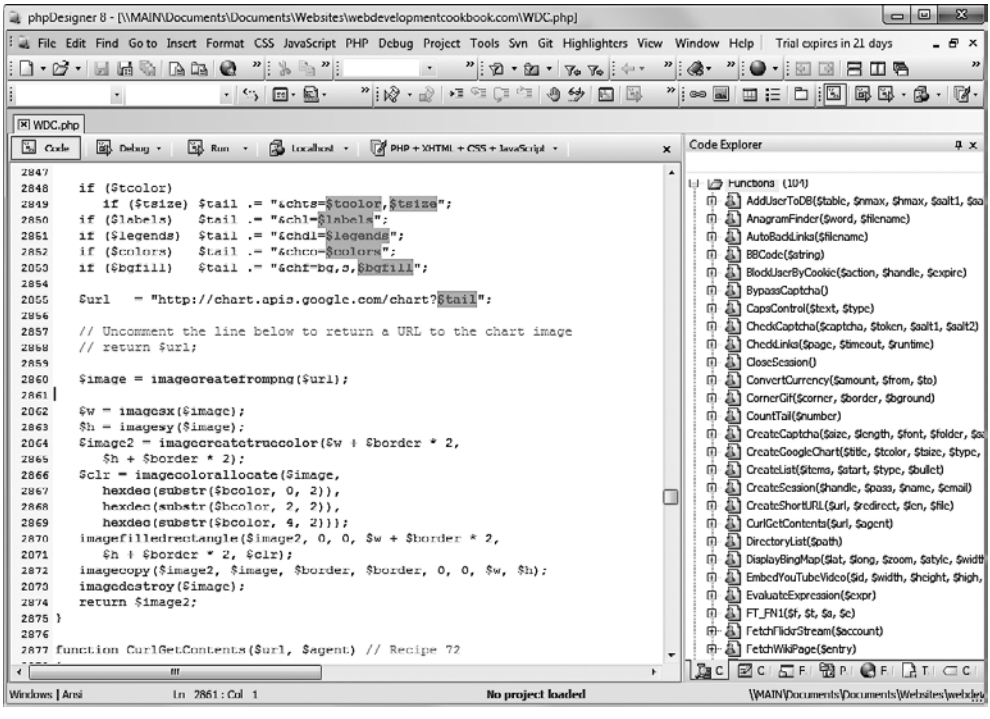

Рис. 2.13. При использовании такой IDE, как phpDesigner, разработка PHP-программы идет намного быстрее и проще

При разработке продукта в IDE можно установить контрольные точки, а затем запустить весь код целиком (или по частям), тогда его выполнение будет останавливаться в контрольных точках и вам будет предоставляться информация о текущем состоянии программы.

Помочь изучению процесса создания программ может то, что приводимые в данной книге примеры могут быть введены в IDE и запущены в этой среде, — тогда вызывать браузер уже не понадобится.

Существует несколько интегрированных сред разработки, доступных для различных платформ. Большинство из них являются коммерческими продуктами, но встречаются и бесплатные версии. В табл. 2.1 приведены некоторые наиболее популярные интегрированные среды разработки PHP-программ и URL-адреса, с которых их можно загрузить.

Таблица 2.1. Интегрированные среды разработки для PHP

IDE	URL-адрес загрузки	Цена	Win	Mac	Linux
Eclipse PDT	http://eclipse.org/pdt/downloads/	Бесплатно	✓	✓	✓
Komodo IDE	http://activestate.com/Products/komodo_ide	$295	✓	✓	✓
NetBeans	http://www.netbeans.org	Бесплатно	✓	✓	✓
phpDesigner	http://mpsoftware.dk	$39	✓	–	–
PHPeclipse	http://phpeclipse.de	Бесплатно	✓	✓	✓
PhpED	http://nusphere.com	$99	✓	–	✓
PHPEdit	http://phpedit.com	$117	✓	–	–

Потратьте время на установку удобного, с вашей точки зрения, редактора программ или IDE, и тогда вы будете готовы опробовать в работе примеры, приводимые в следующих главах.

Теперь, вооружившись редактором программ или IDE, вы готовы к переходу к главе 3, в которой начнется углубленное изучение PHP и совместной работы HTML и PHP, а также структуры самого языка PHP. Но перед тем, как перейти к этой главе, я предлагаю проверить полученные вами знания и ответить на следующие вопросы.

Вопросы

1. В чем разница между WAMP, MAMP и LAMP?
2. Что общего у IP-адреса 127.0.0.1 и URL-адреса http://localhost?
3. Для чего предназначена FTP-программа?
4. Назовите основной недостаток работы на удаленном веб-сервере.
5. Почему лучше воспользоваться редактором программ, а не обычным текстовым редактором?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

3 Введение в PHP

В главе 1 о PHP говорилось как о языке, заставляющем сервер генерировать динамическую, потенциально разную выходную информацию при каждом запросе браузером веб-страницы. В данной главе начнется изучение этого простого, но мощного языка, которое продолжится в следующих главах и завершится в главе 7.

Я призываю вас выполнять разработку кода PHP в одной из интегрированных сред разработки (IDE), упомянутых в главе 2. Она поможет выявить опечатки, что существенно ускорит обучение по сравнению с работой в менее функциональных редакторах.

Многие из IDE позволяют запускать код, рассматриваемый в этой главе, и изучать производимую им выходную информацию. Мы также разберем методы вставки кода PHP в файл HTML, чтобы иметь представление о внешнем виде выходной информации на веб-странице (то есть о том виде, в котором она в итоге предстанет перед пользователями).

В процессе создания веб-страницы будут представлять собой комбинацию PHP, HTML, JavaScript, инструкций MySQL и форматирования с помощью CSS. Кроме того, каждая страница может привести на другие страницы, предоставляя пользователям возможность щелкать на ссылках и заполнять формы. Хотя при изучении этих языков можно обойтись и без таких сложностей. На данном этапе нужно сконцентрироваться исключительно на написании PHP-кода и на достижении предсказуемости содержимого выходной информации или по крайней мере на умении разбираться в характере этой информации.

Включение PHP в HTML

По умолчанию в конце имен PHP-документов ставится расширение PHP. Когда веб-сервер встречает в запрашиваемом файле это расширение, он автоматически передает файл PHP-процессору. Веб-серверы имеют довольно широкий диапазон настроек, и некоторые веб-разработчики выбирают такой режим работы, при

котором для разбора PHP-процессору принудительно передаются также файлы с расширениями HTM или HTML. Обычно это связано с тем, что разработчики хотят скрыть факт использования PHP.

Программа на PHP отвечает за возвращение файла в чистом виде, пригодном для отображения в браузере. В простейшем случае на выходе документа PHP будет получаться только код HTML. Чтобы убедиться в этом, можно взять любой HTML-документ и сохранить его в качестве PHP-документа (например, сохраняя файл `index.html` под именем `index.php`), и он будет отображаться точно так же, как исходный файл.

Для запуска команд PHP нужно изучить новый тег. Его открывающая часть имеет следующий вид:

```
<?php
```

Первое, что может броситься в глаза, — незавершенность тега. Это обусловлено тем, что внутри тега могут помещаться целые фрагменты кода PHP. Они заканчиваются, только когда встречается закрывающая часть тега такого вида:

```
?>
```

Небольшая PHP-программа Hello World может иметь вид, показанный в примере 3.1.

Пример 3.1. Вызов PHP

```
<?php
  echo "Hello world";
?>
```

Этот тег очень гибок в использовании. Некоторые программисты открывают его в начале документа, а закрывают в самом конце и выводят любой код HTML путем непосредственного использования команды PHP.

Другие программисты предпочитают помещать в эти теги как можно меньшие фрагменты кода PHP и именно в тех местах, где нужно воспользоваться динамическими сценариями, а весь остальной документ составлять из стандартного кода HTML.

Сторонники последнего метода программирования зачастую аргументируют свой выбор тем, что такой код выполняется быстрее, а сторонники первого метода утверждают, что увеличение скорости настолько мизерное, что оно не может оправдать дополнительные сложности многочисленных вставок PHP в отдельно взятый документ.

По мере изучения языка вы, несомненно, определитесь в своих стилевых предпочтениях при создании разработок на PHP, но для упрощения примеров, приводимых в этой книге, я свел количество переходов между PHP и HTML к минимуму, в среднем к одному-двум переходам на документ.

Кстати, существует и несколько иной вариант синтаксиса PHP. Если поискать примеры PHP-кода в Интернете, то можно встретить код, где используется следующий синтаксис открывающего и закрывающего тегов:

```
<?
  echo "Hello world";
?>
```

Несмотря на то что здесь неочевиден вызов PHP-парсера, это вполне приемлемый альтернативный синтаксис, который, как правило, также работает. Но я не советую его использовать, поскольку он несовместим с XML и в настоящее время его применение не приветствуется (это значит, что он больше не рекомендуется и его поддержка может быть удалена в будущих версиях).

Если в файле содержится только код PHP, то закрывающий тег ?> можно опустить. Именно так и нужно делать, чтобы гарантировать отсутствие в файлах PHP лишнего пустого пространства (что имеет особую важность при написании объектно-ориентированного кода).

Примеры в этой книге

Чтобы вы не тратили время на набор примеров, приводимых в этой книге, все они заархивированы на сайте по адресу http://lpmj.net, откуда можно загрузить файл архива, щелкнув на ссылке 5th Edition Examples (Примеры пятого издания), которая находится в верхней части страницы (рис. 3.1).

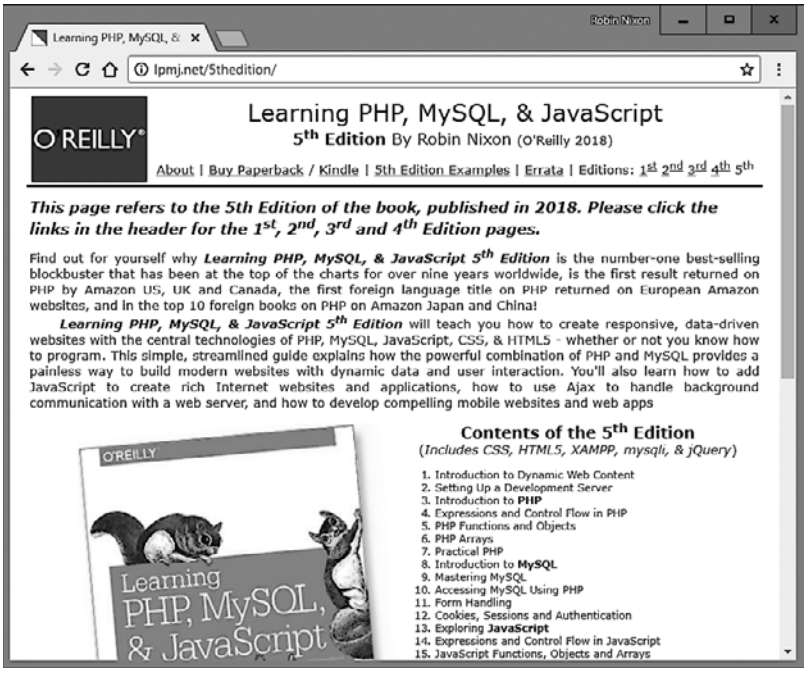

Рис. 3.1. Примеры, приводимые в этой книге, можно загрузить с адреса http://lpmj.net

Вдобавок к перечислению всех примеров по номеру главы и примера (в частности, `example3-1.php`) предоставляемый архив содержит дополнительный каталог `named_examples`, где можно найти все примеры, которые предлагалось сохранять в файлах с конкретными именами (как в показанном далее примере 3.4, который нужно будет сохранить в файле с именем `test1.php`).

Структура PHP

В этом разделе будет рассмотрено довольно много основных положений. Разобраться во всем этом несложно, но я рекомендую проработать материал как можно тщательнее, поскольку он служит основой для понимания всей остальной книги. Как всегда, в конце главы будут заданы вопросы, с помощью которых можно будет проверить, насколько глубоко усвоен материал.

Комментарии

Существует два способа добавления комментариев к коду PHP. Первый, предусматривающий размещение в начале строки двух прямых слешей, превращает в комментарий отдельную строку:

```
// Это комментарий
```

Он хорошо подходит для временного исключения из программы строки кода, являющейся источником ошибок. Например, такой способ комментирования можно применить для того, чтобы скрыть строку кода до тех пор, пока в ней не возникнет необходимость:

```
// echo "X equals $x";
```

Такой комментарий можно также вставить сразу же после строки кода, чтобы описать ее действие:

```
$x += 10; // Увеличение значения $x на 10
```

Когда понадобится комментарий, состоящий из нескольких строк, нужно воспользоваться вторым способом комментирования, который показан в примере 3.2.

Пример 3.2. Многострочный комментарий

```
<?php
/* Это область
   многострочного комментария,
   которая не будет
   подвергаться интерпретации */
?>
```

Для открытия и закрытия комментария можно воспользоваться парами символов `/*` и `*/` практически в любом произвольно выбранном месте кода. Если не все, то большинство программистов используют эту конструкцию для временного превра-

щения в комментарий целого неработоспособного раздела кода или такого раздела, который по тем или иным причинам нежелательно интерпретировать.

Типичная ошибка — применение пар символов /* и */ для того, чтобы закомментировать большой фрагмент кода, уже содержащий закомментированную область, в которой используются эти же пары символов. Комментарии не могут быть вложенными друг в друга, поскольку PHP-интерпретатор не поймет, где заканчивается комментарий, и выведет на экран сообщение об ошибке. Но если вы используете редактор программ или интегрированную среду разработки с подсветкой синтаксиса, то ошибку такого рода нетрудно будет заметить.

Основной синтаксис

PHP — очень простой язык, уходящий своими корнями в языки C и Perl, но все же больше похожий на Java. Он очень гибок, но существует несколько правил, относящихся к его синтаксису и структуре, которые следует изучить.

Точки с запятыми

В предыдущих примерах можно было заметить, что команды PHP завершаются точкой с запятой:

```
$x += 10;
```

Возможно, чаще всего причиной ошибок, с которыми приходится сталкиваться при работе с PHP, становится забывчивость. Если не поставить эту точку с запятой, PHP вынужден будет рассматривать в качестве одной сразу несколько инструкций, при этом он не сможет разобраться в ситуации и выдаст ошибку синтаксического разбора — `Parse error`.

Символ $

Символ `$` в разных языках программирования используется в различных целях. Например, в языке BASIC символ `$` применялся в качестве завершения имен переменных, чтобы показать, что они относятся к строкам.

А в PHP символ `$` должен ставиться перед именами всех переменных. Это нужно для того, чтобы PHP-парсер работал быстрее, сразу же понимая, что имеет дело с переменной. К какому бы типу ни относились переменные — к числам, строкам или массивам, все они должны выглядеть так, как показано в примере 3.3.

Пример 3.3. Три разновидности присваивания значений переменным

```
<?php
  $mycounter = 1;
  $mystring = "Hello";
  $myarray = array("One", "Two", "Three");
?>
```

Вот, собственно, и весь синтаксис, который нужно усвоить. В отличие от языков, в которых отношение к способам отступа текста программы и размещения кода очень строгое (например, от Python), PHP дает полную свободу использования (или игнорирования) любых отступов и любого количества пробелов по вашему усмотрению. В действительности же разумное использование того, что называется свободным пространством, обычно поощряется (наряду с всесторонним комментированием), поскольку помогает разобраться в собственном коде, когда к нему приходится возвращаться по прошествии некоторого времени. Это помогает и другим программистам, вынужденным поддерживать ваш код.

Переменные

Понять, что такое переменные PHP, поможет простая метафора. Думайте о них как о небольших (или больших) спичечных коробках! Именно как о спичечных коробках, которые вы раскрасили и на которых написали некие имена.

Строковые переменные

Представьте, что у вас есть коробок, на котором написано слово *username* (имя пользователя). Затем вы пишете на клочке бумаги *Fred Smith* и кладете эту бумажку в коробок (рис. 3.2). Этот процесс похож на присваивание переменной строкового значения:

```
$username = "Fred Smith";
```

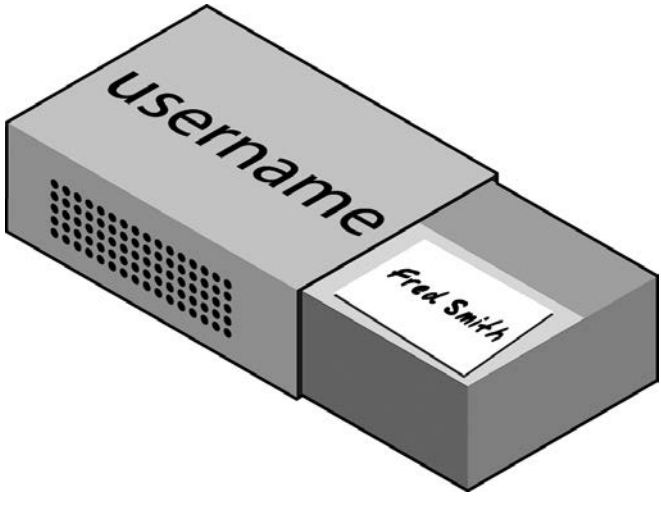

Рис. 3.2. Переменные можно представить в виде спичечного коробка, содержащего какие-то предметы

Кавычки служат признаком того, что `Fred Smith` является *строкой* символов. Каждую строку нужно заключать либо в двойные, либо в одинарные кавычки (апо-

строфы). Между этими двумя видами кавычек есть весьма существенное различие, которое будет рассмотрено далее.

Когда хочется посмотреть, что находится внутри коробка, вы его открываете, вынимаете бумажку и читаете, что на ней написано. В PHP подобное действие (выводящее содержимое переменной на экран) выглядит следующим образом:

```
echo $username;
```

Можно также присвоить содержимое другой переменной (сделать ксерокопию бумажки и поместить ее в другой коробок):

```
$current_user = $username;
```

Если вы стремитесь самостоятельно освоить работу с PHP, то можете попробовать вводить примеры, приводимые в этой главе, в интегрированную среду разработки (согласно рекомендациям, которые были даны в конце главы 2), чтобы тут же посмотреть на результаты, или же можете ввести код примера 3.4 в редактор программ (который также рассматривался в главе 2) и сохранить этот код в каталоге исходного источника документов вашего сервера под именем `test1.php`.

Пример 3.4. Ваша первая PHP-программа

```
<?php // test1.php
  $username = "Fred Smith";
  echo $username;
  echo "<br>";
  $current_user = $username;
  echo $current_user;
?>
```

Теперь эту программу можно запустить, введя в адресную строку браузера следующий адрес:

```
http://localhost/test1.php
```

В маловероятном случае, предполагающем, что в ходе установки веб-сервера (рассмотренной в главе 2) вы изменили назначенный серверу порт на какой-нибудь другой, отличающийся от порта 80, вы должны поместить номер этого порта в URL в данном и всех последующих примерах из этой книги. Например, если вы изменили порт на 8080, то предыдущий URL приобретет следующий вид:

```
http://localhost:8080/test1.php
```

Не забудьте об этом при тестировании других примеров из книги или при написании собственного кода.

Результатом запуска этого кода будет двойное появление имени `Fred Smith`, первое — в результате выполнения команды `echo $username`, а второе — в результате выполнения команды `echo $current_user`.

Числовые переменные

Переменные могут содержать не только строки, но и числа. Если вернуться к аналогии со спичечным коробком, сохранение в переменной `$count` числа 17 будет эквивалентно помещению, скажем, 17 бусин в коробок, на котором написано слово `count`:

```
$count = 17;
```

Можно также использовать числа с плавающей точкой (содержащие десятичную точку). Синтаксис остается прежним:

```
$count = 17.5;
```

Чтобы узнать о содержимом коробка, его нужно просто открыть и посчитать бусины. В PHP можно присвоить значение переменной `$count` другой переменной или вывести его с помощью браузера на экран, воспользовавшись командой `echo`.

Массивы

Массивы можно представить в виде нескольких склеенных вместе спичечных коробков. Например, нам нужно сохранить имена пяти футболистов одной команды в массиве `$team`. Для этого мы склеим вместе боковыми сторонами пять коробков, запишем имена всех игроков на отдельных клочках бумаги и положим каждый клочок в свой коробок.

Вдоль всей верхней стороны склеенных вместе коробков напишем слово `team` (рис. 3.3). В PHP эквивалентом этому действию будет следующий код:

```
$team = array'Bill', 'Mary', 'Mike', 'Chris', 'Anne');
```

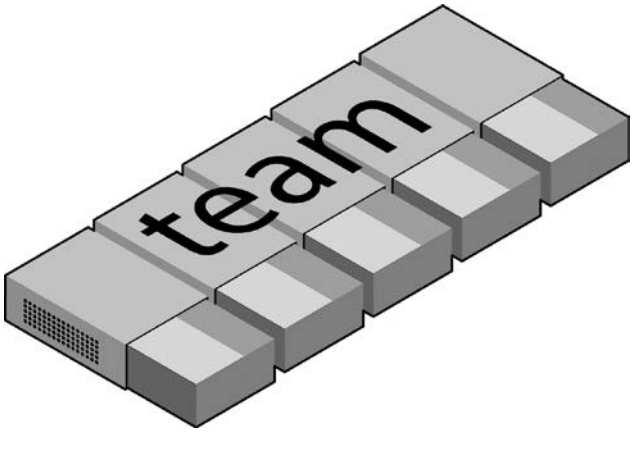

Рис. 3.3. Массив похож на несколько склеенных вместе спичечных коробков

Этот синтаксис несколько сложнее рассмотренных ранее инструкций. Код создания массива представляет собой следующую конструкцию:

```
array();
```

с пятью строками внутри круглых скобок. Каждая строка заключена в одинарные кавычки, и строки должны быть отделены друг от друга запятыми.

Когда потребуется узнать, кто является игроком номер 4, можно воспользоваться следующей командой:

```
echo $team[3]; // Эта команда отображает имя Chris
```

Использование в предыдущем примере числа 3, а не 4 обусловлено тем, что первый элемент PHP-массива является, как правило, нулевым, поэтому номера игроков распределяются в интервале от 0 до 4.

Двумерные массивы

Диапазон использования массивов очень широк. Например, вместо выстраивания одномерных рядов коробков из них можно построить двумерную матрицу, а массивы могут иметь три и более измерения.

Чтобы привести пример двумерного массива, представим, что нужно отслеживать ход игры в крестики-нолики, для чего требуется структура данных, состоящая из девяти клеток, сгруппированных в квадрат 3 × 3. Для наглядности вообразите себе девять коробков, склеенных в матрицу, состоящую из трех строк и трех столбцов (рис. 3.4).

Рис. 3.4. Многомерный массив, смоделированный с помощью коробков

Теперь для каждого хода можно класть в нужные коробки клочки бумаги с крестиком или ноликом. Чтобы сделать это в коде PHP, необходимо создать массив,

содержащий три других массива, как в примере 3.5, в котором массив создается для отображения уже ведущейся игры.

Пример 3.5. Определение двумерного массива

```
<?php
  $oxo = array(array('x', ' ', 'o'),
               array('o', 'o', 'x'),
               array('x', 'o', ' '));
?>
```

Мы сделали еще один шаг к усложнению, но смысл его нетрудно понять, если усвоен основной синтаксис массива. Здесь три конструкции `array()` вложены во внешнюю по отношению к ним конструкцию `array()`. Мы заполнили каждую строку массивом, состоящим только из одного символа: `x`, `o` или пробела. (Мы воспользовались пробелом для того, чтобы все ячейки при отображении были одинаковой ширины.)

Для возвращения в дальнейшем третьего элемента во второй строке этого массива можно воспользоваться следующей PHP-командой, которая отобразит символ «x»:

```
echo $oxo[1][2];
```

Не забывайте о том, что отсчет индексов массива (указателей на элементы внутри массива) начинается с нуля, а не с единицы, поэтому в предыдущей команде индекс [1] ссылается на второй из трех массивов, а индекс [2] — на третью позицию внутри этого массива. Эта команда вернет содержимое третьего слева и второго сверху коробка.

Как уже упоминалось, поддерживаются даже массивы с большей размерностью, получаемые путем простого создания большего количества вложенных друг в друга массивов. Но в этой книге массивы с размерностью больше двух рассматриваться не будут.

Подробнее о массивах мы поговорим в главе 6.

Правила присваивания имен переменным

При создании PHP-переменных следует придерживаться четырех правил.

- Имена переменных после знака доллара должны начинаться с буквы или с символа _ (подчеркивания).
- Имена переменных могут содержать только символы: a–z, A–Z, 0–9 и _ (подчеркивание).
- Имена переменных не должны включать в себя пробелы. Если имя переменной нужно составить более чем из одного слова, то в качестве разделителя рациональнее всего будет использовать символ подчеркивания (например, `$user_name`).

- Имена переменных чувствительны к регистру символов. Переменная `$High_Score` отличается от переменной `$high_score`.

Чтобы позволить использование ASCII-символов, включающих диакритические знаки, PHP также поддерживает в именах переменных байты от 127 и до 255. Но пока ваш код не будет поддерживаться только теми программистами, которые знакомы с такими символами, от их применения лучше отказаться, поскольку программисты, использующие английские раскладки клавиатуры, будут испытывать трудности при доступе к таким символам.

Операторы

Операторы позволяют указать выполняемые математические операции, такие как сложение, вычитание, умножение и деление. Но существует также и ряд других типов операторов, например операторы работы со строками, проведения сравнений и выполнения логических операций. Математические операции в PHP во многом похожи на обычные арифметические записи. Например, в результате работы следующего оператора выводится число 8:

```
echo 6 + 2;
```

Перед тем как приступить к изучению возможностей PHP, следует уделить немного внимания рассмотрению предоставляющих эти возможности различных операторов.

Арифметические операторы

Арифметические операторы проделывают вполне ожидаемую работу. Они применяются для выполнения математических операций. Их можно использовать для проведения четырех основных операций (сложения, вычитания, умножения и деления), а также для нахождения модуля (остатка от деления) и увеличения или уменьшения значения на единицу (табл. 3.1).

Таблица 3.1. Арифметические операторы

Оператор	Описание	Пример
+	Сложение	$j + 1
–	Вычитание	$j – 6
*	Умножение	$j * 11
/	Деление	$j / 4
%	Модуль (остаток от деления)	$j % 9
++	Инкремент (приращение)	++$j
––	Декремент (отрицательное приращение)	––$j
**	Возведение в степень	$j**2

Операторы присваивания

Эти операторы используются для присваивания значений переменным. К ним относится самый простой оператор `=`, а также операторы `+=`, `-=` и т. д. (табл. 3.2). Оператор `+=` вместо полного замещения находящегося слева значения добавляет к нему значение, которое находится справа от него. Итак, если переменная `$count` имела начальное значение `5`, то оператор:

```
$count += 1;
```

устанавливает значение `$count` равным `6` точно так же, как более привычный оператор присваивания:

```
$count = $count + 1;
```

Таблица 3.2. Операторы присваивания

Оператор	Пример	Эквивалент
=	$j = 15	$j = 15
+=	$j += 5	$j = $j + 5
–=	$j –= 3	$j = $j – 3
*=	$j *= 8	$j = $j * 8
/=	$j /= 16	$j = $j / 16
.=	$j .= $k	$j = $j . $k
%=	$j %= 4	$j = $j % 4

Операторы сравнения

Как правило, операторы сравнения используются внутри таких конструкций, как инструкция `if`, в которых требуется сравнивать значения двух элементов. Например, если необходимо узнать, не достигло ли значение переменной, подвергающееся приращению, какого-то конкретного значения или не превышает ли значение другой переменной установленного значения и т. д. (табл. 3.3).

Таблица 3.3. Операторы сравнения

Оператор	Описание	Пример
==	Равно	$j == 4
!=	Не равно	$j != 21
>	Больше	$j > 3
<	Меньше	$j < 100
>=	Больше или равно	$j >= 15
<=	Меньше или равно	$j <= 8

Оператор	Описание	Пример
<>	Не равно	$j <> 23
===	Тождественно	$j === "987"
!==	Не тождественно	$j !== "1.2e3"

Учтите, что операторы = и == предназначены для разных действий. Если первый является оператором присваивания, то второй — оператором сравнения. Иногда в спешке даже более опытные программисты могут вместо одного из них поставить другой, поэтому будьте внимательны, используя эти операторы.

Логические операторы

Если логические операторы вам раньше не встречались, то поначалу они могут показаться чем-то необычным. Нужно представить, что вы делаете логические заключения на простом разговорном языке. Например, можно сказать самому себе: «Если время уже больше 12, но меньше 14 часов, значит нужно пообедать». В PHP код для такого высказывания может выглядеть следующим образом:

```
if ($hour > 12 && $hour < 14) dolunch();
```

Здесь набор инструкций для самого обеда помещен в функцию по имени `dolunch`, которую позже нужно будет создать.

Как видно из предыдущего примера, логический оператор обычно используется для объединения результатов работы двух операторов сравнения, показанных в предыдущем разделе. Результат работы одного логического оператора может служить входным значением для другого логического оператора («Если время уже больше 12, но меньше 14 часов или же если в комнате пахнет жареным и тарелки уже стоят на столе...»). Как правило, если какое-то действие имеет истинное или ложное значение — `TRUE` или `FALSE`, оно может служить входным значением для логического оператора, который берет два истинных или ложных входных значения и выдает в качестве результата истинное или ложное значение. Логические операторы показаны в табл. 3.4.

Таблица 3.4. Логические операторы

Оператор	Описание	Пример
&&	И	$j == 3 && $k == 2
and	Низкоприоритетное И	$j == 3 and $k == 2
||	ИЛИ	$j < 5 || $j > 10
or	Низкоприоритетное ИЛИ	$j < 5 or $j > 10
!	НЕ	! ($j == $k)
xor	Исключающее ИЛИ	$j xor $k

Заметьте, что оператор && обычно взаимозаменяем с оператором AND; то же самое справедливо и для операторов || и OR. Но, поскольку у операторов and и or более низкий приоритет, их применения следует избегать, если только нет никаких других вариантов, как в следующей инструкции, где необходимо использовать оператор or (для того чтобы принудительно выполнялась вторая инструкция, если выполнение первой завершится неудачно, оператор || применяться не может):

```
mysql_select_db($database) or die("Невозможно выбрать базу данных");
```

Наиболее непривычным из этих операторов является xor, предназначенный для операции *исключающего ИЛИ*, который возвращает истинное значение TRUE, если любое из входных значений истинно, и ложное значение FALSE, если оба они имеют значение TRUE или FALSE. Чтобы понять его работу, представьте, что хотите изобрести чистящее средство для дома. Как аммиак (ammonia), так и хлорка (bleach) обладают хорошими чистящими свойствами, поэтому нужно, чтобы ваше средство содержало одно из этих веществ. Но оба они не могут в нем присутствовать, поскольку их сочетание опасно. В PHP это можно представить в следующем виде:

```
$ ingredient = $ammonia xor $bleach;
```

В представленном фрагменте, если любая из двух переменных, $ammonia или $bleach, имеет значение TRUE, то значение переменной $ingredient также будет установлено в TRUE. Но если обе они имеют значение TRUE или значение FALSE, то значение переменной $ingredient будет установлено в FALSE.

Присваивание значений переменным

Синтаксис присваивания значения переменной всегда имеет вид *переменная = значение*. Для передачи значения другой переменной он имеет немного иной вид: *другая_переменная = переменная*.

Есть еще несколько дополнительных операторов присваивания, которые могут оказаться полезными. Например, нам уже встречался оператор:

```
$x += 10;
```

Он предписывает PHP-парсеру добавить значение, расположенное справа от него (в данном случае это значение равно 10), к значению переменной $x. Подобным образом можно вычесть значение:

```
$y -= 10;
```

Увеличение и уменьшение значения переменной на единицу

Добавление или вычитание единицы — настолько часто встречающаяся операция, что PHP предоставляет для этого специальные операторы. Вместо операторов += и -= можно воспользоваться одним из следующих операторов:

```
++$x;
--$y;
```

В сочетании с проверкой (инструкцией `if`) можно воспользоваться таким кодом:

```
if (++$x == 10) echo $x;
```

Этот код предписывает PHP *сначала* увеличить значение переменной `$x` на единицу, а затем проверить, не имеет ли она значение `10`; если переменная имеет такое значение, его следует вывести на экран. Можно также потребовать от PHP увеличить значение переменной на единицу (или, как в следующем примере, уменьшить на единицу) *после* того, как ее значение будет проверено:

```
if ($y-- == 0) echo $y;
```

что дает несколько иной результат. Предположим, что первоначальное значение переменной `$y` до выполнения оператора было равно нулю. Операция сравнения вернет результат `TRUE`, но после того, как она будет проведена, переменной `$y` будет присвоено значение `-1`. Тогда что же отобразит инструкция `echo`: `0` или `-1`? Попробуйте догадаться, а потом, чтобы подтвердить свою догадку, испытайте работу инструкции в PHP-процессоре. Поскольку такая комбинация операторов может вас запутать, ее можно применять только в качестве обучающего примера, но ни в коем случае не рассматривать как приемлемый стиль программирования.

Короче говоря, когда именно увеличено или уменьшено на единицу значение переменной, до или после проверки, зависит от того, где помещен оператор инкремента или декремента — перед именем переменной или после него.

Кстати, правильный ответ на предыдущий вопрос таков: инструкция `echo` отобразит результат `-1`, потому что значение переменной `$y` было уменьшено на единицу сразу же после того, как к ней получила доступ инструкция `if`, и до того, как к ней получила доступ инструкция `echo`.

Объединение строк

Объединение, или как несколько непонятно называют эту операцию, *конкатенация*, обозначает помещение одного содержимого после другого. При объединении строк, когда к одной строке символов добавляется другая строка, используется символ точки (`.`). Самый простой способ объединения строк выглядит следующим образом:

```
echo "У вас " . $msgs . " сообщений.";
```

Если предположить, что переменной `$msgs` присвоено значение `5`, то эта строка кода выведет следующую информацию:

```
У вас 5 сообщений.
```

Так же как с помощью оператора `+=` можно добавить значение к числовой переменной, с помощью оператора `.=` можно добавить одну строку к другой:

```
$bulletin .= $newsflash;
```

В данном случае, если в переменной `$bulletin` содержится сводка новостей, а в переменной `$newsflash` — экстренное сообщение, команда добавляет это сообщение к сводке новостей, и теперь переменная `$bulletin` включает в себя обе строки текста.

Типы строк

В PHP поддерживаются два типа строк, которые обозначаются типом используемых кавычек. Если требуется присвоить переменной значение текстовой строки, сохраняя ее точное содержимое, нужно воспользоваться одинарными кавычками (апострофами):

```
$ info = 'Предваряйте имена переменных символом $, как в данном примере: $variable';
```

В данном случае переменной `$info` присваивается каждый символ, находящийся внутри строки в одинарных кавычках. Если воспользоваться двойными кавычками, то PHP попытается вычислить `$variable` и получить значение переменной.

В то же время, если требуется включить в состав строки значение переменной, используется строка, заключенная в двойные кавычки:

```
echo "На этой неделе ваш профиль просмотрело $count человек ";
```

Из этого следует, что данный синтаксис предлагает более простой вариант объединения, в котором для добавления одной строки к другой не нужно использовать символ точки или закрывать и снова открывать кавычки. Этот прием называется *подстановкой переменной*, и одни программисты пользуются им довольно интенсивно, в то время как другие вообще его не применяют.

Изменение предназначения символов

Иногда в строке должны содержаться символы, которые имеют специальное предназначение и могут быть неправильно интерпретированы. Например, следующая строка кода не будет работать, потому что вторая кавычка (апостроф), встреченная в слове spelling's, укажет PHP-парсеру на то, что достигнут конец строки. Следовательно, вся остальная часть строки будет отвергнута как ошибочная:

```
$text = 'My spelling's atroshus'; // Ошибочный синтаксис
```

Для исправления ошибки нужно непосредственно перед вызывающим неоднозначное толкование символом кавычки добавить обратный слеш, чтобы заставить PHP рассматривать этот символ буквально и не подвергать его интерпретации:

```
$text = 'My spelling\'s still atroshus';
```

Этот прием можно применить практически во всех ситуациях, где в противном случае PHP вернул бы ошибку, пытаясь интерпретировать символ. Например,

следующая строка, заключенная в двойные кавычки, будет присвоена переменной без ошибок:

```
$text = "She wrote upon it, \"Return to sender\".";
```

Кроме того, для вставки в строку различных специальных символов, например табуляции, новой строки и возврата каретки, могут применяться управляющие символы: `\t`, `\n` и `\r` соответственно. Вот пример, в котором символы табуляции используются для разметки заголовка (они включены в строку лишь для иллюстрации применения символа обратного слеша, поскольку существуют более подходящие способы разметки веб-страниц):

```
$heading = "Дата\tИмя\tПлатеж";
```

Эти специальные символы, предваряемые символами обратного слеша, работают только в строках, заключенных в двойные кавычки. Если заключить предыдущую строку в одинарные кавычки, то вместо символов табуляции в ней будут отображены нелепые последовательности символов `\t`. Внутри строк, заключенных в одинарные кавычки, в качестве символов с измененным предназначением распознаются только измененный апостроф (`\'`) и сам измененный обратный слеш (`\\`).

Многострочные команды

Иногда нужно вывести из PHP большой объем текста, а использование нескольких инструкций `echo` (или `print`) заняло бы много времени и было бы неразумным. PHP предлагает два удобных средства, предназначенных для того, чтобы справиться с подобной ситуацией. Первое из них состоит в заключении в кавычки нескольких строк, как в примере 3.6. Переменным также можно присвоить значения способом, показанным в примере 3.7.

Пример 3.6. Инструкция echo, использующая несколько строк

```
<?php
  $author = "Steve Ballmer";

  echo "Developers, Developers, developers, developers,
  developers, developers, developers, developers, developers!

  $author.";
?>
```

Пример 3.7. Многострочное присваивание

```
<?php
  $author = "Bill Gates";
  $text = "Measuring programming progress by lines of code is like
  measuring aircraft building progress by weight.

  - $author.";
?>
```

В PHP можно также воспользоваться многострочной последовательностью, добавив оператор <<<, который обычно называют *here-document* («здесь документ») или *heredoc*. Он представляет собой способ указания строкового литерала, сохраняющего в тексте обрывы строк и другие пустые пространства (включая отступы). Его использование показано в примере 3.8.

Пример 3.8. Еще один вариант инструкции echo, использующей сразу несколько строк

```
<?php
  $author = "Brian W. Kernighan";

  echo <<<_END
  Debugging is twice as hard as writing the code in the first place.
  Therefore, if you write the code as cleverly as possible, you are,
   by definition, not smart enough to debug it.

  $author.
_END;
?>
```

Этот код предписывает PHP вывести все, что находится между двумя тегами `_END`, как будто все это является строкой, заключенной в двойные кавычки (за исключением того, что изменять предназначение кавычек в heredoc не нужно). Это означает, что разработчику можно, например, написать целый раздел HTML-кода прямо в коде PHP, а затем заменить конкретные динамические части переменными PHP.

Важно запомнить, что закрывающий тег `_END`; *должен* появляться строго в начале новой строки и быть *единственным* содержимым этой строки — к ней не разрешается добавлять даже комментарии (нельзя ставить даже одиночный пробел). Как только многострочный блок закрыт, можно снова воспользоваться тем же самым именем тега.

Запомните: используя heredoc-конструкцию <<<_END..._END;, вы не должны добавлять символы \n, чтобы отправить команду на перевод строки, достаточно просто нажать клавишу Enter и приступить к набору новой строки. В отличие от других строк, заключенных в одинарные или двойные кавычки, внутри конструкции heredoc можно по своему усмотрению совершенно свободно пользоваться всеми одинарными или двойными кавычками, не изменяя их первоначального предназначения с помощью обратного слеша (\).

В примере 3.9 показано, как использовать этот же синтаксис для присваивания переменной многострочного значения.

Пример 3.9. Присваивание переменной многострочного значения

```
<?php
  $author = "Scott Adams";

  $out = <<<_END
  Normal people believe that if it ain't broke, don't fix it.
```

```
  Engineers believe that if it ain't broke, it doesn't have enough
  features yet.

  - $author.
_END;
echo $out;
?>
```

После этого переменная `$out` будет наполнена содержимым, размещенным между двумя тегами. Если не присваивать, а добавлять значение, то для добавления строки к значению переменной `$out` вместо оператора `=` можно воспользоваться оператором `.=`.

Будьте внимательны, не ставьте точку с запятой сразу же за первым тегом `_END`, поскольку она прервет многострочный блок еще до его начала и вызовет сообщение об ошибке синтаксического разбора — `Parse error`. Точку с запятой нужно ставить только после закрывающего тега `_END`, хотя внутри блока можно свободно пользоваться точкой с запятой как обычным текстовым символом.

Кстати, тег `_END` — лишь один из многих, я выбрал его для этих примеров, поскольку его использование где-нибудь еще в коде PHP маловероятно, и поэтому он уникален. Вы можете использовать по собственному усмотрению любой тег, например `_SECTION1` или `_OUTPUT` и т. д. И еще, чтобы отличать подобные теги от переменных или функций, обычно в начале их имени ставят знак подчеркивания; но если не хотите, можете им не пользоваться.

Многострочную разметку текста можно рассматривать как удобное средство, упрощающее чтение вашего кода PHP, поскольку, как только текст отображается на веб-странице, вступают в силу правила форматирования HTML и пустые пространства скрываются (но имя переменной $author в нашем примере по-прежнему будет заменяться ее значением в соответствии с правилами вставки значений переменных).

Допустим, если загрузить эти примеры многострочного вывода в браузер, они не будут отображены в виде нескольких строк, потому что все браузеры рассматривают символы новой строки просто как пробелы. Но если воспользоваться свойством браузера, позволяющим просматривать исходный код, обнаружится, что все символы новой строки правильно расставлены и этим кодом PHP сохранены переносы текста на следующие строки.

Типы переменных

PHP относится к очень слабо типизированным языкам. Это значит, что переменные не требуют объявления перед своим использованием и что PHP всегда преобразует переменные в тот тип, который требуется для их окружения на момент доступа к ним.

Например, можно создать число, состоящее из нескольких цифр, и извлечь из него *n*-ю цифру, просто предположив, что это число является строкой. В примере 3.10

перемножаются числа 12 345 и 67 890 и возвращается результат `838102050`, который затем помещается в переменную `$number`.

Пример 3.10. Автоматическое преобразование числа в строку

```
<?php
  $number = 12345 * 67890;
  echo substr($number, 3, 1);
?>
```

Когда присваивается значение, `$number` является числовой переменной. Но во второй строке кода вызов значения этой переменной помещен в PHP-функцию `substr`, которая должна вернуть из переменной `$number` один символ, стоящий на четвертой позиции (не забывайте, что в PHP отсчет позиции начинается с нуля). Для выполнения этой задачи PHP превращает `$number` в строку, состоящую из девяти символов, чтобы функция `substr` могла получить к ней доступ и вернуть символ, в данном случае `1`.

То же самое происходит при необходимости превратить строку в число и т. д. А в примере 3.11 переменной `$pi` присвоено строковое значение, которое затем в третьей строке кода автоматически превращается в число с плавающей точкой, чтобы стать частью уравнения по вычислению площади круга, которое выводит значение `78,5398175`.

Пример 3.11. Автоматическое преобразование строки в число

```
<?php
  $pi     = "3.1415927";
  $radius = 5;
  echo $pi * ($radius * $radius);
?>
```

На практике все это означает, что вам не стоит слишком волноваться за типы переменных. Им следует просто присвоить значения, имеющие для вас смысл, и PHP при необходимости их преобразует. Затем, если понадобится извлечь значение, их нужно просто запросить, например, с помощью инструкции `echo`.

Константы

Константы, как и переменные, хранят информацию для последующего доступа, за исключением того, что они оправдывают свое название констант (постоянных). Иными словами, после определения констант их значения устанавливаются для всей остальной программы и не могут быть изменены.

К примеру, константа может использоваться для хранения местоположения корневого каталога вашего сервера (папки, содержащей основные файлы вашего сайта). Определить такую константу можно следующим образом:

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Затем для чтения содержимого константы нужно просто сослаться на нее как на обычную переменную (но не предваряя ее имя знаком доллара):

```
$directory = ROOT_LOCATION;
```

Теперь, как только понадобится запустить ваш PHP-код на другом сервере с другой конфигурацией папок, придется изменить только одну строку кода.

Важно помнить о двух основных особенностях констант: перед их именами не нужно ставить символ $ (в отличие от имен обычных переменных) и их можно определить только с помощью функции define.

По общепринятому соглашению считается правилом хорошего тона использовать в именах констант буквы только верхнего регистра, особенно если ваш код будет читать кто-нибудь другой.

Предопределенные константы

PHP поставляется в виде готового продукта с десятками предопределенных констант, которые редко используются новичками в программировании. Тем не менее существуют константы, известные как *волшебные*, которые могут оказаться для вас полезными с самого начала. У имен волшебных констант в начале и в конце всегда стоят два символа подчеркивания, чтобы нельзя было случайно назвать одну из собственных констант уже занятым под эти константы именем. Подробности о волшебных константах приведены в табл. 3.5. Понятия, упомянутые в таблице, будут раскрыты в следующих главах.

Таблица 3.5. Волшебные константы PHP

Волшебная константа	Описание
LINE	Номер текущей строки в файле
FILE	Полное путевое имя файла. Если используется внутри инструкции include, то возвращается имя включенного файла. В некоторых операционных системах допускается использование псевдонимов для каталогов, которые называются символическими ссылками; в _FILE_ они всегда заменяются реальными каталогами
DIR	Каталог файла. Если используется внутри инструкции include, то возвращается каталог включенного файла. Такой же результат дает применение функции dirname(FILE). В этом имени каталога отсутствует замыкающий слеш, если только этот каталог не является корневым (добавлена в PHP 5.3.0)

Продолжение ⇨

Таблица 3.5 *(продолжение)*

Волшебная константа	Описание
FUNCTION	Имя функции. Начиная с PHP 5, возвращает имя функции, под которым она была объявлена (с учетом регистра символов). В PHP 4 возвращаемое значение всегда составлено из символов нижнего регистра (добавлена в PHP 4.3.0)
CLASS	Имя класса. Начиная с PHP 5, возвращает имя класса, под которым он был объявлен (с учетом регистра символов). В PHP 4 возвращаемое значение всегда составлено из символов нижнего регистра (добавлена в PHP 4.3.0)
METHOD	Имя метода класса. Возвращает имя метода, под которым он был объявлен (с учетом регистра символов) (добавлена в PHP 5.0.0)
NAMESPACE	Имя текущего пространства имен (с учетом регистра символов). Эта константа определена во время компиляции (добавлена в PHP 5.3.0)

Эти константы полезны при отладке, когда нужно вставить строку кода, чтобы понять, до какого места дошло выполнение программы:

```
echo "Это строка " . _LINE_ . " в файле " . _FILE_;
```

Эта команда выведет в браузер текущую строку программы с указанием текущего файла, исполняемого в данный момент (включая путь к нему).

Различие между командами echo и print

Нам уже встречались разнообразные способы использования команды `echo` для вывода текста с сервера в браузер. В одних случаях выводится строковый литерал, в других сначала происходило объединение строк или вычисление значений переменных. Был также показан вывод, распространяющийся на несколько строк.

Но команде `echo` есть альтернатива, которой также можно воспользоваться: команда `print`. Эти две команды очень похожи друг на друга, но `print` — конструкция, похожая на функцию, воспринимающую единственный параметр и имеющую возвращаемое значение (которое всегда равно `1`), а `echo` — в чистом виде конструкция языка PHP.

В общем, команда `echo` обычно работает при выводе обычного текста быстрее `print`, поскольку она не устанавливает возвращаемое значение. С другой стороны, поскольку она не является функцией, ее, в отличие от `print`, нельзя использовать как часть более сложного выражения. В следующем примере для вывода информации о том, является значение переменной истинным (`TRUE`) или ложным (`FALSE`), используется функция `print`, но сделать то же самое с помощью команды `echo` не представляется возможным, поскольку она выведет на экран сообщение об ошибке синтаксического разбора — `Parse error`:

```
$b ? print "TRUE" : print "FALSE";
```

Использование вопросительного знака — самый простой способ задать вопрос о том, какое значение имеет переменная `$b`, истинное или ложное. Команда, которая располагается слева от двоеточия, выполняется в том случае, если `$b` имеет истинное значение, а команда, которая располагается справа, выполняется, если `$b` имеет ложное значение.

Тем не менее в приводимых здесь примерах чаще всего используется команда `echo`, и я рекомендую применять именно ее до тех пор, пока вам при PHP-разработке реально не потребуется задействовать функцию `print`.

Функции

Функции используются для выделения блоков кода, выполняющих конкретную задачу. Например, если вам часто приходится искать какие-то данные и выводить их в определенном формате, то вполне разумно будет обратиться к функции. Код, выполняющий эту задачу, может занимать всего три строки, но, пока вы не воспользуетесь функцией, необходимость вставлять этот код в программу десятки раз делает ее неоправданно большой и сложной. А если вы чуть позже захотите изменить формат вывода данных, помещение кода в функцию будет означать, что вам придется внести изменения только в одном месте программы.

Помещение кода в функцию не только сокращает размер программы и делает ее более удобной для чтения, но и дает дополнительные функциональные возможности (эта игра слов носит преднамеренный характер), поскольку функциям могут передаваться параметры, которые вносят изменения в характер их работы. Функции также могут возвращать значения вызывающему их коду.

Чтобы создать функцию, нужно ее объявить, как показано в примере 3.12.

Пример 3.12. Простое объявление функции

```
<?php
  function longdate($timestamp)
  {
    return date("l F jS Y", $timestamp);
  }
?>
```

Эта функция возвращает дату в формате «Вторник май 2 2021». Между стоящими после имени функции круглыми скобками может размещаться любое количество параметров, но для этой функции выбран прием только одного параметра. Весь код, который выполняется при последующем вызове функции, заключается в фигурные скобки. Обратите внимание, что в этом примере первой буквой в вызове функции даты является L в нижнем регистре, которую не следует путать с цифрой 1.

Чтобы с помощью этой функции вывести сегодняшнюю дату, нужно поместить в свой код следующий вызов:

```
echo longdate(time0);
```

Если требуется вывести дату семнадцатидневной давности, нужно сделать следующий вызов:

```
echo longdate(time() - 17 * 24 * 60 * 60);
```

в котором функции `longdate` передается текущая отметка времени, уменьшенная на количество секунд, которое прошло за 17 дней (17 дней × 24 ч × 60 мин × 60 с).

Функции могут также воспринимать несколько параметров и возвращать несколько результатов, используя технологию, которая будет рассмотрена в следующих главах.

Область видимости переменной

Если программа очень длинная, то с подбором подходящих имен переменных могут возникнуть трудности, но, программируя на PHP, можно определить *область видимости* переменной. Иными словами, можно, к примеру, указать, что переменная `$temp` будет использоваться только внутри конкретной функции, чтобы забыть о том, что она после возврата из кода функции применяется где-нибудь еще. Фактически именно такой в PHP является по умолчанию область видимости переменных.

В качестве альтернативы можно проинформировать PHP о том, что переменная имеет глобальную область видимости и доступ к ней может быть осуществлен из любого места программы.

Локальные переменные

Локальные переменные создаются внутри функции, и к ним имеется доступ только из кода этой функции. Обычно это временные переменные, которые используются до выхода из функции для хранения частично обработанных результатов.

Одним из наборов локальных переменных является перечень аргументов функции. В предыдущем разделе была определена функция, воспринимающая параметр по имени `$timestamp`. Значение этого параметра действительно только в теле функции, за пределами этой функции его значение нельзя ни получить, ни установить.

Чтобы привести еще один пример локальной переменной, рассмотрим функцию `longdate` еще раз в немного измененном варианте (пример 3.13).

Пример 3.13. Расширенная версия функции longdate

```
<?php
  function longdate($timestamp)
  {
    $temp = date("l F jS Y", $timestamp);
    return "Дата: $temp";
  }
?>
```

В этом примере значение, возвращенное функцией `date`, присваивается временной переменной `$temp`, которая затем вставляется в строку, возвращаемую определяемой функцией. Как только будет осуществлен выход из функции, переменная `$temp` и ее содержимое исчезают, как будто они вообще никогда не использовались.

Теперь, чтобы посмотреть на области видимости переменных в действии, изучим похожий код, показанный в примере 3.14. Здесь переменная `$temp` была создана еще до вызова функции `longdate`.

Пример 3.14. Неудачная попытка получить доступ к переменной $temp в функции longdate

```
<?php
  $temp = "Дата: ";
  echo longdate(time());

  function longdate($timestamp)
  {
    return $temp . date("l F jS Y", $timestamp);
  }
?>
```

Но поскольку переменная `$temp` не была создана внутри функции `longdate`, а также не была передана ей в качестве параметра, функция `longdate` не может получить к ней доступ. Поэтому этот фрагмент кода выведет только дату без предшествующего ей текста. На самом деле в зависимости от параметров конфигурации PHP сначала может быть отображено сообщение об ошибке, предупреждающее об использовании неопределенной переменной (`Notice: Undefined variable: temp`), показывать которое пользователям не хотелось бы.

Причина в том, что по умолчанию переменные, созданные внутри функции, являются локальными для нее, а переменные, созданные за пределами любой функции, могут быть доступны только из того кода, который не входит в код ни одной из функций.

В примерах 3.15 и 3.16 показано несколько способов исправления кода, приведенного в примере 3.14.

Пример 3.15. Решить проблему можно путем переноса ссылки на переменную $temp в ее локальную область видимости

```
<?php
  $temp = "Дата: ";
  echo $temp . longdate(time());

  function longdate($timestamp)
  {
    return date("l F jS Y", $timestamp);
  }
?>
```

В примере 3.15 ссылка на `$temp` перемещена за пределы функции. Эта ссылка появляется в той же области видимости, в которой была определена переменная.

Пример 3.16. Альтернативное решение: передача $temp в качестве аргумента

```
<?php
  $temp = "Дата: ";
  echo longdate($temp, time());

  function longdate($text, $timestamp)
  {
    return $text . date("l F jS Y", $timestamp);
  }
?>
```

В примере 3.16 принято другое решение: передать значение переменной `$temp` функции `longdate` в качестве дополнительного аргумента. Функция `longdate` считывает это значение во временную переменную, которую она создает под именем `$text`, и выводит желаемый результат.

Программисты часто допускают ошибку, забывая об области видимости переменных, поэтому, если не помнить принципы ее работы, это поможет в отладке некоторых весьма неочевидных ошибок программного кода. Достаточно сказать, что, если вы не объявили переменную каким-нибудь особым образом, ее область видимости ограничена локальным пространством: либо в пределах кода текущей функции, либо в пределах кода, не принадлежащего никаким функциям, в зависимости от того, где эта переменная была впервые создана или где к ней впервые был получен доступ, внутри функции или за ее пределами.

Глобальные переменные

Бывают случаи, когда требуется переменная, имеющая *глобальную* область видимости, поскольку нужно, чтобы к ней имелся доступ из всего кода программы. К тому же некоторые данные могут быть настолько объемными и сложными, что их не захочется передавать функциям в качестве аргументов.

Чтобы получить доступ к переменным из глобальной области видимости, следует добавить ключевое слово `global`. Предположим, что существует некий способ входа пользователей на ваш сайт и нужно, чтобы весь код программы знал, с кем он имеет дело — с зарегистрированным пользователем или с гостем. Один из способов решения этой задачи заключается в применении перед именем переменной ключевого слова `global`, как это сделано с переменной `$is_logged_in`:

```
global $is_logged_in;
```

Теперь вашей функции входа в систему нужно лишь при удачной попытке входа на сайт присвоить этой переменной значение `1`, а при неудачной попытке — зна-

чение `0`. Поскольку переменная обладает глобальной областью видимости, доступ к ней может быть получен из любой строки кода вашей программы.

Но пользоваться глобальными переменными нужно с оглядкой. Я рекомендую создавать их только в том случае, если без них совершенно невозможно добиться нужного результата. Вообще-то программы, разбитые на небольшие фрагменты и отдельные данные, содержат меньше ошибок и проще в обслуживании. Если ваша программа состоит из нескольких тысяч строк кода (а когда-нибудь такое случится) и оказалось, что где-то глобальная переменная имеет неверное значение, то сколько времени уйдет на поиски кода, который присваивает ей это значение?

Кроме того, если задействуется слишком много переменных с глобальной областью видимости, возникает риск воспользоваться одним из их имен еще раз в локальном пространстве или по крайней мере полагать, что такая переменная применяется локально, хотя на самом деле она уже была объявлена в качестве глобальной. Из таких ситуаций и возникают разные непонятные ошибки.

Иногда я придерживаюсь соглашения о написании имен всех переменных, требующих глобальной доступности, в верхнем регистре (что совпадает с рекомендациями о написании в этом же регистре имен констант), чтобы можно было с первого взгляда определить область видимости переменной.

Статические переменные

В пункте «Локальные переменные» выше было упомянуто, что значение локальной переменной стирается сразу же после выхода из функции. Если функция вызывается многократно, она начинает свою работу со свежей копией переменной и ее прежние установки не имеют никакого значения.

Интересно, а что, если внутри функции есть такая локальная переменная, к которой не должно быть доступа из других частей программы, но значение которой желательно сохранять до следующего вызова функции? Зачем? Возможно, потому, что нужен некий счетчик, чтобы следить за количеством вызовов функции. Решение, показанное в примере 3.17, заключается в объявлении *статической переменной*.

Пример 3.17. Функция, использующая статическую переменную

```
<?php
  function test()
  {
    static $count = 0;
    echo $count;
    $count++;
  }
?>
```

В этом примере в самой первой строке функции создается статическая переменная по имени `$count`, которой присваивается нулевое начальное значение. В следующей строке выводится значение переменной, а в последней строке это значение увеличивается на единицу.

При следующем вызове функции, поскольку переменная `$count` уже была объявлена, первая строка функции пропускается и до нового увеличения значения переменной `$count` отображается ее предыдущее значение.

Планируя использование статических переменных, следует учесть, что при их определении присвоить им результат какого-нибудь выражения невозможно. Они могут инициализироваться только предопределенными значениями (пример 3.18).

Пример 3.18. Допустимые и недопустимые объявления статических переменных

```
<?php
  static $int = 0;          // Допустимо
  static $int = 1+2;        // Недопустимо (вызовет ошибку
                            // синтаксического разбора (Parse error))
  static $int = sqrt(144);  // Недопустимо
?>
```

Суперглобальные переменные

Начиная с версии PHP 4.1.0, стали доступны некоторые предопределенные переменные. Они известны как *суперглобальные переменные*. Смысл этого названия заключается в том, что они предоставляются средой окружения PHP и имеют глобальную область видимости внутри программы, то есть доступны абсолютно из любого ее места.

В этих суперглобальных переменных содержится масса полезной информации о текущей работающей программе и ее окружении (табл. 3.6). Такие переменные имеют структуру ассоциативных массивов, которые будут рассмотрены в главе 6.

Таблица 3.6. Суперглобальные переменные PHP

Имя суперглобальной переменной	Ее содержимое
$GLOBALS	Все переменные, которые на данный момент определены в глобальной области видимости сценария. Имена переменных служат ключами массива
$_SERVER	Информация о заголовках, путях, местах расположения сценариев. Элементы этого массива создаются веб-сервером, и это не дает гарантии, что каждый веб-сервер будет предоставлять какую-то часть информации или ее всю
$_GET	Переменные, которые передаются текущему сценарию методом HTTP GET

Имя суперглобальной переменной	Ее содержимое
$_POST	Переменные, которые передаются текущему сценарию методом HTTP POST
$_FILES	Элементы, подгруженные к текущему сценарию методом HTTP POST
$_COOKIE	Переменные, переданные текущему сценарию посредством HTTP cookies
$_SESSION	Переменные сессии, доступные текущему сценарию
$_REQUEST	Содержимое информации, переданной от браузера; по умолчанию $_GET, $_POST и $_COOKIE
$_ENV	Переменные, переданные текущему сценарию методом environment

В именах всех суперглобальных переменных (за исключением `$GLOBALS`) присутствует один знак подчеркивания и используются только заглавные буквы, поэтому, чтобы избежать путаницы, не следует присваивать своим переменным имена, оформленные в таком же стиле.

Для иллюстрации порядка применения суперглобальных переменных разберем типовой пример. Среди многой другой интересной информации, предоставляемой суперглобальными переменными, есть и URL-адрес той страницы, с которой пользователь был перенаправлен на текущую веб-страницу. Эта информация может быть получена следующим образом:

```
$came_from = $_SERVER['HTTP_REFERRER'];
```

Как видите, ничего сложного. Если же пользователь зашел непосредственно на вашу страницу, к примеру набрав ее URL-адрес прямо в браузере, переменной `$came_from` будет присвоена пустая строка.

Суперглобальные переменные и проблемы безопасности

Обратите внимание, что суперглобальные переменные часто используются злоумышленниками, пытающимися отыскать средства для атаки и вмешательства в работу вашего сайта. Они загружают в `$_POST`, `$_GET` или в другие суперглобальные переменные вредоносный код, например команды UNIX или MySQL, которые, если вы по незнанию к ним обратитесь, могут повредить данные или отобразить незащищенные данные.

Именно поэтому перед применением суперглобальные переменные всегда следует подвергать предварительной обработке. Для этого можно воспользоваться PHP-функцией `htmlentities`. Она занимается преобразованием всех символов в элементы HTML. Например, символы «меньше» и «больше» (< и >) превращаются в строки `<` и `>`, то же самое делается для перевода в безопасное состояние всех кавычек, обратных слешей и т. д.

Поэтому более подходящий способ доступа к $_SERVER (и другим суперглобальным переменным) выглядит следующим образом:

```
$came from = htmlentities($ SERVER['HTTP REFERRER']);
```

Использование для санации функции htmlentities считается важной практикой не только в отношении суперглобальных переменных, но и при любых обстоятельствах, в которых данные, исходящие от пользователя или поступающие из сторонних источников, обрабатываются для получения выходных данных.

В этой главе были заложены надежные основы, необходимые для работы с PHP. В главе 4 мы приступим к практическому использованию изученного материала для построения выражений и управления ходом программы, иными словами, перейдем к реальному программированию.

Но перед изучением новой главы я рекомендую проверить свои знания, ответив на приведенные далее вопросы, чтобы убедиться в том, что вы полностью усвоили содержимое этой главы.

Вопросы

1. Какой тег PHP служит основанием для того, чтобы приступить к интерпретации программного кода? Как выглядит краткая форма этого тега?
2. Какие два вида тегов используются для добавления комментариев?
3. Какой символ должен стоять в конце каждой инструкции PHP?
4. Какой символ используется в начале имен всех переменных PHP?
5. Что может храниться в переменных?
6. В чем разница между выражениями `$variable = 1` и `$variable == 1`?
7. Как вы считаете, почему подчеркивания разрешено использовать в именах переменных (например, `$current_user`), а дефисы — нет (например, `$current-user`)?
8. Чувствительны ли имена переменных к регистру букв?
9. Можно ли в именах переменных использовать пробелы?
10. Как преобразовать значение одного типа переменной в значение другого типа (скажем, строку в число)?
11. В чем разница между `++$j` и `$j++`?
12. Являются ли взаимозаменяемыми операторы `&&` и `and`?
13. Как создается многострочный вывод: с использованием команды `echo` или присвоением многострочного значения?
14. Можно ли переопределить константу?

15. Как изменить исходное предназначение кавычки?
16. В чем разница между командами `echo` и `print`?
17. Каково назначение функций?
18. Как сделать переменную доступной для всего кода PHP-программы?
19. Какими двумя способами можно передать всей остальной программе данные, которые были созданы внутри функции?
20. Что является результатом объединения строки с числом?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

4 Выражения и управление процессом выполнения программы в PHP

В предыдущей главе уже упоминались темы, которые более полно будут рассмотрены в данной главе, например выбор (ветвление) и создание сложных выражений. В главе 3 мне хотелось сконцентрировать внимание на наиболее общих вопросах синтаксиса и работы в PHP, но при этом невозможно было не затронуть темы более высокого уровня. А вот теперь можно преподнести вам основы, необходимые для полноценного использования всех сильных сторон PHP.

В этой главе будет заложен фундамент практики программирования на PHP и рассмотрены основные способы управления процессом выполнения программы.

Выражения

Начнем с базовой части любого языка программирования — *выражения*.

Выражение представляет собой сочетание значений, переменных, операторов и функций, в результате вычисления которого выдается новое значение. Оно знакомо всем, кто когда-либо имел дело с обыкновенной школьной алгеброй. Рассмотрим пример:

```
y = 3(abs|2x| + 4)
```

что в PHP приобретает следующий вид:

```
$y = 3 * (abs(2*$x) + 4);
```

Возвращаемое значение (в математическом выражении *y* или в PHP `$y`) может быть числом, строкой или *булевым (логическим)* значением (названным так в честь Джорджа Буля, английского математика и философа XIX века). Первые два типа значений вам уже должны быть знакомы, поэтому я объясню, что такое третий тип.

TRUE или FALSE?

Элементарное булево значение может быть либо истинным — TRUE, либо ложным — FALSE. Например, выражение 20 > 9 (20 больше 9) является истинным (TRUE), а выражение 5 == 6 (5 равно 6) — ложным (FALSE). (Булевы, или логические, операции могут быть объединены путем использования таких операторов, как И, ИЛИ и исключающее ИЛИ, то есть AND, OR и XOR, которые будут рассмотрены в этой главе.)

Обратите внимание, что для имен TRUE и FALSE я использую буквы верхнего регистра. Это обусловлено тем, что в PHP они являются предопределенными константами. При желании можно также применять и их версии, составленные из букв нижнего регистра, поскольку они также являются предопределенными константами. Кстати, версия, в которой задействуются буквы нижнего регистра, более надежна, потому что PHP не допускает ее переопределения, а версия, использующая буквы верхнего регистра, может быть переопределена, и это нужно иметь в виду при импортировании чужого кода.

Если, как показано в примере 4.1, потребовать вывод на экран предопределенных констант, PHP их не выведет. В каждой строке код примера выводит букву с двоеточием и предопределенную константу. PHP самовольно сопоставляет со значением TRUE числовое значение 1, поэтому при выполнении кода примера после a: выводится 1. И еще загадочнее выглядит то, что попытка вывести сразу за b: значение FALSE приводит к тому, что на экране вместо этого значения вообще ничего не появляется. Константа FALSE в PHP определяется как еще одна ничем не обозначенная предопределенная константа NULL.

Пример 4.1. Вывод на экран значений TRUE и FALSE

```
<?php // test2.php
  echo "a: [" . TRUE  . "]<br>";
  echo "b: [" . FALSE . "]<br>";
?>
```

Здесь при помощи тегов
 создаются переводы на новую строку, разбивающие вывод на экран в HTML на две строки. Этот вывод выглядит следующим образом:

```
a: [1]
b: []
```

Вернемся к булевым выражениям. В коде примера 4.2 показывается несколько простых выражений: два уже упомянутых ранее и еще два дополнительных.

Пример 4.2. Четыре простых булевых выражения

```
<?php
  echo "a: [" (20 > 9) . "]<br>";
  echo "b: [" (5 == 6) . "]<br>";
  echo "c: [" (1 == 0) . "]<br>";
  echo "d: [" (1 == 1) . "]<br>";
?>
```

Этот код выведет следующую информацию:

```
a: [1]
b: []
c: []
d: [1]
```

К счастью, булевы выражения чаще всего окружены другим кодом и обычно вас абсолютно не волнует, как выглядит внутреннее отображение TRUE и FALSE. Фактически даже эти имена довольно редко попадаются в коде.

Литералы и переменные

Это базовые элементы программирования, которые к тому же являются строительными блоками выражений. Под *литералом* просто подразумевается нечто, вычисляющееся само в себя, например число 73 или строка "Hello". Переменная, с которой мы уже знакомы, обладает именем, начинающимся со знака доллара, и вычисляется в присвоенное этой переменной значение. Простейшее выражение имеет вид одиночного литерала или переменной, поскольку они возвращают значение.

В примере 4.3 показаны три литерала и две переменные, все они возвращают значения, хотя и разных типов.

Пример 4.3. Литералы и переменные

```
<?php
  $myname = "Brian";
  $myage = 37;

  echo "a: " . 73      . "<br>";  // Числовой литерал
  echo "b: " . "Hello" . "<br>";  // Строковый литерал
  echo "c: " . FALSE   . "<br>";  // Литерал константы
  echo "d: " . $myname . "<br>";  // Строковая переменная
  echo "e: " . $myage  . "<br>";  // Числовая переменная
?>
```

Как и ожидалось, в выходной информации вы увидите возвращаемое значение всех этих выражений, за исключением c:, результат вычисления которого равен FALSE и ничего не возвращает:

```
a: 73
b: Hello
c:
d: Brian
e: 37
```

Объединив простейшие выражения с операторами, можно создать более сложные выражения, результатом вычисления которых будут какие-нибудь полезные результаты.

Чтобы сформировать *инструкции*, программисты комбинируют выражения с другими конструкциями языка, такими как рассмотренные ранее операторы присваивания.

В примере 4.4 показаны две инструкции. В первой из них осуществляется присваивание результата выражения `366 - $day_number` переменной `$days_to_new_year`, а во второй выводится приветственное сообщение, если выражение `$days_to_new_year < 30` вычисляется как `TRUE`.

Пример 4.4. Выражение и инструкция

```
<?php
  $days_to_new_year = 366 - $day_number; // Выражение

  if ($days_to_new_year < 30)
  {
    echo "Скоро Новый год!";             // Инструкция
  }
?>
```

Операторы

В PHP имеется множество мощных операторов: от арифметических, строковых и логических до операторов присваивания, сравнения и многих других (табл. 4.1).

Таблица 4.1. Типы операторов PHP

Оператор	Описание	Пример
Арифметический	Элементарная математика	$a + $b
Для работы с массивом	Слияние массивов	$a + $b
Присваивания	Присваивание значений	$a = $b + 23
Поразрядный	Манипуляция битами в байте	12 ^ 9
Сравнения	Сравнение двух значений	$a < $b
Выполнения	Выполнение содержимого, заключенного в обратные кавычки	`ls –al`
Инкремента/ декремента	Добавление или вычитание единицы	$a++
Логический	Выполнение булевых сравнений	$a and $b
Строковый	Объединение строк	$a . $b

Различные типы операторов воспринимают разное количество операндов.

- *Унарные* операторы, такие как оператор инкремента (`$a++`) или отрицания (`!$a`), воспринимают только один операнд.

- *Бинарные* операторы, представленные большим количеством операторов PHP, включая операторы сложения, вычитания, умножения и деления, воспринимают два операнда.
- Один *трехкомпонентный* оператор, имеющий форму `expr ? x : y` и требующий наличия трех операндов. По сути, это состоящая из трех частей однострочная инструкция `if`, возвращающая `x`, если `expr` вычисляется в `TRUE`, и `y`, если `expr` вычисляется в `FALSE`.

Приоритетность операторов

Если бы у всех операторов был один и тот же уровень приоритета, то они обрабатывались бы в том порядке, в котором встречались интерпретатору. Фактически многие операторы имеют одинаковый уровень приоритета, что и показано в примере 4.5.

Пример 4.5. Три эквивалентных выражения

```
1 + 2 + 3 - 4 + 5
2 - 4 + 5 + 3 + 1
5 + 2 - 4 + 1 + 3
```

Из примера видно, что, несмотря на перестановку чисел (и предшествующих им операторов), результат каждого выражения — `7`, поскольку у операторов «плюс» и «минус» одинаковый уровень приоритета. Можно проделать то же самое с операторами умножения и деления (пример 4.6).

Пример 4.6. Три выражения, которые также являются эквивалентными

```
1 * 2 * 3 / 4 * 5
2 / 4 * 5 * 3 * 1
5 * 2 / 4 * 1 * 3
```

В этом примере итоговое значение всегда равно `7.5`. Но все меняется, когда в выражении присутствуют операторы с разными уровнями приоритета, как в примере 4.7.

Пример 4.7. Три выражения, в которых присутствуют операторы с разными уровнями приоритета

```
1 + 2 * 3 - 4 * 5
2 - 4 * 5 * 3 + 1
5 + 2 - 4 + 1 * 3
```

Если бы не существовало приоритетности операторов, то в результате вычисления этих выражений получались бы числа `25`, `-29` и `12` соответственно. Но поскольку операторы умножения и деления имеют более высокий уровень приоритета по сравнению с операторами сложения и вычитания, выражения вычисляются так, будто эти части выражений заключены в скобки, как математическая запись, показанная в примере 4.8.

Пример 4.8. Три выражения, в которых отображены предполагаемые скобки

```
1 + (2 * 3) - (4 * 5)
2 - (4 * 5 * 3) + 1
5 + 2 - 4 + (1 * 3)
```

Сначала PHP вычисляет подвыражения, заключенные в скобки, чтобы получились частично вычисленные выражения, показанные в примере 4.9.

Пример 4.9. Выражения после вычисления подвыражений в скобках

```
1 + (6) - (20)
2 - (60) + 1
5 + 2 - 4 + (3)
```

Окончательный результат вычисления этих выражений равен соответственно `-13`, `-57` и `6` (что абсолютно отличается от результатов `25`, `-29` и `12`, которые мы увидели бы при отсутствии приоритетности операторов).

Разумеется, исходную приоритетность операторов можно отменить, расставив собственные скобки, и принудительно получить тот порядок вычислений, который вам нужен (пример 4.10).

Пример 4.10. Принудительное выполнение вычислений справа налево

```
((1 + 2) * 3 - 4) * 5
(2 - 4) * 5 * 3 + 1
(5 + 2 - 4 + 1) * 3
```

Теперь, если скобки расставлены правильно, мы увидим значения `25`, `-29` и `12` соответственно.

В табл. 4.2 перечислены операторы PHP в порядке их приоритетности от самого высокого до самого низкого уровня.

Таблица 4.2. Операторы PHP, расположенные по уровню их приоритетности (сверху вниз)

Оператор	Тип
()	Скобки
++ --	Инкремент/декремент
!	Логический
* / %	Арифметические
+ - .	Арифметические и строковые
<< >>	Побитовые
< <= > >= <>	Сравнения
= = != === !==	Сравнения

Продолжение ⇨

Таблица 4.2 *(продолжение)*

Оператор	Тип
&	Поразрядный (и ссылочный)
^	Поразрядный
|	Поразрядный
&&	Логический
||	Логический
? :	Трехкомпонентный
= += –= *= /= .= %= &= != ^= <<= >> =	Присваивания
and	Логический
xor	Логический
or	Логический

Порядок следования операторов в этой таблице не является произвольным. Он тщательно разработан таким образом, что наиболее распространенными и интуитивно понятными по приоритетности операторами можно пользоваться без применения скобок. Например, два выражения сравнения можно разделить оператором `and` или `or` и получить вполне ожидаемый результат.

Взаимосвязанность операторов

Мы рассматривали обработку выражений слева направо, за исключением тех случаев, в которых вступала в силу приоритетность операторов. Но некоторые операторы могут также потребовать обработки справа налево. Направление обработки обусловливается *взаимосвязанностью* операторов. Для отдельных операторов взаимосвязанность отсутствует.

Взаимосвязанность (подробно показана в табл. 4.3) приобретает большое значение в тех случаях, когда вы явным образом не меняете приоритетности. Для этого вам нужно знать о действиях операторов по умолчанию.

Таблица 4.3. Взаимосвязанность операторов

Оператор	Описание	Взаимосвязанность
< <= >= == != === !== <>	Сравнение	Отсутствует
!	Логическое НЕ	Правая
~	Поразрядное НЕ	Правая
++ ––	Инкремент и декремент	Правая

Оператор	Описание	Взаимосвязанность
(int)	Преобразование в целое число	Правая
(double) (float) (real)	Преобразование в число с плавающей точкой	Правая
(string)	Преобразование в строку	Правая
(array)	Преобразование в массив	Правая
(object)	Преобразование в объект	Правая
@	Подавление сообщения об ошибке	Правая
= += –= *= /=	Присваивание	Правая
.– %= &= |= ^= <<= >> =	Присваивание	Правая
+	Сложение и унарный плюс	Левая
–	Вычитание и отрицание	Левая
*	Умножение	Левая
/	Деление	Левая
%	Модуль	Левая
.	Конкатенация строк	Левая
<< >> & ^ |	Поразрядные операции	Левая
?:	Операция с тремя операндами	Левая
|| && and or xor	Логические операции	Левая
,	Разделение	Левая

Рассмотрим оператор присваивания, показанный в примере 4.11, где всем трем переменным присваивается значение `0`.

Пример 4.11. Оператор множественного присваивания

```
<?php
  $level = $score = $time = 0;
?>
```

Такое множественное присваивание возможно только в том случае, если сначала вычисляется самая правая часть выражения, а затем процесс продолжается справа налево.

Новичкам следует научиться в процессе работы с PHP избегать потенциальных просчетов в вопросах взаимосвязанности операторов и всегда принудительно задавать порядок вычислений, заключая подвыражения в круглые скобки. Это поможет и другим программистам, которые будут обслуживать ваш код, понять, что в нем происходит.

Операторы отношения

Операторы отношения отвечают на такие вопросы, как «Имеет ли эта переменная нулевое значение» и «У какой переменной значение больше». Эти операторы проверяют значения двух операндов и возвращают логический результат, равный либо TRUE, либо FALSE. Существует три типа операторов отношения: *операторы равенства*, *сравнения* и *логические* операторы.

Операторы равенства

С оператором равенства == (двойным знаком равенства) мы уже не раз встречались в этой книге. Его не следует путать с оператором присваивания = (одинарным знаком равенства). В примере 4.12 первый оператор присваивает значение, а второй проверяет его на равенство.

Пример 4.12. Присваивание значения и проверка его на равенство

```
<?php
  $month = "Март";

  if ($month == "Март") echo "Весна наступила";
?>
```

Как видно из примера, возвращая значение TRUE или FALSE, оператор сравнения позволяет проверять условия, используя инструкцию if. Но это еще не все, поскольку PHP является языком со слабой типизацией. Если два операнда выражения равенства имеют разные типы, PHP преобразует их к тому типу, который имеет для него наибольший смысл. Редко используемый оператор *тождественности*, состоящий из трех подряд знаков равенства, можно задействовать для сравнения элементов без выполнения преобразования.

К примеру, любые строки, составленные полностью из цифр, при сравнении с числами будут преобразованы в числа. В примере 4.13 переменные $a и $b являются двумя разными строками, и поэтому вряд ли стоило ожидать, что какая-то из инструкций if выведет результат.

Пример 4.13. Операторы равенства и тождественности

```
<?php
  $a = "1000";
  $b = "+1000";

  if ($a == $b)  echo "1";
  if ($a === $b) echo "2";
?>
```

Но если запустить этот пример, то он выведет число. Это означает, что результат вычисления первой инструкции if равен TRUE. Причина в том, что обе строки сначала конвертируются в числа и 1000 имеет такое же числовое значение, что и +1000. В отличие от первой, во второй инструкции if используется оператор

тождественности, следовательно переменные $a и $b сравниваются как строки и теперь считаются отличающимися друг от друга, и на экран ничего не выводится.

Как и в случае с принудительным заданием уровня приоритетности операторов, если возникнут сомнения в том, будет ли PHP конвертировать типы операндов, для отмены такого поведения интерпретатора можно воспользоваться оператором тождественности.

Аналогично применению оператора равенства для определения равенства операндов можно проверить их на неравенство, используя оператор *неравенства* !=. Пример 4.14 является измененным примером 4.13, в котором операторы равенства и тождественности были заменены противоположными им операторами.

Пример 4.14. Операторы неравенства и нетождественности

```
<?php
  $a = "1000";
  $b = "+1000";

  if ($a != $b)  echo "1";
  if ($a !== $b) echo "2";
?>
```

Как, наверное, и ожидалось, первая инструкция if не выводит на экран число 1, потому что в коде ставится вопрос о неравенстве числовых значений переменных $a и $b. Вместо этого будет выведено число 2, поскольку вторая инструкция if ставит вопрос о нетождественности $a и $b друг другу в их текущем строковом типе, и ответом будет TRUE, потому что они не тождественны.

Операторы сравнения

Используя операторы сравнения, можно расширить круг проверок, не ограничивая его только равенством и неравенством. PHP предоставляет вам для этого операторы > (больше), < (меньше), >= (больше или равно) и <= (меньше или равно). В примере 4.15 показано использование этих операторов.

Пример 4.15. Четыре оператора сравнения

```
<?php
  $a = 2; $b = 3;

  if ($a > $b)  echo "$a больше $b<br>";
  if ($a < $b)  echo "$a меньше $b<br>";
  if ($a >= $b) echo "$a больше или равно $b<br>";
  if ($a <= $b) echo "$a меньше или равно $b<br>";
?>
```

Этот пример, в котором переменная $a имеет значение 2, а переменная $b — значение 3, выведет на экран следующую информацию:

```
2 меньше 3
2 меньше или равно 3
```

Попробуйте самостоятельно запустить этот пример, меняя значения переменных `$a` и `$b`, чтобы увидеть результаты. Присвойте им одинаковые значения и посмотрите, что из этого получится.

Логические операторы

Логические (или *булевы*) операторы выдают истинные или ложные результаты. Всего имеется четыре таких оператора (табл. 4.4).

Таблица 4.4. Логические операторы

Логический оператор	Описание
AND	Возвращает истинное значение (TRUE), если оба операнда имеют истинные значения
OR	Возвращает истинное значение (TRUE), если любой из операндов имеет истинное значение
XOR	Возвращает истинное значение (TRUE), если один из двух операндов имеет истинное значение
! (NOT)	Возвращает истинное значение (TRUE), если операнд имеет ложное значение, или ложное значение (FALSE), если он имеет истинное значение

Использование этих операторов показано в примере 4.16. Обратите внимание, что PHP требует указывать вместо слова `NOT` символ `!`. Кроме того, операторы могут быть составлены из букв нижнего или верхнего регистра.

Пример 4.16. Использование логических операторов

```
<?php
  $a = 1; $b = 0;

  echo ($a AND $b)  . "<br>";
  echo ($a or $b)   . "<br>";
  echo ($a XOR $b)  . "<br>";
  echo !$a          . "<br>";
?>
```

Этот пример выводит построчно на экран `NULL`, `1`, `1`, `NULL`. Это значит, что только вторая и третья инструкции `echo` получают в результате вычисления значение `TRUE`. (Следует помнить, что `NULL`, или ничто, отображает значение `FALSE`.) Такой результат получается, потому что оператору `AND` для возвращения значения `TRUE` нужно, чтобы оба операнда имели истинное значение, а четвертый оператор проводит над значением переменной `$a` операцию `NOT`, превращая его из `TRUE` (значения, равного единице) в `FALSE`. Если есть желание поэкспериментировать, запустите этот код, присваивая переменным `$a` и `$b` разные значения, выбранные из `1` и `0`.

Занимаясь программированием, следует помнить, что у операторов AND и OR более низкий уровень приоритета, чем у других версий этих операторов — && и ||.

Использование в инструкции `if` оператора `OR` может стать причиной непредвиденных проблем, поскольку второй операнд не будет вычисляться, если в результате вычисления первого операнда уже получено значение `TRUE`. В примере 4.17 функция `getnext` никогда не будет вызвана, если переменная `$finished` имеет значение `1`.

Пример 4.17. Инструкция, использующая оператор OR

```
<?php
  if ($finished == 1 OR getnext() == 1) exit;
?>
```

Если нужно, чтобы функция `getnext` вызывалась для каждой инструкции `if`, следует внести в код изменения, показанные в примере 4.18.

Пример 4.18. Изменения в инструкции if ... OR, гарантирующие вызов функции getnext

```
<?php
  $gn = getnext();

  if ($finished == 1 OR $gn == 1) exit;
?>
```

В этом случае код в функции `getnext` будет выполнен и возвращенное значение сохранится в переменной `$gn` еще до выполнения инструкции `if`.

Другое решение заключается в том, чтобы обеспечить выполнение функции getnext за счет простой перестановки условий местами, поскольку тогда вызов функции будет появляться в выражении первым.

В табл. 4.5 показаны все допустимые варианты использования логических операторов. Следует заметить, что `!TRUE` является эквивалентом `FALSE`, а `!FALSE` — эквивалентом `TRUE`.

Таблица 4.5. Все логические выражения, допустимые в PHP

Входные данные		**Операторы и результаты**		
a	**b**	**AND**	**OR**	**XOR**
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE

Условия

Условия изменяют процесс выполнения программы. Они позволяют задавать конкретные вопросы и по-разному реагировать на полученные ответы. Условия играют важную роль при разработке динамических веб-страниц — основной цели использования PHP, поскольку облегчают создание разных вариантов информации, выводимой при каждом просмотре веб-страницы.

В этом разделе будут представлены условные конструкции трех основных типов: инструкции `if` и `switch` и оператор `?`. Кроме того, будут представлены циклические условные конструкции (к изучению которых мы вскоре перейдем), код которых выполняется снова и снова до тех пор, пока не будет соблюдено определенное условие.

Инструкция if

Процесс выполнения программы можно представить себе как езду на машине по однополосной магистрали. Эта магистраль большей частью прямолинейна, но иногда встречаются различные дорожные знаки, задающие направление движения.

Когда встречается инструкция `if`, можно представить, что машина подошла к знаку объезда, предписаниям которого необходимо следовать, если определенные условия вычисляются как `TRUE`. При этом вы съезжаете с магистрали и следуете по объездному пути до тех пор, пока не вернетесь снова на магистраль и не продолжите движение по исходному маршруту. Или же, если условие не вычисляется как `TRUE`, вы игнорируете объезд и продолжаете ехать по магистрали как ни в чем не бывало (рис. 4.1).

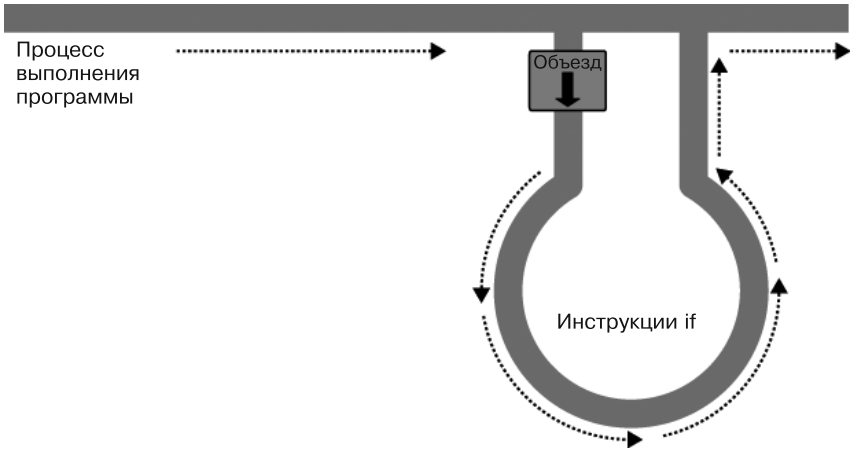

Рис. 4.1. Процесс выполнения программы похож на движение по однополосной магистрали

Содержимым условной инструкции `if` может быть любое допустимое PHP-выражение, включая проверку на равенство, выражения сравнения, проверку на нуль и `NULL` и даже значения, возвращаемые функциями (как встроенными, так и созданными самостоятельно).

Действия, предпринимаемые при вычислении условия `if` в `TRUE`, помещаются, как правило, в фигурные скобки (`{ }`). Но эти скобки можно опустить, если нужно выполнить всего одну инструкцию. Тем не менее, если всегда использовать фигурные скобки, можно избежать «охоты» на трудно отслеживаемые ошибки, возникающие, к примеру, когда к условной инструкции добавляется еще одна строка, но забывается о необходимости добавить фигурные скобки, из-за чего строка не вычисляется.

Пресловутая уязвимость системы безопасности, известная как ошибка goto fail, многие годы преследовала код Secure Sockets Layer (SSL) в продуктах Apple, когда программист забывал заключить тело инструкции if в фигурные скобки, и это приводило к тому, что функция временами выдавала отчет об успешном подключении, хотя по факту так было не всегда. Это позволяло злоумышленникам получать признание сертификата безопасности в тех условиях, когда он должен был быть отклонен. Если есть сомнения, лучше все же помещать тело инструкций if в фигурные скобки.

Учтите, что в целях экономии места и доходчивости материала в тех случаях, когда в примерах, приводимых в книге, была всего одна исполняемая инструкция, я не следовал этому совету и опускал фигурные скобки.

В примере 4.19 следует представить, что подошел конец месяца и нужно платить по всем счетам, поэтому вы проводите некоторые операции с банковским счетом.

Пример 4.19. Инструкция if, в которой используются фигурные скобки

```
<?php
  if ($bank_balance < 100)
  {
    $money        = 1000;
    $bank_balance += $money;
  }
?>
```

В этом примере проверяется, не стал ли баланс ниже $100 (или 100 единиц другой используемой вами валюты). Если баланс стал ниже этой суммы, вы платите сами себе $1000, а затем прибавляете их к балансу. (Хорошо бы так просто зарабатывать деньги!)

Если баланс счета в банке равен $100 или превышает эту сумму, условные инструкции игнорируются и процесс выполнения программы переходит на следующую строку кода (которая здесь не показана).

Одни разработчики предпочитают ставить первую фигурную скобку справа от условного выражения, а другие начинают с нее новую строку. В этой книге открывающая фигурная скобка обычно располагается на новой строке. Подойдет любой из этих вариантов, поскольку PHP позволяет оставлять на ваше усмотрение какие угодно свободные пространства (пробелы, символы новых строк и табуляции). Но код будет легче читаться и отлаживаться, если у каждого уровня условий будет свой отступ, сформированный с помощью символа табуляции.

Инструкция else

Бывают случаи, когда условие не вычисляется как `TRUE`, но вам не хочется сразу же продолжать выполнение основного кода программы, а вместо этого нужно сделать что-либо другое. Тогда пригодится инструкция `else`. С ее помощью на вашей магистрали можно организовать второй объезд, показанный на рис. 4.2.

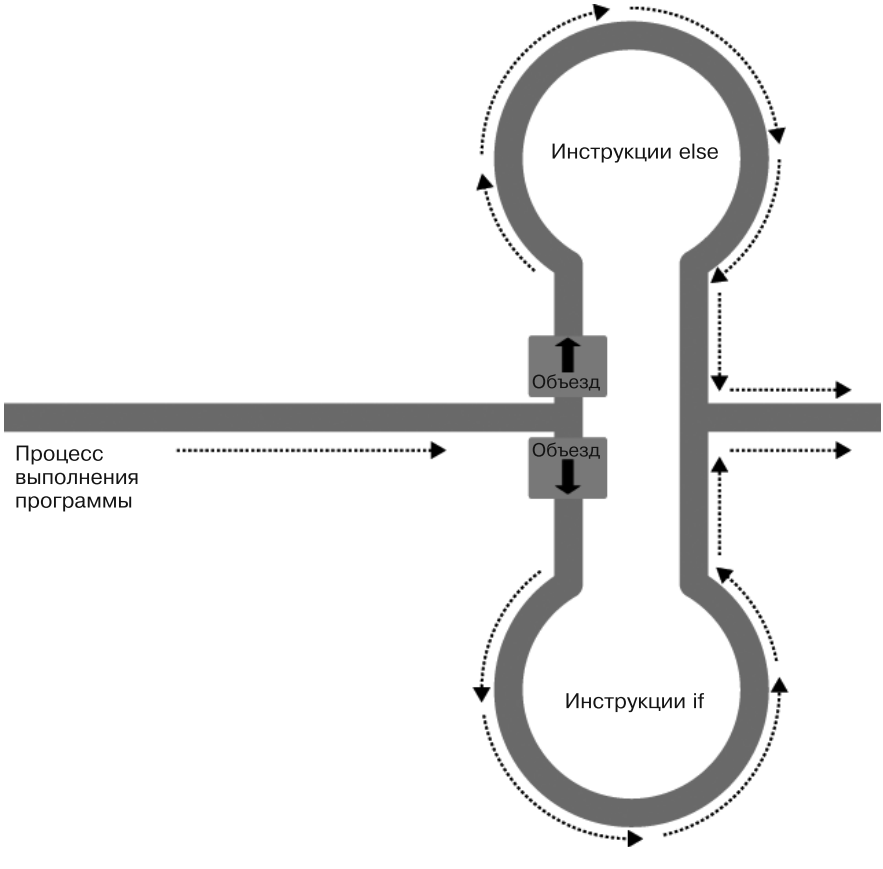

Рис. 4.2. Теперь у магистрали есть объезд if и объезд else

Если при использовании конструкции `if...else` условие вычисляется как `TRUE`, то выполняется первая условная инструкция. Но если это условие вычисляется как `FALSE`, то выполняется вторая условная инструкция. Для выполнения должна быть выбрана одна из этих двух инструкций, но обе сразу они не будут выполнены ни при каких условиях, и обязательно будет выполнена хотя бы одна из них. Использование конструкции `if...else` показано в примере 4.20.

Пример 4.20. Конструкция if...else, в которой используются фигурные скобки

```
<?php
  if ($bank_balance < 100)
  {
    $money         = 1000;
    $bank_balance += $money;
  }
  else
  {
    $savings      += 50;
    $bank_balance -= 50;
  }
?>
```

Если в этом примере будет установлено, что в банке лежит $100 или более, то выполняется инструкция `else`, с помощью которой часть этих денег перемещается на ваш сберегательный счет.

Точно так же, как и у `if`, если у инструкции `else` есть только одна условная инструкция, то фигурные скобки можно не ставить. (Хотя фигурные скобки рекомендуется использовать в любом случае. Во-первых, при их наличии проще разобраться в коде, а во-вторых, они облегчают последующее добавление инструкций к этому ветвлению.)

Инструкция elseif

Случается, что на основе последовательности условий нужно осуществить сразу несколько действий. Достичь желаемого результата можно, используя инструкцию `elseif`. Можно предположить, что она похожа на инструкцию `else`, за исключением того, что до кода условия вставляется еще одно условное выражение. Полноценная конструкция `if...elseif...else` показана в примере 4.21.

Пример 4.21. Конструкция if...elseif...else, в которой используются фигурные скобки

```
<?php
  if ($bank_balance < 100)
  {
    $money         = 1000;
    $bank_balance += $money;
  }
  elseif ($bank_balance > 200)
```

```
  {
    $savings      += 100;
    $bank_balance -= 100;
  }
  else
  {
    $savings      += 50;
    $bank_balance -= 50;
  }
?>
```

В этом примере инструкция `elseif` была вставлена между инструкциями `if` и `else`. Она проверяет, не превышает ли баланс банковского счета сумму $200, и если превышает, принимается решение о том, что в этом месяце можно позволить себе положить на сберегательный счет $100.

Это все можно представить в виде набора объездов в нескольких направлениях (рис. 4.3).

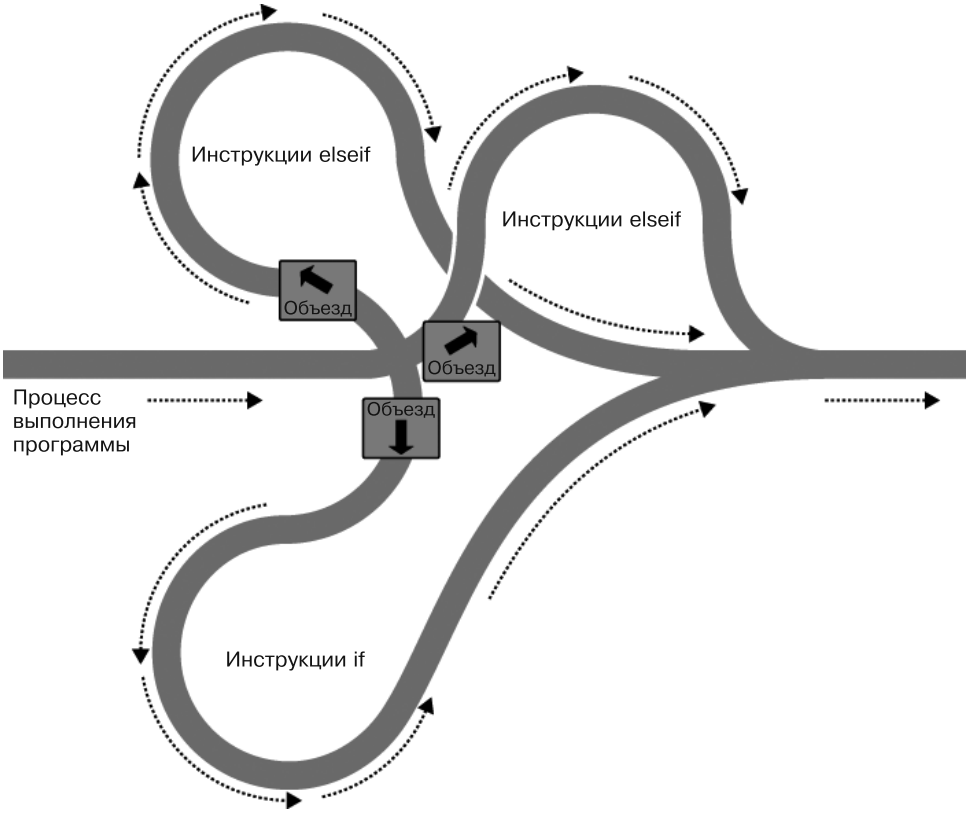

Рис. 4.3. Магистраль с объездами if, elseif и else

Инструкция else завершает либо конструкцию if...else, либо конструкцию if... elseif...else. Если она не нужна, то финальную инструкцию else можно опустить, но ни одна из этих инструкций не должна стоять перед инструкцией elseif, точно так же, как ни одна инструкция elseif не должна стоять перед инструкцией if.

Количество используемых инструкций `elseif` не ограничено. Но по мере роста количества этих инструкций будет лучше, наверное, обратиться к инструкции `switch`, если, конечно, она отвечает вашим запросам. Именно ее мы сейчас и рассмотрим.

Инструкция switch

Инструкция `switch` применяется в тех случаях, когда у одной переменной или у результата вычисления выражения может быть несколько значений, каждое из которых должно вызывать особое действие.

Рассмотрим, например, управляемую кодом PHP систему меню, которая в соответствии с пожеланием пользователя передает отдельную строку коду основного меню. Предположим, что есть следующие варианты: Home, About, News, Login и Links, а переменная `$page` принимает одно из этих значений в соответствии с информацией, введенной пользователем.

Код реализации этого замысла с использованием конструкции `if...elseif...else` может иметь вид, показанный в примере 4.22.

Пример 4.22. Многострочная инструкция if...elseif

```
<?php
  if     ($page == "Home")  echo "Вы выбрали Home";
  elseif ($page == "About") echo "Вы выбрали About";
  elseif ($page == "News")  echo "Вы выбрали News";
  elseif ($page == "Login") echo "Вы выбрали Login";
  elseif ($page == "Links") echo "Вы выбрали Links";
  else                      echo "Нераспознанный выбор";
?>
```

Код, в котором используется инструкция `switch`, показан в примере 4.23.

Пример 4.23. Инструкция switch

```
<?php
  switch ($page)
  {
    case "Home":
        echo "Вы выбрали Home";
        break;
    case "About":
        echo "Вы выбрали About";
        break;
    case "News":
        echo "Вы выбрали News";
```

```
        break;
    case "Login":
        echo "Вы выбрали Login";
        break;
    case "Links":
        echo "Вы выбрали Links";
        break;
  }
?>
```

Как видите, переменная `$page` используется только один раз — в начале инструкции `switch`. После этого все соответствия проверяются командой `case`. Когда найдено соответствие, выполняется его условная инструкция. Разумеется, в настоящей программе в этом месте будет применяться код отображения или перехода на страницу, а не простое сообщение пользователю о том, что именно он выбрал.

В инструкциях switch внутри команд case фигурные скобки не используются. Вместо этого инструкции начинаются с двоеточия и заканчиваются командой break. Тем не менее весь перечень команд case в инструкции switch заключается в фигурные скобки.

Прекращение работы инструкции switch

Если нужно, чтобы инструкция `switch` прекратила свою работу из-за выполнения условия, используется команда `break`. Она предписывает PHP выйти из инструкции `switch` и перейти к выполнению следующей инструкции.

Если в примере 4.23 не расставить команды `break` и результат вычисления команды `case`, проверяющей условие `Home`, получится `TRUE`, то будут выполнены все пять условных инструкций, следующих за командами `case`. Или же если переменная `$page` имела значение `News`, то, начиная с этого места, будут выполнены все оставшиеся команды `case`. Это сделано преднамеренно для расширения возможностей программирования, но в большинстве случаев не следует забывать ставить команду `break` во всех местах, где набор условных инструкций, следующих за командами `case`, завершает свою работу. Надо сказать, что случайный пропуск команд `break` является весьма распространенной ошибкой.

Действие по умолчанию

Типичным требованием для инструкции `switch` является переход к действию по умолчанию, если не будет выполнено ни одно из условий, содержащихся в командах `case`. Например, к коду меню, показанному в примере 4.23, можно непосредственно перед закрывающей фигурной скобкой добавить код, показанный в примере 4.24.

Пример 4.24. Инструкция default для добавления к примеру 4.23

```
default:
    echo "Нераспознанный выбор";
    break;
```

Тем самым повторяется эффект инструкции `else` из примера 4.22.

Здесь ставить команду `break` не требуется, поскольку `default` является заключительной внутренней инструкцией и процесс выполнения программы автоматически продолжится после закрывающей фигурной скобки, но если вы решите поставить инструкцию `default` выше этого места, ей определенно понадобится команда `break`, для того чтобы процесс выполнения программы не затронул все стоящие ниже условные инструкции. Лучше перестраховаться и в конце этой инструкции всегда ставить команду `break`.

Альтернативный синтаксис

Открывающую фигурную скобку инструкции `switch` можно заменить двоеточием, а закрывающую — командой `endswitch` (пример 4.25).

Такой вариант используется довольно редко, и здесь он упоминается на тот случай, если придется столкнуться с ним в коде, созданном кем-нибудь другим.

Пример 4.25. Альтернативный синтаксис инструкции switch

```
<?php
  switch ($page):
    case "Home":
      echo "Вы выбрали Home";
      break;

    // и т. д.

    case "Links":
      echo "Вы выбрали Links";
      break;
  endswitch;
?>
```

Оператор ?

Использование трехкомпонентного оператора `?` позволяет избежать многословности инструкций `if` и `else`. Необычность этого оператора заключается в том, что он использует не два, как большинство других операторов, а три операнда.

В главе 3 уже состоялось краткое знакомство с этим оператором при выяснении разницы между `print` и `echo`, где он приводился в качестве примера оператора, который хорошо работает с `print`, но не работает с `echo`.

Оператору ? передаются выражение, которое он должен вычислить, и два выполняемых оператора: один для выполнения, когда результат вычисления выражения TRUE, а другой — когда FALSE.

В примере 4.26 показан код, который может использоваться для вывода предупреждения об уровне топлива в автомобиле на его панель приборов.

Пример 4.26. Использование оператора ?

```
<?php
  echo $fuel <= 1 ? "Требуется дозаправка" : "Топлива еще достаточно";
?>
```

Если топлива остается всего 1 галлон[1] или меньше (иными словами, переменная $fuel имеет значение, равное единице или меньше ее), то этот оператор возвращает предыдущей команде echo строку Требуется дозаправка. В противном случае он возвращает строку Топлива еще достаточно. Значение, возвращаемое оператором ?, можно также присвоить какой-нибудь переменной (пример 4.27).

Пример 4.27. Присваивание условного результата оператора ? переменной

```
<?php
  $enough = $fuel <= 1 ? FALSE : TRUE;
?>
```

В этом примере переменной $enough будет присвоено значение TRUE только в том случае, если в баке более 1 галлона топлива, в противном случае ей будет присвоено значение FALSE.

Если вы считаете синтаксис оператора ? слишком запутанным, то можете вместо него воспользоваться инструкцией if, но о нем все равно нужно знать, поскольку он может встретиться в программном коде, созданном другим программистом. Чтение кода, в котором используется этот оператор, может быть сильно затруднено из-за частого применения в нескольких местах одной и той же переменной. Например, весьма популярен код такого вида:

```
$saved = $saved >= $new ? $saved : $new;
```

Понять, что он делает, можно только после тщательного разбора:

```
$saved =              // Присваивание значения переменной $saved...
        $saved >= $new // Сравнение $saved и $new
    ?                 // Если сравнение выдает истинный результат...
        $saved        // ...ей присваивается текущее значение $saved
    :                 // Если сравнение выдает ложный результат...
        $new;         // ...ей присваивается значение переменной $new
```

Это весьма компактный способ отслеживания самого большого значения, которое может встретиться в процессе выполнения программы. Самое большое значение

[1] Галлон (американский) = 3,79 л. — *Примеч. ред.*

содержится в переменной `$saved` и при поступлении нового значения сравнивается со значением переменной `$new`. Программисты, освоившие оператор `?`, считают, что для таких коротких сравнений его удобнее применять, чем инструкции `if`.

Если этот оператор не используется для создания компактного кода, то он обычно применяется для принятия решений, умещающихся на одной строке, например для проверки того, установлено ли значение переменной перед передачей ее функции.

Организация циклов

Компьютеры славятся своей способностью быстро и неутомимо повторять вычисления. Зачастую от программы требуется снова и снова повторять одну и ту же последовательность кода, пока не произойдет какое-нибудь событие, например ввод значения пользователем или достижение программой своего естественного окончания. Имеющиеся в PHP разнообразные структуры организации циклов предоставляют великолепные способы решения подобных задач.

Чтобы представить, как это работает, посмотрите на рис. 4.4. Здесь мы вновь обратимся к метафоре с магистралью, которая использовалась для иллюстрации работы инструкции `if`, за исключением того, что у объезда также есть замкнутый участок, из которого машина может выйти только при соблюдении определенных программных условий.

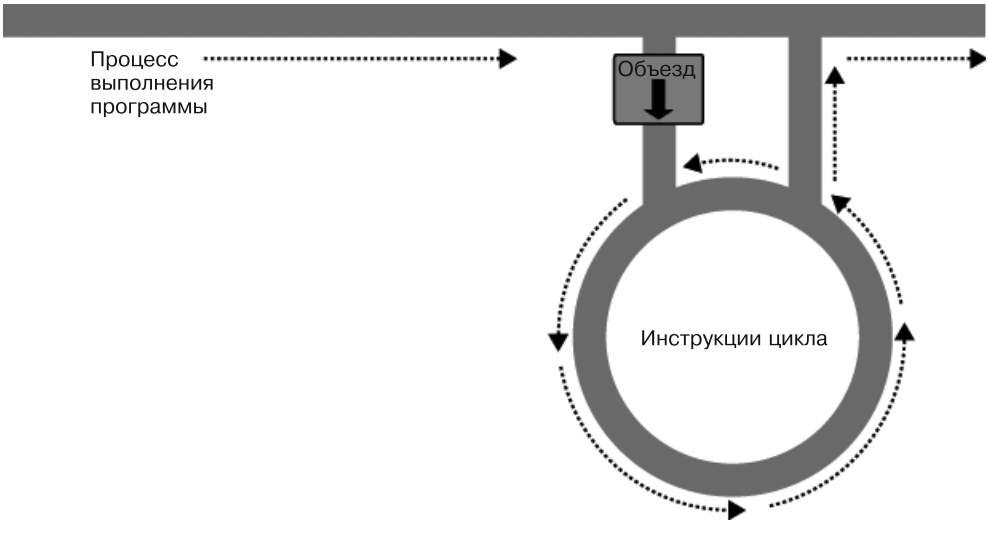

Рис. 4.4. Представление цикла как части программы магистральной разметки

Циклы while

Превратим автомобильную панель приборов из примера 4.26 в цикл, постоянно проверяющий уровень топлива при езде на машине, в котором используется инструкция цикла `while` (пример 4.28).

Пример 4.28. Цикл while

```
<?php
  $fuel = 10;

  while ($fuel > 1)
  {
    // Продолжение поездки...
    echo "Топлива еще достаточно";
  }
?>
```

Вообще-то, вы можете предпочесть выводу текста горящий зеленый сигнал, но суть в том, что любая разновидность позитивной индикации об уровне топлива помещается в цикл `while`. Кстати, учтите, что если вы запустите этот пример на выполнение, то он будет постоянно выводить строку, до тех пор пока вы не остановите работу браузера.

Здесь, как и в случае с инструкциями if, для хранения инструкций внутри цикла while используются фигурные скобки, если только в этом цикле не задействована лишь одна инструкция.

В примере 4.29 показан еще один вариант использования цикла `while`, в котором выводится таблица умножения на 12.

Пример 4.29. Цикл while для вывода таблицы умножения на 12

```
<?php
  $count = 1;

  while ($count <= 12)
  {
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br>";
    ++$count;
  }
?>
```

В этом примере переменной `$count` присваивается начальное значение `1`, а затем запускается цикл `while`, в котором используется выражение сравнения `$count <= 12`. Цикл будет выполняться до тех пор, пока значение переменной не станет больше 12. Данный код выведет следующий текст:

```
Число 1, умноженное на 12, равно 12
Число 2, умноженное на 12, равно 24
Число 3, умноженное на 12, равно 36
и т. д.
```

Внутри цикла осуществляется вывод строки, а также значения переменной `$count`, умноженного на 12. Чтобы упорядочить вывод, после всего этого использован тег `<br>`, вызывающий переход на новую строку. Затем перед закрывающей фигурной скобкой, предписывающей PHP вернуться к началу цикла, значение переменной `$count` увеличивается на единицу.

Теперь значение переменной `$count` опять проверяется, чтобы узнать, не превышает ли оно число 12. Оно не превышает этого числа, но теперь равно `2`, и после 11 последующих прохождений цикла станет равно `13`. Когда это произойдет, код, находящийся внутри цикла `while`, будет пропущен и станет выполняться код, следующий за циклом, в данном случае это будет завершение программы.

При отсутствии оператора `++$count` (вместо которого с таким же успехом может быть применен оператор `$count++`) этот цикл будет похож на первый, показанный в этом разделе. Он никогда не закончится и будет снова и снова выводить один и тот же результат `1 * 12`.

Но есть и более изящный способ написания этого цикла, который должен вам понравиться. Посмотрите на код примера 4.30.

Пример 4.30. Укороченная версия примера 4.29

```
<?php
  $count = 0;

  while (++$count <= 12)
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br>";
?>
```

В этом примере была возможность перемещения оператора `++$count` из тела цикла `while` непосредственно в выражение условия цикла. Теперь PHP вычисляет значение переменной `$count` в начале каждого прохода цикла (итерации) и, заметив, что перед именем переменной стоит оператор инкремента, сначала увеличивает значение переменной на 1 и только потом сравнивает его с числом 12. Следовательно, теперь переменной `$count` присваивается начальное значение `0`, а не `1`, поскольку это значение увеличивается сразу же, как только происходит вход в цикл. Если оставить начальное значение, равное `1`, то будут выведены результаты для чисел между 2 и 12.

Циклы do...while

Цикл `do...while` представляет собой небольшую модификацию цикла `while`, используемую в том случае, когда нужно, чтобы блок кода был исполнен хотя бы один раз, а условие проверялось только после этого.

В примере 4.31 показана модифицированная версия таблицы умножения на 12, в которой использован этот цикл.

Пример 4.31. Цикл do...while, используемый для вывода таблицы умножения на 12

```
<?php
  $count = 1;
  do
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br>";
  while (++$count <= 12);
?>
```

Заметьте, что теперь мы вернулись к присваиванию переменной `$count` начального значения `1` (а не `0`), потому что находящаяся в цикле инструкция `echo` выполняется перед появлением у нас возможности увеличения значения переменной на 1. Во всем остальном этот код очень похож на показанный в примере 4.29.

Разумеется, если внутри цикла `do...while` находится несколько инструкций, то не следует забывать ставить вокруг них фигурные скобки, как показано в примере 4.32.

Пример 4.32. Расширенная версия примера 4.31, использующая фигурные скобки

```
<?php
  $count = 1;
  do {
    echo "Число $count, умноженное на 12, равно " . $count * 12;
    echo "<br>";
  } while (++$count <= 12);
?>
```

Циклы for

Цикл `for`, являющийся последней разновидностью инструкций цикла, к тому же еще и самый мощный из них, поскольку в нем сочетаются возможности установки значения переменных при входе в цикл, проверки соблюдения условия при каждом проходе цикла (итерации) и модификации значений переменных после каждой итерации.

В примере 4.33 продемонстрирована возможность вывода таблицы умножения с использованием цикла `for`.

Пример 4.33. Вывод таблицы умножения на 12 из цикла for

```
<?php
  for ($count = 1 ; $count <= 12 ; ++$count)
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br>";
?>
```

Как видите, весь код сведен к одной инструкции `for`, в которой содержится одна условная инструкция. И вот что из этого получается. Каждая инструкция `for` воспринимает три параметра:

- выражение инициализации;
- выражение условия;
- выражение модификации.

Эти три выражения отделяются друг от друга точкой с запятой: `for (`*`выражение1 ; выражение2 ; выражение3`*`)`. В начале первой итерации выполняется выражение инициализации. В нашем коде таблицы умножения переменная `$count` инициализируется значением `1`. Затем при каждой итерации проверяется выражение условия (в данном случае `$count <= 12`), и выход из цикла осуществляется только в том случае, если результат вычисления условия будет `TRUE`. И наконец, в завершение каждой итерации выполняется выражение модификации. В случае с таблицей умножения значение переменной `$count` увеличивается на 1.

Эта структура в явном виде исключает любые требования по размещению управляющих элементов цикла в его собственном теле, освобождая его для инструкций, требующих циклического выполнения.

Если в теле цикла `for` содержится несколько инструкций, не забудьте воспользоваться фигурными скобками (пример 4.34).

Пример 4.34. Цикл for из примера 4.33 с добавлением фигурных скобок

```
<?php
  for ($count = 1 ; $count <= 12 ; ++$count)
  {
    echo "Число $count, умноженное на 12, равно " . $count * 12;
    echo "<br>";
  }
?>
```

Сравним условия, при которых следует использовать циклы `for`, с условиями, при которых нужно применять циклы `while`. Цикл `for` явно создавался под отдельное значение, изменяющееся на постоянную величину. Обычно мы имеем дело с увеличивающимся значением — это похоже на то, как если бы вам передали перечень того, что выбрал пользователь, и от вас бы требовалось обработать каждый его выбор по очереди. Но переменную можно видоизменять по вашему усмотрению. Более сложная форма инструкции `for` позволяет даже осуществлять со всеми тремя параметрами сразу несколько операций:

```
for ($i = 1, $j = 1 ; $i + $j < 10 ; $i++ , $j++)
{
  // ...
}
```

Однако новичкам использовать такую сложную форму не рекомендуется. Здесь главное — отличать запятые от точки с запятой. Все три параметра должны быть отделены друг от друга точкой с запятой.

Несколько операторов внутри каждого параметра должны быть отделены друг от друга запятыми. Первый и третий параметры в предыдущем примере содержат по два оператора:

```
$i = 1, $j = 1  // Инициализация переменных $i и $j
$i + $j < 10    // Условие окончания работы цикла
$i++ , $j++     // Модификация $i и $j в конце каждой итерации
```

Главное, что следует уяснить из этого примера, — три секции параметров должны разделяться точкой с запятой, а не запятыми (которые могут использоваться только для разделения операторов внутри каждой секции параметров).

Тогда при каких условиях следует отдавать предпочтение инструкциям `while` перед инструкциями `for`? Когда ваше условие не зависит от простого изменения переменной на постоянной основе. Например, инструкция `while` применяется в том случае, если нужно проверить, не введено ли какое-то определенное значение или не возникла ли какая-то конкретная ошибка, и завершить цикл сразу же, как только это произойдет.

Прекращение работы цикла

Прекратить работу цикла `for` (или любого другого цикла) можно точно так же, как и работу рассмотренной уже инструкции `switch`, — используя команду `break`. К примеру, это может понадобиться, когда одна из ваших инструкций вернет ошибку и продолжать выполнение цикла станет небезопасно.

Один из таких случаев может произойти, когда при записи файла возникнет ошибка, возможно, из-за нехватки места на диске (пример 4.35).

Пример 4.35. Запись файла, использующая цикл for с перехватом ошибки

```
<?php
  $fp = fopen("text.txt", 'wb');

  for ($j = 0 ; $j < 100 ; ++$j)
  {
    $written = fwrite($fp, "data");

    if ($written == FALSE) break;
  }
  fclose($fp);
?>
```

Это наиболее сложный из всех ранее приведенных фрагментов кода, но вы уже готовы к его пониманию. Команды обработки файлов будут рассмотрены в главе 7, а сейчас нужно лишь знать, что в первой строке кода открывается файл `text.txt` для записи в двоичном режиме, а затем переменной `$fp` возвращается указатель на него, который в дальнейшем используется для ссылки на этот открытый файл.

Затем осуществляется 100 проходов цикла (от 0 до 99), записывающих строку `data` в файл. После каждой записи функция `fwrite` присваивает переменной `$written` значение, представляющее собой количество успешно записанных символов. Но если происходит ошибка, то функция `fwrite` присваивает этой переменной значение `FALSE`.

Поведение функции `fwrite` облегчает коду проверку переменной `$written` на наличие значения `FALSE`, и если она имеет такое значение, код прекращает работу цикла и передает управление инструкции, закрывающей файл.

При желании улучшить код можно упростить строку:

```
if ($written == FALSE) break;
```

за счет использования оператора NOT:

```
if (!$written) break;
```

Фактически пара инструкций, находящихся внутри цикла, может быть сокращена до одной:

```
if (!fwrite($fp, "data")) break;
```

Иными словами, переменную $written можно исключить, поскольку она существует только для проверки значения, возвращенного из функции fwrite. Вместо этого можно протестировать возвращаемое значение напрямую.

Но команда break обладает более широкими возможностями, чем можно было бы предположить, поскольку, если нужно прекратить работу кода, вложенного глубже, чем на один уровень, после команды break можно поставить число, показывающее, работу скольких уровней нужно прекратить, например:

```
break 2;
```

Инструкция continue

Инструкция continue немного похожа на команду break, только она предписывает PHP остановить обработку текущей итерации цикла и перейти непосредственно к его следующей итерации, то есть вместо прекращения работы всего цикла PHP осуществляет выход только из текущей итерации.

Этот прием может пригодиться в тех случаях, когда известно, что нет смысла продолжать выполнение текущего цикла и нужно сберечь процессорное время или избежать ошибки путем перехода сразу к следующей итерации цикла. В примере 4.36 инструкция continue используется для того, чтобы избежать ошибки деления на нуль за счет ее вызова в тот момент, когда переменная $j имеет значение 0.

Пример 4.36. Перехват ошибки деления на нуль с помощью инструкции continue

```
<?php
  $j = 10;

   while ($j > -10)
  {
    $j--;

    if ($j == 0) continue;

    echo (10 / $j) . "<br>";
  }
>
```

Для всех значений переменной `$j` в диапазоне чисел между 10 и –10, за исключением 0, отображается результат деления числа 10 на значение переменной `$j`. Но для конкретного случая, когда значение `$j` равно `0`, вызывается инструкция `continue` и дальнейшее выполнение итерации сразу же пропускается с переходом к следующей итерации цикла.

Неявное и явное преобразование типов

PHP является языком со слабой типизацией, который позволяет объявлять переменную и ее тип путем простого использования этой переменной. При необходимости он также осуществляет автоматическое преобразование одного типа в другой. Этот процесс называется *неявным преобразованием* типов.

Однако могут возникнуть ситуации, когда присущее PHP неявное преобразование типов станет совсем нежелательным действием. Рассматривая пример 4.37, обратите внимание на то, что входные данные для операции деления являются целыми числами. По умолчанию PHP осуществляет преобразование выходных данных к числу с плавающей точкой, чтобы получалось наиболее точное значение — `4.66` и 6 в периоде.

Пример 4.37. Этот пример возвращает число с плавающей точкой

```
<?php
  $a = 56;
  $b = 12;
  $c = $a / $b;

  echo $c;
?>
```

Но что делать, если вместо этого нужно получить значение переменной `$c` в виде целого числа? Этого можно добиться разными способами, одним из которых является принудительное преобразование результата `$a / $b` в целое число путем использования оператора преобразования `(int)`:

```
$c = (int) ($a / $b);
```

Такой способ называется *явным преобразованием* типов. Обратите внимание, что для обеспечения преобразования в целое число значения всего выражения это выражение помещено в круглые скобки. В противном случае преобразованию подверглось бы только значение переменной `$a`, что не имело бы никакого смысла, поскольку деление на значение переменной `$b` все равно вернуло бы результат в виде числа с плавающей точкой.

Можно провести явное преобразование значений в те типы, которые показаны в табл. 4.6.

Необходимости использования преобразования можно избежать за счет вызова одной из встроенных функций PHP. Например, для получения целочисленного значения можно задействовать функцию intval. Этот раздел, как и многие другие в данной книге, предназначен в основном для того, чтобы помочь разобраться с чужим кодом, который может вам встретиться.

Таблица 4.6. Типы преобразований, доступных в PHP

Тип преобразования	Описание
(int) (integer)	Преобразование в целое число путем отбрасывания десятичной части
(bool) (boolean)	Преобразование в логическое значение
(float) (double) (real)	Преобразование в число с плавающей точкой
(string)	Преобразование в строку
(array)	Преобразование в массив
(object)	Преобразование в объект

Динамическое связывание в PHP

Поскольку PHP является языком программирования и получаемая в результате его работы выходная информация может быть совершенно разной для различных пользователей, есть возможность запускать целый сайт из одной веб-страницы, созданной с помощью PHP. При каждом щелчке пользователя на каком-нибудь элементе подробности могут отправляться назад той же веб-странице, которая будет принимать решение, что делать дальше, в соответствии с различными объектами cookie и/или данными сессии, которые могут быть сохранены.

Но несмотря на возможность создания таким способом целого сайта, этого делать не рекомендуется, поскольку исходный код будет все время разрастаться и приобретет громадные размеры по мере того, как ему придется принимать во внимание разнообразные действия пользователя.

Будет куда благоразумнее разделить разработку сайта на несколько разных частей. Например, один автономный процесс будет заниматься подпиской на сайт со всеми вытекающими отсюда проверками допустимости адреса электронной почты, незадействованности имени пользователя и т. д.

Второй модуль неплохо было бы создать для регистрации пользователей, предшествующей их допуску к основной части вашего сайта. Затем можно создать модуль вывода сообщений, в котором пользователи могли бы оставлять свои комментарии, модуль, содержащий ссылки и полезную информацию, еще один модуль, позволяющий загружать на сайт фотографии, и т. д.

Как только будет создано средство для отслеживания действий пользователя на вашем сайте, использующее объекты cookie или переменные сессии (оба этих средства будут более подробно рассмотрены в следующих главах), можно разделить сайт на удобные секции PHP-кода, каждая из которых будет независима от других. Таким образом, вы существенно облегчите себе будущую разработку каждого нового свойства и обслуживание уже имеющихся.

Динамическое связывание в действии

Одним из наиболее популярных в настоящее время приложений, управляемых PHP, является платформа для ведения блогов WordPress (рис. 4.5). При ведении или чтении блога этого можно и не понять, но для каждой основной секции выделен свой основной PHP-файл, а огромное количество совместно используемых функций помещено в отдельные файлы, которые включаются основными PHP-страницами по мере необходимости.

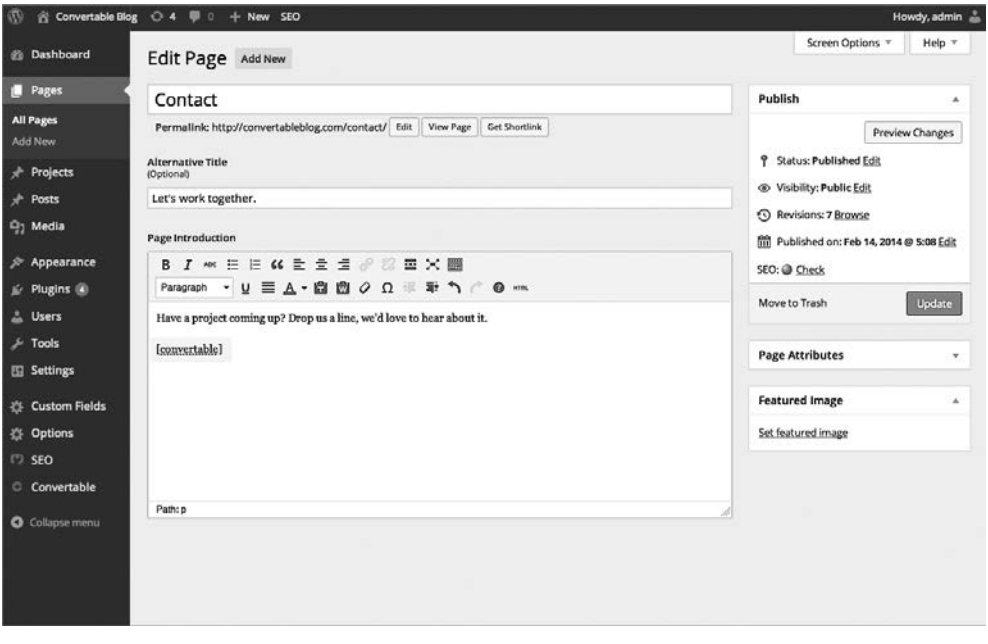

Рис. 4.5. Инструментальная панель платформы WordPress, предназначенной для ведения блогов

Вся платформа держится на закулисном отслеживании сессии, поэтому вы вряд ли знаете о том, когда осуществляется переход от одной подчиненной секции к другой. В итоге, если веб-разработчик хочет провести тонкую настройку WordPress, ему не трудно найти конкретный файл, который для этого применяется, и выполнить его проверку и отладку, не теряя понапрасну времени на не связанные с ним части

программы. Когда в очередной раз будете использовать WordPress, проследите за адресной строкой своего браузера, особенно при управлении блогом, и тогда вы сможете заметить обращения к разнообразным PHP-файлам, которые используются в этом приложении.

В текущей главе были рассмотрены обширные сведения, закладывающие основу для дальнейшего изучения материала книги. Теперь вы уже должны уметь составлять свои собственные небольшие PHP-программы. Но, перед тем как перейти к следующей главе, посвященной функциям и объектам, можете проверить приобретенные знания, ответив на контрольные вопросы.

Вопросы

1. Какие основные значения представлены ключевыми словами `TRUE` и `FALSE`?
2. Что представляют собой две самые простые формы выражений?
3. В чем разница между унарными, бинарными и трехкомпонентными операторами?
4. В чем заключается наилучший способ установки собственной приоритетности операторов?
5. Что означает понятие *взаимосвязанности операторов*?
6. Когда следует использовать оператор тождественности (`===`)?
7. Назовите три типа условных инструкций.
8. Какую команду можно использовать для пропуска текущей итерации цикла и перехода к следующей итерации?
9. Почему цикл `for` считается более мощным, чем `while`?
10. Как инструкции `if` и `while` интерпретируют условные выражения, составленные из разных типов данных?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

5 Функции и объекты PHP

К основным требованиям к любому языку программирования относится наличие места для хранения данных, средств для направления процесса выполнения программы и других мелочей, таких как вычисление выражений, управление файлами и вывод текста. PHP обладает всем этим, и вдобавок у него имеется облегчающий жизнь инструментарий наподобие инструкций `else` и `elseif`. Но даже если все это входит в наш набор инструментов, программирование может быть слишком нудным и утомительным занятием, особенно если регулярно будет возникать необходимость вновь и вновь набирать очень похожие друг на друга фрагменты кода.

И тут нам на помощь приходят функции и объекты. Нетрудно догадаться, что *функция* — это набор инструкций, который выполняет конкретную задачу и в дополнение к этому может вернуть какое-нибудь значение. Можно извлечь фрагмент кода, который используется более одного раза, поместить его в функцию и вызвать функцию по имени в тот момент, когда этот код нужно будет выполнить.

По сравнению с непрерывным линейным кодом у функций есть масса преимуществ.

- Экономия времени при наборе текста программы.
- Сокращение количества синтаксических и прочих ошибок программирования.
- Сокращение времени загрузки файлов программы.
- Сокращение времени выполнения, поскольку каждая функция компилируется только один раз, независимо от частоты ее вызовов.
- Возможность использовать функции как в рядовых, так и в особенных случаях, поскольку они воспринимают аргументы.

Объекты являются дальнейшим развитием этой концепции. *Объект* объединяет одну или несколько функций и данные, которые ими используются, в единую структуру, которая называется *классом*.

В этой главе будет рассмотрено все, что касается использования функций, — от их определения и вызова до различных способов передачи данных. Вооружившись

этими знаниями, вы сможете создавать функции и использовать их в собственных объектах (в которых они будут упоминаться как *методы*).

Временами еще встречаются случаи использования любой версии PHP ниже 5.4, что, несомненно, не рекомендуется делать. Поэтому при изучении данной главы предполагается, что выпуск с этим номером является той самой младшей версией, с которой вы будете работать. Вообще-то, я рекомендую пользоваться версией 5.6 или новой версией 7.0 или 7.1 (выпуск версии 6 не состоялся). Любую из этих версий можно выбрать на панели управления AMPPS, воспользовавшись рекомендациями из главы 2.

Функции PHP

PHP поставляется с несколькими сотнями готовых к работе встроенных функций, превращающих его в язык с очень богатыми возможностями. Чтобы воспользоваться функцией, ее нужно вызвать по имени. Посмотрим, например, как работает функция `date`:

```
echo date("l"); // Показывает день недели
```

Круглые скобки сообщают PHP, что вы ссылаетесь на функцию. В противном случае будет считаться, что вы ссылаетесь на константу.

Функции могут принимать любое количество аргументов, включая нулевое. Например, показанная ниже функция `phpinfo` отображает множество информации о текущей установке PHP и не требует никаких аргументов:

```
phpinfo ();
```

Результат вызова этой функции показан на рис. 5.1.

Функция phpinfo весьма полезна для получения информации о текущей установке PHP, но этой информацией могут воспользоваться и потенциальные злоумышленники. Поэтому никогда не оставляйте вызов этой функции в коде, подготовленном для работы в сети.

В примере 5.1 показано несколько встроенных функций, использующих один аргумент и более.

Пример 5.1. Три функции для работы со строками

```
<?php
  echo strrev(" .dlrow olleH"); // Реверсирование строки
  echo str_repeat("Hip ", 2);   // Повторение строки
  echo strtoupper("hooray!");   // Преобразование символов строки в верхний регистр
```

Рис. 5.1. Информация, выводимая встроенной в PHP функцией phpinfo

В этом примере используются три функции для обработки строк, выводящие следующий текст:

```
Hello world. Hip Hip HOORAY!
```

Как следует из результата, функция `strrev` реверсирует порядок символов в строке, функция `str_repeat` дважды повторяет строку `"Hip"` (в соответствии с требованием второго аргумента), а функция `strtoupper` переводит буквы в слове `"hooray!"` в верхний регистр.

Определение функции

В общем виде для функции используется следующий синтаксис:

```
function имя_функции([параметр [, ...]])
{
  // Инструкции
}
```

В первой строке синтаксиса показано следующее:

- определение начинается со слова `function`;
- за ним идет имя, которое должно начинаться с буквы или символа подчеркивания;
- за ними может следовать любое количество букв, цифр или знаков подчеркивания;

- наличие круглых скобок обязательно;
- к необязательному элементу (обозначаемому тем, что он заключен в квадратные скобки) относится один или несколько параметров, разделенных запятыми.

Имена функций нечувствительны к регистру использующихся в них букв, поэтому все следующие строки могут ссылаться на одну и ту же функцию `print`: `PRINT`, `Print` и `PrInT`.

С открывающей фигурной скобки начинаются инструкции, которые будут выполнены при вызове функции; они должны завершаться закрывающей фигурной скобкой, составляющей пару первой скобке. В составе этих инструкций должны быть одна или несколько инструкций `return`, заставляющих функцию прекратить выполнение и вернуть управление вызывавшему функцию коду. Если инструкция `return` продолжена каким-нибудь значением, то вызывающий код может его извлечь, что мы сейчас и увидим.

Возвращение значения

Рассмотрим простую функцию, преобразующую буквы чьих-нибудь полных имен в нижний регистр, а затем переводящую в верхний регистр первую букву каждой из составных частей имени.

В примере 5.1 нам уже встречалась встроенная PHP-функция `strtoupper`. Для нашей текущей функции будет использована ее противоположность: функция `strtolower`:

```
$lowered = strtolower("люБОЕ нУжное Вам количество Букв и Знаков Пунктуации");

echo $lowered;
```

На выходе этого эксперимента получается следующая строка:

```
любое нужное вам количество букв и знаков пунктуации
```

Но нам не нужны имена, полностью состоящие из букв нижнего регистра, мы хотим превратить первые буквы в прописные. (Не будем в этом примере брать в расчет такие редкие имена, как Mary-Ann или Jo-En-Lai.) Нам и здесь сопутствует удача: PHP предоставляет также функцию `ucfirst`, которая переводит первую букву строки в верхний регистр:

```
$ucfixed = ucfirst("любое нужное вам количество букв и знаков пунктуации");

echo $ucfixed;
```

На выходе получается следующая строка:

```
Любое нужное вам количество букв и знаков пунктуации
```

Теперь мы можем внести свою лепту в конструирование программы: чтобы получить слово с первой прописной буквой, сначала для строки будет вызвана функция `strtolower`, а затем функция `ucfirst`. Для этого вызов функции `strtolower` будет

вложен в вызов функции `ucfirst`. Посмотрим, зачем это делается, потому что нам важно понять порядок вычисления кода.

Если воспользоваться следующим простым вызовом функции `print`:

```
print(5-8);
```

то сначала будет вычислено выражение `5 – 8` и на выходе будет получено число `–3`. (В предыдущей главе уже было показано, что PHP для отображения этого результата превращает его в строку.) Если выражение содержит функцию, то сначала вычисляется эта функция:

```
print(abs(5-8));
```

Для выполнения этой короткой инструкции PHP совершает следующие действия.

1. Вычисляет `5 – 8`, выдавая результат `–3`.
2. Использует функцию `abs`, превращая `–3` в `3`.
3. Превращает результат в строку и выводит его, используя функцию `print`.

Такой порядок работы обусловлен тем, что PHP вычисляет каждый элемент, начиная с самого внутреннего и заканчивая тем, который находится снаружи. То же самое происходит при обработке следующего вызова:

```
ucfirst(strtolower("люБОЕ нУжное Вам количество Букв и Знаков Пунктуации"))
```

PHP передает нашу строку функции `strtolower`, а затем функции `ucfirst`, выдавая следующий результат (который мы уже видели, когда вызывали функции отдельно друг от друга):

```
Любое нужное вам количество букв и знаков пунктуации
```

Теперь определим функцию (показанную в примере 5.2), которая берет три имени и переводит их буквы в нижний регистр, после чего превращает первую букву в прописную.

Пример 5.2. Приведение в порядок полного имени

```
<?php
  echo fix_names("WILLIAM", "henry", "gatES");

  function fix_names($n1, $n2, $n3)
  {
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return $n1 . " " . $n2 . " " . $n3;
  }
?>
```

Пользователи часто забывают вовремя выключить режим Caps Lock, случайно ставят прописные буквы не там, где нужно, и даже вообще забывают о них, от чего вы тоже не застрахованы. В результате выполнения кода этого примера будет выведен следующий текст:

```
William Henry Gates
```

Возвращение массива

Выше была рассмотрена функция, возвращающая единственное значение. Но существуют также способы получения при выполнении функции сразу нескольких значений.

Самый подходящий из них возвращает эти значения в виде массива. В главе 3 уже было показано, что массив похож на связку переменных в одной строке. Использование массива для возвращения значений функции отображено в примере 5.3.

Пример 5.3. Возвращение нескольких значений в массиве

```
<?php
  $names = fix_names("WILLIAM", "henry", "gatES");
  echo $names[0] . " " . $names[1] . " " . $names[2];

  function fix_names($n1, $n2, $n3)
  {
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return array($n1, $n2, $n3);
  }
?>
```

У этого метода есть преимущество, заключающееся в том, что все три имени содержатся по отдельности, а не объединяются в одну строку, что дает возможность обращаться к любому пользователю просто по его имени или фамилии, не извлекая каждое имя из возвращаемой строки.

Передача аргументов по ссылке

В версиях PHP, предшествующих версии 5.3, вы привыкли пользоваться возможностью употребления при вызове функции символа `&` (например, `increment(&$myvar);`), чтобы заставить парсер передавать ссылку на переменную, а не значение самой переменной. Тем самым функции предоставлялся доступ к переменной (позволяющий записывать в нее различные значения).

В версии PHP 5.3 передача по ссылке при вызове функции попала в число нерекомендуемых приемов, а из версии PHP 5.4.0 возможность такой передачи была удалена. Поэтому вам не следует пользоваться этим приемом нигде, кроме как на устаревших сайтах, и даже при этом рекомендуется переписать код, передающий значения по ссылке, поскольку в новых версиях PHP он будет приводить к остановке программы с выдачей неустранимой ошибки.

Но *внутри* определения функции можно продолжать обращаться к аргументам по ссылке. Этот подход может представлять для вас определенные сложности, поэтому вернемся к метафоре со спичечным коробком, которая использовалась в главе 3.

Представьте, что вы не вынимаете клочок бумаги из коробка, не читаете то, что на нем написано, не копируете эту надпись на другой клочок бумаги, не возвращаете оригинал в коробок и не передаете копию функции, а просто привязываете нитку к исходному клочку бумаги и передаете функции второй конец этой нитки (рис. 5.2).

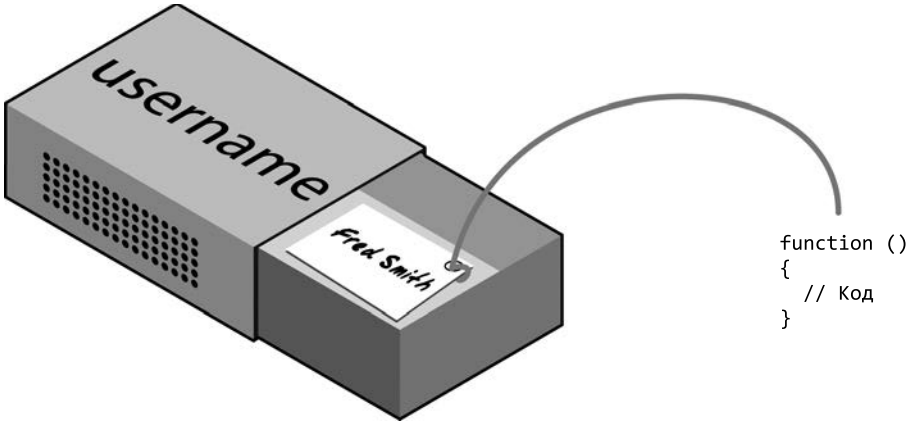

Рис. 5.2. Представление ссылки в виде нитки, привязанной к значению переменной

Теперь, чтобы найти данные, к которым она обращается, функция может проследовать по нитке. Таким образом исключаются все издержки на создание копии переменной, предназначенной только для того, чтобы в функции можно было воспользоваться ее значением. Более того, теперь функция может изменить значение переменной.

Значит, пример 5.3 можно переписать: передать ссылки на все параметры, чтобы после этого функция напрямую смогла внести в них изменения (пример 5.4).

Пример 5.4. Передача значений в функцию по ссылке

```
<?php
  $a1 = "WILLIAM";
  $a2 = "henry";
  $a3 = "gatES";
```

```
  echo $a1 . " " . $a2 . " " . $a3 . "<br>";
  fix_names($a1, $a2, $a3);
  echo $a1 . " " . $a2 . " " . $a3;

  function fix_names(&$n1, &$n2, &$n3)
  {
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));
  }
?>
```

Вместо передачи строк непосредственно функции они сначала присваиваются в качестве значений переменным и выводятся на экран, чтобы можно было посмотреть их состояние «до». Затем, как и раньше, вызывается функция, но внутри определения функции перед именем каждого параметра, передаваемого по ссылке, ставится символ `&`.

Теперь к переменным `$n1`, `$n2` и `$n3` привязаны «ниточки», ведущие к значениям переменных `$a1`, `$a2` и `$a3`. Иными словами, существует одна группа значений, но два набора имен переменных, позволяющих к ним обратиться.

Поэтому функции `fix_names` нужно только присвоить новые значения переменным `$n1`, `$n2` и `$n3`, чтобы обновить значения переменных `$a1`, `$a2` и `$a3`. В результате выполнения этого кода будут выведены следующие строки:

```
WILLIAM henry gatES
William Henry Gates
```

Как видите, в обеих инструкциях `echo` используются только значения переменных `$a1`, `$a2` и `$a3`.

Возвращение глобальных переменных

Лучшим способом предоставления функции доступа к переменной, созданной за ее пределами, является объявление ее глобальной прямо из тела функции. За ключевым словом `global` должно следовать имя переменной, тогда полный доступ к этой переменной можно будет получить из любой части вашего кода (пример 5.5).

Пример 5.5. Возвращение значений в глобальных переменных

```
<?php
  $a1 = "WILLIAM";
  $a2 = "henry";
  $a3 = "gatES";

  echo $a1 . " " . $a2 . " " . $a3 . "<br>";
  fix_names();
  echo $a1 . " " . $a2 . " " . $a3;

  function fix_names()
  {
```

```
    global $a1; $a1 = ucfirst(strtolower($a1));
    global $a2; $a2 = ucfirst(strtolower($a2));
    global $a3; $a3 = ucfirst(strtolower($a3));
  }
?>
```

Теперь уже не нужно передавать функции параметры и она не должна их принимать. После объявления эти переменные остаются глобальными и доступными коду всей остальной программы, включая ее функции.

И еще раз об области видимости переменных

Кратко напомню те сведения, которые были получены при изучении главы 3.

- *Локальные переменные* доступны лишь из той части кода, в которой они были определены. Если это произошло за пределами функции, доступ к переменным будет возможен из всего кода, находящегося за пределами функций, классов и т. д. Если переменная была определена внутри функции, значит доступ к ней может получить только код этой функции и ее значение теряется при выходе из функции.
- *Глобальные переменные* доступны из любых частей вашего кода.
- *Статические переменные* доступны только внутри функции, в которой они были объявлены, но при этом они сохраняют свое значение в процессе многократных вызовов функции.

Включение и запрос файлов

По мере приобретения навыков программирования на PHP вы, скорее всего, приступите к созданию библиотеки, состоящей из функций, которые, по вашему мнению, смогут пригодиться в будущем. Кроме того, наверное, вы начнете пользоваться библиотеками, созданными другими программистами.

Копировать эти функции и вставлять их в свой код не имеет никакого смысла. Можно сохранить эти функции в отдельных файлах и воспользоваться командами для их извлечения. Для этого существуют две команды: `include` (включить) и `require` (затребовать).

Инструкция include

При использовании инструкции `include` можно потребовать у PHP извлечения конкретного файла и загрузки всего его содержимого. Это равносильно вставке включаемого файла в данное место текущего файла. В примере 5.6 показано, как нужно включать файл под названием `library.php`.

Пример 5.6. Включение файла PHP

```
<?php
  include "library.php";

  // Сюда помещается ваш код
?>
```

Инструкция include_once

При каждом использовании директивы `include` она снова вставляет требуемый файл, даже если он уже был вставлен. Предположим, к примеру, что в библиотеке `library.php` содержится масса полезных функций. Вы включаете ее в свой файл, но, кроме нее, включаете еще одну библиотеку, которая содержит `library.php`. Из-за этой вложенности вы непреднамеренно вставляете `library.php` дважды. В результате будут появляться сообщения об ошибках, потому что будет предпринята попытка несколько раз объявить одну и ту же константу или функцию. Поэтому вместо данной директивы нужно использовать инструкцию `include_once` (пример 5.7).

Пример 5.7. Однократное включение файла PHP

```
<?php
  include_once "library.php";

  // Сюда помещается ваш код
?>
```

Теперь любые дальнейшие попытки включения того же самого файла (с помощью `include` или `include_once`) будут проигнорированы. Чтобы определить, был ли запрошенный файл уже включен, абсолютный путь к нему сравнивается со всеми раскрытыми относительными путями и файлом, найденным в пути, который указан в вашей инструкции `include`.

Лучше будет придерживаться использования инструкции include_once и не применять инструкцию include. Тогда у вас никогда не будет проблем с тем, что файлы вставляются по нескольку раз.

Инструкции require и require_once

Потенциальная проблема, возникающая при использовании инструкций `include` и `include_once`, состоит в том, что для вставки нужного файла PHP предпримет всего одну попытку. Выполнение программы продолжится даже в том случае, если файл не будет найден.

Когда вставка файла имеет принципиальную важность, его нужно затребовать, то есть применить инструкцию `require`. По тем же причинам, которые излагались при рассмотрении использования инструкции `include_once`, я рекомендую, чтобы вы, когда нужно затребовать файл, придерживались главным образом использования инструкции `require_once` (пример 5.8).

Пример 5.8. Однократное востребование файла PHP

```
<?php
  require_once "library.php";

  // Сюда помещается ваш код
?>
```

Совместимость версий PHP

PHP продолжает совершенствоваться и существует в нескольких версиях. Если нужно проверить доступность в вашем коде какой-нибудь конкретной функции, можно воспользоваться функцией `function_exists`, которая проверяет все предопределенные и созданные пользователем функции.

В примере 5.9 проверяется доступность функции `array_combine`, которая имеется в PHP версии 5.

Пример 5.9. Проверка существования функции

```
<?php
  if (function_exists("array_combine"))
  {
    echo "Функция существует";
  }
  else
  {
    echo "Функция не существует, желательно создать ее самостоятельно";
  }
?>
```

Добавив подобный код, можно воспользоваться любыми функциональными возможностями, имеющимися в новых версиях PHP, которые вам придется смоделировать, если нужно будет, чтобы ваш код работал и в более ранних версиях. Ваши функции могут работать медленнее встроенных, но код по крайней мере будет обладать более широкой переносимостью.

Чтобы определить версию PHP, под которой запущен ваш код, можно также воспользоваться функцией `phpversion`. Возвращаемый результат в зависимости от версии будет иметь следующий вид:

```
5.5.38
```

Объекты PHP

Практически так же, как применение функций стало фактором существенного увеличения эффективности программирования на заре развития вычислительной техники (когда лучшим из доступных средств программной навигации порой была самая элементарная инструкция `GOTO` или `GOSUB`), *объектно-ориентированное программирование (ООП)* подняло использование функций на совершенно новый уровень.

Как только у вас появится навык сведения повторно используемых фрагментов кода в функции, останется сделать еще один небольшой шаг и присмотреться к связыванию функций и данных, которыми они оперируют, в объекты.

Рассмотрим сайт социальной сети, состоящий из множества различных частей. Одна из таких частей управляет всеми пользовательскими функциями: ее код позволяет новым пользователям зарегистрироваться, а уже зарегистрированным — изменить свои личные данные. В стандартном PHP можно создать для управления всеми этими действиями ряд функций и встроить несколько вызовов к базе данных MySQL, чтобы ввести данные по всем пользователям.

А теперь вообразите, насколько проще будет создать объект, представляющий текущего пользователя. Для этого можно создать класс по имени `User`, в котором будут содержаться весь код, необходимый для обслуживания пользователей, и все переменные, требующиеся для работы с данными внутри класса. Затем, когда понадобится управлять пользовательскими данными, можно будет просто создать новый объект класса `User`.

Этот новый объект можно будет рассматривать в качестве настоящего пользователя. Например, объекту можно передать имя, пароль и адрес электронной почты, спросить его о том, существует ли уже такой пользователь, и если нет, заставить его создать нового пользователя с данными атрибутами. Можно даже иметь объект мгновенных сообщений или объект, позволяющий учитывать дружеские отношения между двумя пользователями.

Терминология

При создании программы, рассчитанной на использование объектов, нужно сконструировать некую совокупность данных и кода, называемую классом. Каждый новый объект, основанный на этом классе, называется *экземпляром* (или *случаем употребления*) этого класса.

Данные, связанные с объектом, называются его *свойствами*, а используемые им функции — *методами*. При определении класса задаются имена его свойств и код для его методов. На рис. 5.3 приведен метафорический пример объекта в виде музыкального автомата. Компакт-диски в его карусели можно рассматривать

в качестве его свойств, а метод их проигрывания заключается в нажатии кнопки на передней панели. Есть также щель для опускания монет (метод, используемый для активизации объекта) и устройство чтения компакт-дисков (метод, используемый для извлечения музыки, или свойств, с компакт-дисков).

Рис. 5.3. Музыкальный автомат как подходящий пример автономного объекта

При создании объектов предпочтительно воспользоваться инкапсуляцией или создавать класс таким образом, чтобы с его свойствами могли работать только его собственные методы. Иными словами, нужно запретить внешнему коду непосредственный доступ к данным объекта. Предоставляемые объектом методы известны как *интерфейс* объекта.

Такой подход упрощает отладку: дефектный код придется исправлять только в пределах класса.

Кроме того, когда нужно будет обновить программу, при использовании надлежащей инкапсуляции и поддержке одинакового интерфейса можно будет просто разработать новые классы для замены старых, полностью их отладить, а затем заменить ими старые классы. Если они будут в чем-то неработоспособными, можно будет вернуть назад старые классы для немедленного устранения проблемы перед дальнейшей отладкой новых классов.

Как только класс будет создан, может выясниться, что нужен еще один, похожий на него, но все же несколько отличающийся класс. Быстрее и проще всего будет определить новый класс, воспользовавшись *наследованием*. При этом ваш новый класс сохранит все свойства, присущие тому классу, чьим наследником он является.

Исходный класс теперь будет называться *суперклассом*, а новый класс — *подклассом* (или *производным* классом).

Вернемся к примеру с музыкальным автоматом. Если вы изобретаете новый музыкальный автомат, который наряду с музыкой может воспроизводить и видеоклипы, то можете сохранить все свойства и методы исходного музыкального автомата и добавить несколько новых свойств (видеоклипов) и новых методов (видеоплееров).

Существенным преимуществом этой системы является то, что, если вы увеличили скорость работы или улучшили другие аспекты работы суперкласса, его подклассы пользуются теми же самыми усовершенствованиями.

Объявление класса

Перед тем как получить возможность использования объекта, нужно определить класс с помощью ключевого слова `class`. Определение класса включает в себя имя класса (чувствительное к регистру букв), его свойства и методы. В примере 5.10 дается определение класса `User`, имеющего два свойства: `$name` и `$password` (которые обозначены ключевым словом `public` — см. подраздел «Область видимости свойств и методов» данного раздела). В нем также создается новый экземпляр этого класса (по имени `$object`).

Пример 5.10. Объявление класса и проверка объекта

```
<?php
  $object = new User;
  print_r($object);

  class User {
    public $name, $password;

    function save_user()
    {
      echo "Сюда помещается код, сохраняющий данные пользователя";
    }
  }
?>
```

Здесь также задействована поистине бесценная функция под названием `print_r`. Она требует от PHP отобразить информацию о переменной в удобной для восприятия человеком форме, о чем говорит элемент `_r` в ее имени (означающий readable — «читаемый»). Для нового объекта `$object` эта функция выводит следующую информацию:

```
User Object
(
  [name]      =>
  [password]  =>
)
```

Но браузер сжимает все пустые пространства, поэтому выводимая в нем информация читается немного сложнее:

```
User Object ( [name] => [password] => )
```

В любом случае выведенная информация свидетельствует о том, что `$object` является объектом, определенным пользователем, и содержит свойства `name` и `password`.

Создание объекта

Для создания объекта определенного класса используется ключевое слово `new`, применяемое в выражении: *`$объект`* = `new` *`Класс`*. Вот два способа создания объектов:

```
$object = new User;
$temp   = new User('name', 'password');
```

В первой строке мы просто назначаем объект классу `User`. А во второй строке передаем вызову параметры.

Класс может требовать или запрещать аргументы; он также может разрешать аргументы без их явного востребования.

Доступ к объектам

Добавим к примеру 5.10 еще несколько строк и проверим результаты. В примере 5.11 предыдущий код расширяется за счет установки свойств объекта и вызова метода.

Пример 5.11. Создание объекта и взаимодействие с ним

```
<?php
  $object = new User;
  print_r($object); echo "<br>";

  $object->name = "Joe";
  $object->password = "mypass";
  print_r($object); echo "<br>";

  $object->save_user();

  class User {
    public $name, $password;

    function save_user()
    {
      echo "Сюда помещается код, сохраняющий данные пользователя";
    }
  }
?>
```

Из примера видно, что для доступа к свойству объекта используется следующий синтаксис: *`$объект->свойство`*. Похожим образом можно вызвать и метод: *`$объект->метод()`*.

Можно было заметить, что перед именами свойств и методов отсутствуют символы доллара (`$`). Если на первой позиции имен поставить символ `$`, то код не будет работать, поскольку он попробует обратиться к значению, хранящемуся в переменной. Например, выражение `$object->$property` будет пытаться найти значение, присвоенное переменной по имени `$property` (скажем, это значение является строкой `brown`), а затем обратиться к свойству `$object->brown`. Если переменная `$property` не определена, то будет предпринята попытка обратиться к свойству `$object->NULL`, что спровоцирует возникновение ошибки.

Если организовать просмотр, используя имеющееся в браузере средство для просмотра исходного кода, то код примера 5.11 выведет следующую информацию:

```
User Object
(
  [name]     =>
  [password] =>
)
User Object
(
  [name]     => Joe
  [password] => mypass
)
Сюда помещается код, сохраняющий данные пользователя
```

Здесь также используется функция `print_r`, которая предоставляет содержимое переменной `$object` до и после присваивания свойству значения. В дальнейшем я не стану использовать инструкцию `print_r`, но, если материал этой книги будет прорабатываться на вашем разработочном сервере, вы сможете поместить в код несколько таких инструкций, чтобы иметь полное представление о том, что происходит.

Можно также было заметить, что благодаря вызову метода `save_user` был выполнен код этого метода, который вывел строку, напоминающую о происходящем.

Определения функций и классов можно помещать в любое место вашего кода, до или после инструкций, в которых они используются. Но правилом хорошего тона считается помещать их ближе к концу файла.

Клонирование объектов

Если объект уже создан, то в качестве параметра он передается по ссылке. Если вернуться к примеру со спичечным коробком, то это похоже на привязывание сразу нескольких ниток к объекту, хранящемуся в коробке, что позволяет получить к нему доступ, следуя по любой из привязанных ниток.

Иными словами, присваивание объектов не приводит к их полному копированию.

Как это работает, показано в примере 5.12, где определяется очень простой пользовательский класс `User`, который не имеет методов и содержит всего лишь одно свойство `name`.

Пример 5.12. Копирование объекта

```
<?php
  $object1       = new User();
  $object1->name = "Alice";
  $object2       = $object1;
  $object2->name = "Amy";

  echo "object1 name = " . $object1->name . "<br>";
  echo "object2 name = " . $object2->name;

  class User
  {
    public $name;
  }
?>
```

Мы создали объект `$object1` и присвоили свойству `name` значение `Alice`. Затем создали `$object2`, присвоили ему значение `$object1` и присвоили значение `Amy` непосредственно свойству `name` объекта `$object2` — или подумали, что присвоили. Но этот код выдаст следующую информацию:

```
object1 name = Amy
object2 name = Amy
```

Что же произошло? И `$object1`, и `$object2` ссылаются на *один и тот же* объект, поэтому изменение свойства `name`, принадлежащего `$object2`, на `Amy` устанавливает такое же значение и для свойства, принадлежащего `$object1`.

Во избежание подобной путаницы следует использовать инструкцию `clone`, которая создает новый экземпляр класса и копирует значения свойств из исходного класса в новый экземпляр. Применение этой инструкции показано в примере 5.13.

Пример 5.13. Клонирование объекта

```
<?php
  $object1       = new User();
  $object1->name = "Alice";
  $object2       = clone $object1;
  $object2->name = "Amy";

  echo "object1 name = " . $object1->name . "<br>";
  echo "object2 name = " . $object2->name;

  class User
  {
    public $name;
  }
?>
```

Вот и все. Этот код выдает то, что нам требовалось получить с самого начала:

```
object1 name = Alice
object2 name = Amy
```

Конструкторы

При создании нового объекта вызываемому классу можно передать перечень аргументов. Они передаются специальному методу внутри класса, который называется *конструктором* и занимается инициализацией различных свойств.

Для этого используется функция по имени `__construct` (то есть перед `construct` ставятся два символа подчеркивания), как в примере 5.14.

Пример 5.14. Создание метода-конструктора

```
<?php
  class User
  {
    function __construct($param1, $param2)
    {
      // Сюда помещаются инструкции конструктора
      public $username = "Guest";
    }
  }
?>
```

Деструкторы

Имеется также возможность создания методов-*деструкторов*. Она подходит для тех случаев, когда код ссылается на объект в последний раз или когда сценарий подошел к концу. В примере 5.15 показано, как создается метод-деструктор. Деструктор может выполнять очистку, например сбросить подключение к базе данных или к каким-нибудь другим ресурсам, зарезервированным в классе. Поскольку ресурсы резервируются внутри класса, их высвобождение должно происходить здесь же, или они будут удерживаться до бесконечности. Когда программа резервирует ресурсы и забывает их высвободить, возникают проблемы общесистемного характера.

Пример 5.15. Создание метода-деструктора

```
<?php
  class User
  {
    function __destruct()
    {
      // Сюда помещается код деструктора
    }
  }
?>
```

Написание методов

Как видите, объявление метода похоже на объявление функции, но есть некоторые отличия. Например, имена методов, начинающиеся с двойного подчеркивания (__), являются зарезервированными словами, и вы не должны больше создавать ничего подобного.

Кроме того, существует специальная переменная `$this`, которая может использоваться для доступа к свойствам текущего объекта. Чтобы понять, как это работает, посмотрите на код примера 5.16, содержащий еще один метод из определения класса `User`, который называется `get_password`.

Пример 5.16. Использование в методе переменной $this

```
<?php
  class User
  {
    public $name, $password;

    function get_password()
    {
      return $this->password;
    }
  }
?>
```

Метод получения пароля — `get_password` — применяет переменную `$this` для доступа к текущему объекту, а затем возвращает значение свойства `password`, принадлежащего этому объекту. Обратите внимание на то, как при использовании оператора `->` в имени свойства `$password` опускается первый символ `$`. Если оставить его на прежнем месте, особенно при первом применении этого свойства, будет допущена весьма типичная ошибка.

Класс, определенный в примере 5.16, нужно использовать следующим образом:

```
$object           = new User;
$object->password = "secret";

echo $object->get_password();
```

Этот код выводит пароль `secret`.

Объявление свойств

В явном объявлении свойств внутри классов нет необходимости, поскольку они могут быть определены неявным образом при первом же их использовании. Для иллюстрации этой особенности класс `User` в примере 5.17 не имеет ни свойств, ни методов, но при этом в коде его определения нет ничего противозаконного.

Пример 5.17. Неявное объявление свойства

```
<?php
  $object1       = new User();
  $object1->name = "Alice";

  echo $object1->name;

  class User {}
?>
```

Этот код вполне корректно и без проблем выведет строку `Alice`, поскольку PHP неявным образом объявит для вас свойство `$object1->name`. Но такой стиль программирования может привести к ошибкам, найти которые будет невероятно трудно, поскольку свойство `name` было объявлено за пределами класса.

Чтобы не создавать трудностей ни себе, ни тому, кто впоследствии будет обслуживать ваш код, я советую выработать привычку всегда объявлять свойства внутри класса в явном виде. И поверьте, вы об этом никогда не пожалеете.

К тому же, когда свойство объявляется внутри класса, ему можно присвоить значение по умолчанию. Используемое вами значение должно быть константой, а не результатом вызова функции или вычисления выражения. Несколько допустимых и недопустимых присваиваний показано в примере 5.18.

Пример 5.18. Допустимые и недопустимые объявления свойств

```
<?php
  class Test
  {
    public $name  = "Paul Smith";  // Допустимое
    public $age   = 42;            // Допустимое
    public $time  = time();        // Недопустимое — вызывает функцию
    public $score = $level * 2;    // Недопустимое — использует выражение
  }
?>
```

Объявление констант

По аналогии с созданием глобальных констант внутри определения функций можно определять константы и внутри классов. Чтобы константы выделялись на общем фоне, обычно для их имен используют буквы верхнего регистра (пример 5.19).

Пример 5.19. Определение констант внутри класса

```
<?php
  Translate::lookup();

  class Translate
  {
    const ENGLISH = 0;
```

```
    const SPANISH = 1;
    const FRENCH = 2;
    const GERMAN = 3;
    // ...

    static function lookup()
    {
      echo self::SPANISH;
    }
  }
?>
```

К константам можно обращаться напрямую, с помощью ключевого слова `self` и оператора двойного двоеточия. Обратите внимание на то, что этот код, в первой строке которого используется оператор двойного двоеточия, вызывает класс напрямую, без предварительного создания его экземпляра. Как и ожидалось, значение, выводимое при запуске этого кода на выполнение, будет равно `1`.

Запомните, что константа после ее определения не может быть изменена.

Область видимости свойств и методов

PHP предоставляет три ключевых слова для управления областью видимости свойств и методов *(элементов)*.

- `public` (открытые). На открытые элементы можно ссылаться откуда угодно, включая другие классы и экземпляры объекта. Свойства с этой областью видимости получаются по умолчанию при объявлении переменной с помощью ключевых слов `var` или `public` или когда переменная объявляется неявно при первом же ее использовании.

  Ключевые слова `var` и `public` являются взаимозаменяемыми. Хотя сейчас использование `var` не приветствуется, оно сохранено для совместимости с предыдущими версиями PHP. Методы считаются открытыми по умолчанию.
- `protected` (защищенные). На эти элементы можно ссылаться только через принадлежащие объектам методы класса и такие же методы любых подклассов.
- `private` (закрытые). К представителям класса с этой областью видимости можно обращаться через методы этого же класса, но не через методы его подклассов.

Решение о том, какую область видимости применить, принимается на основе следующих положений:

- открытую (`public`) область видимости следует применять, когда к представителю класса нужен доступ из внешнего кода и когда расширенные классы должны его наследовать;
- защищенную (`protected`) область видимости необходимо использовать, когда к представителю класса не должно быть доступа из внешнего кода, но расширенные классы все же должны его наследовать;

- закрытую (`private`) область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода и когда расширенные классы не должны его наследовать.

Применение этих ключевых слов показано в примере 5.20.

Пример 5.20. Изменение области видимости свойства и метода

```
<?php
  class Example
  {
    var $name = "Michael";   // Нерекомендуемая форма, аналогичная public
    public $age = 23;        // Открытое свойство
    protected $usercount;    // Защищенное свойство

    private function admin() // Закрытый метод
    {
      // Сюда помещается код метода admin
    }
  }
?>
```

Статические методы

Метод можно определить в качестве статического, `static`, что будет означать, что он вызывается классом, а не объектом. Статический метод не имеет доступа к свойствам объекта. Порядок его создания и доступа к нему показан в примере 5.21.

Пример 5.21. Создание статического метода и обращение к нему

```
<?php
  User::pwd_string();

  class User
  {
    static function pwd_string()
    {
      echo "Пожалуйста, введите ваш пароль";
    }
  }
?>
```

Обратите внимание на вызов самого класса со статическим методом, при котором используется двойное двоеточие (также известное как оператор разрешения области видимости), а не сочетание символов `->`. Статические функции применяются для выполнения действий, относящихся к самому классу, а не к конкретным экземплярам класса. В примере 5.19 можно увидеть еще один вариант применения статического метода.

При попытке получить доступ из статической функции к свойству текущего объекта с помощью выражения $this->property или обратиться к другим свойствам объекта будет получено сообщение об ошибке.

Статические свойства

Большинство данных и методов применяются в экземплярах класса. Например, в классе `User` следует установить конкретный пароль пользователя или проверить, когда пользователь был зарегистрирован. Эти факты и операции имеют особое отношение к каждому конкретному пользователю и поэтому применяют специфические для экземпляра свойства и методы.

Но время от времени возникает потребность обслуживать данные, относящиеся целиком ко всему классу. Например, для отчета о том, сколько пользователей зарегистрировалось, будет храниться переменная, имеющая отношение ко всему классу `User`. Для таких данных PHP предоставляет статические свойства и методы.

В примере 5.21 было показано, что объявление представителей класса статическими делает их доступными и без создания экземпляров класса. Свойство, объявленное статическим, `static`, не может быть доступно непосредственно из экземпляра класса, но может быть доступно из статического метода.

В примере 5.22 определяется класс по имени `Test`, в котором содержатся статическое свойство и открытый метод.

Пример 5.22. Определение класса со статическим свойством

```
<?php
  $temp = new Test();

  echo "Тест А: " . Test::$static_property . "<br>";
  echo "Тест Б: " . $temp->get_sp()        . "<br>";
  echo "Тест В: " . $temp->static_property . "<br>";

  class Test
  {
    static $static_property = "Это статическое свойство";

    function get_sp()
    {
      return self::$static_property;
    }
  }
?>
```

Когда код будет запущен на выполнение, он выдаст следующую информацию:

```
Тест А: Это статическое свойство
Тест Б: Это статическое свойство

Notice: Undefined property: Test::$static_property
Тест В:
```

В этом примере показано, что на свойство `$static_property` можно ссылаться напрямую из самого класса, используя в тесте А оператор двойного двоеточия. Тест Б также может получить его значение путем вызова метода `get_sp` объекта `$temp`, созданного из класса `Test`. Но тест В терпит неудачу, потому что статическое свойство `$static_property` недоступно объекту `$temp`.

Обратите внимание на то, как метод `get_sp` получает доступ к свойству `$static_property`, используя ключевое слово `self`. Именно таким способом можно получить непосредственный доступ к статическому свойству или константе внутри класса.

Наследование

Как только класс будет создан, из него можно будет получить подкласс. Это сэкономит массу времени: вместо скрупулезного переписывания кода можно будет взять класс, похожий на тот, который следует создать, распространить его на подкласс и просто внести изменения в те места, которые будут иметь характерные особенности. Это достигается за счет использования ключевого слова `extends`.

В примере 5.23 класс `Subscriber` объявляется подклассом `User` путем использования инструкции `extends`.

Пример 5.23. Наследование и распространение класса

```
<?php
  $object           = new Subscriber;
  $object->name     = "Fred";
  $object->password = "pword";
  $object->phone    = "012 345 6789";
  $object->email    = "fred@bloggs.com";
  $object->display();

  class User
  {
    public $name, $password;

    function save_user()
    {
      echo "Сюда помещается код, сохраняющий данные пользователя";
    }
  }

  class Subscriber extends User
  {
    public $phone, $email;

    function display()
    {
      echo "Name:  " . $this->name     . "<br>";
      echo "Pass:  " . $this->password . "<br>";
      echo "Phone: " . $this->phone    . "<br>";
      echo "Email: " . $this->email;
    }
  }
?>
```

У исходного класса `User` имеются два свойства — `$name` и `$password`, а также метод для сохранения данных текущего пользователя в базе данных. Подкласс `Subscriber` расширяет этот класс за счет добавления еще двух свойств — `$phone` и `$email` и включения метода, отображающего свойства текущего объекта, который использует

переменную $this. Данная переменная ссылается на текущее значение объекта, к которому осуществляется доступ. Этот код выведет следующую информацию:

```
Name: Fred
Pass: pword
Phone: 012 345 6789
Email: fred@bloggs.com
```

Ключевое слово parent

Когда в подклассе создается метод с именем, которое уже фигурирует в его родительском классе, его инструкции переписывают инструкции из родительского класса. Иногда такое поведение идет вразрез с вашими желаниями, и вам нужно получить доступ к родительскому методу. Для этого можно воспользоваться инструкцией parent, которая показана в примере 5.24.

Пример 5.24. Переписывание метода и использование инструкции parent

```
<?php
  $object = new Son;
  $object->test();
  $object->test2();

  class Dad
  {
    function test()
    {
      echo "[Class Dad] Я твой отец<br>";
    }
  }

  class Son extends Dad
  {
    function test()
    {
      echo "[Class Son] Я Люк<br>";
    }

    function test2()
    {
      parent::test();
    }
  }
?>
```

Этот код создает класс по имени Dad (Отец), а затем подкласс по имени Son (Сын), который наследует свойства и методы родительского класса, а затем переписывает метод test. Поэтому, когда во второй строке кода вызывается метод test, выполняется новый метод. Единственный способ выполнения переписанного метода test в том варианте, в котором он существует в классе Dad, заключается в использовании инструкции parent, как показано в функции test2 класса Son. Этот код выведет следующую информацию:

```
[Class Son] Я Люк
[Class Dad] Я твой отец
```

Если нужно обеспечить вызов метода из текущего класса, можно воспользоваться ключевым словом `self`:

```
self::method();
```

Конструкторы подкласса

При распространении класса и объявлении собственного конструктора вы должны знать, что PHP не станет автоматически вызывать метод-конструктор родительского класса. Чтобы обеспечивалось выполнение всего кода инициализации, подкласс, как показано в примере 5.25, всегда должен вызывать родительские конструкторы.

Пример 5.25. Вызов конструктора родительского класса

```
<?php
  $object = new Tiger():
  echo "У тигров есть...<br>";
  echo "Мех: " . $object->fur . "<br>";
  echo "Полосы: " . $object->stripes;

  class Wildcat
  {
    public $fur: // У диких кошек есть мех

    function __construct()
    {
      $this->fur = "TRUE";
    }
  }

  class Tiger extends Wildcat
  {
    public $stripes; // У тигров есть полосы

    function __construct()
    {
      parent::__construct();  // Первоочередной вызов родительского конструктора
      $this->stripes = "TRUE";
    }
  }
?>
```

В этом примере используются обычные преимущества наследования. В классе `Wildcat` (Дикая кошка) создается свойство `$fur` (мех), которое хотелось бы использовать многократно, потому мы создаем класс `Tiger` (Тигр), наследующий свойство `$fur`, и дополнительно создаем еще одно свойство — `$stripes` (полосы). Чтобы проверить вызов обоих конструкторов, программа выводит следующую информацию:

```
У тигров есть...
Мех: TRUE
Полосы: TRUE
```

Методы Final

При необходимости помешать подклассу переписать метод суперкласса можно воспользоваться ключевым словом `final`. Как это делается, показано в примере 5.26.

Пример 5.26. Создание метода final

```
<?php
  class User
  {
    final function copyright()
    {
      echo "Этот класс был создан Джо Смитом";
    }
  }
?>
```

Усвоив содержание этой главы, вы должны получить твердое представление о возможностях PHP. Вы сумеете без особого труда воспользоваться функциями и при необходимости создать объектно-ориентированный код. В главе 6 мы завершим изучение основ PHP и рассмотрим работу с массивами.

Вопросы

1. В чем основное преимущество, получаемое при использовании функции?
2. Сколько значений может вернуть функция?
3. В чем разница между доступом к переменной по имени и по ссылке?
4. Что в PHP означает термин *«область видимости»*?
5. Как можно включить один файл PHP в другой?
6. Чем объект отличается от функции?
7. Как в PHP создаются новые объекты?
8. Какой синтаксис используется для создания подкласса из существующего класса?
9. Как можно заставить объект проинициализироваться при его создании?
10. Почему объявлять свойства внутри класса лучше явным образом?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

6 Массивы в PHP

В главе 3 у нас уже состоялось краткое знакомство с массивами в PHP, позволившее составить первичное представление об их возможностях. В данной главе будет продемонстрирован большой арсенал приемов работы с массивами, некоторые из них при наличии у вас опыта работы с языками со строгой типизацией, например C, могут удивить своей простотой и изяществом.

Массивы — одна из составляющих популярности PHP. Кроме того что они не дают умереть со скуки при создании кода для работы со сложными структурами данных, они предоставляют множество невероятно быстрых способов доступа к данным.

Основные подходы к массивам

Массивы уже рассматривались в виде группы склеенных вместе спичечных коробков. Их можно представить также в виде нитки бус, где бусины обозначают переменные, которые могут быть числовыми, строковыми и даже другими массивами. Массивы похожи на нитки бус, потому что каждый элемент имеет собственное место и у каждого элемента (кроме первого и последнего) с обеих сторон есть другие элементы.

Часть массивов использует ссылки по числовым индексам, другая — позволяет работать с буквенно-цифровыми идентификаторами. Встроенные функции дают возможность проводить сортировку, добавлять и удалять отрезки и перебирать элементы для обработки каждого из них, используя специальный вид цикла. А за счет размещения одного или нескольких массивов внутри других массивов можно создавать массивы любой размерности.

Массивы с числовой индексацией

Представим, что перед вами поставлена задача создать простой сайт для компании по поставке товаров для офиса и сейчас вы ведете разработку его раздела, в котором предложены различные сорта бумаги. Как вариант, различные единицы хранения данной категории можно поместить в числовой массив, чтобы получить возможность управлять ими. Простейший способ реализации этого подхода показан в примере 6.1.

Пример 6.1. Добавление элементов в массив

```
<?php
  $paper[] = "Copier";
  $paper[] = "Inkjet";
  $paper[] = "Laser";
  $paper[] = "Photo";

  print_r($paper);
?>
```

В этом примере при каждом присваивании массиву `$paper` значения для хранения последнего используется первое же свободное место, а значение существующего в PHP внутреннего указателя увеличивается на единицу, чтобы указывать на свободное место, готовое для следующей вставки значения. Уже известная нам функция `print_r` (которая выводит на экран содержимое переменной, массива или объекта) применяется для проверки правильности заполнения массива. Результат ее работы имеет следующий вид:

```
Array
(
  [0] => Copier
  [1] => Inkjet
  [2] => Laser
  [3] => Photo
)
```

Предыдущий код может быть написан и так, как показано в примере 6.2, где для каждого элемента указывается точное место в массиве. Но, как видите, такой подход требует набора лишних символов и усложняет обслуживание кода в том случае, если понадобится вставлять товары в массив или удалять их оттуда. Поэтому, если не нужно указывать какой-нибудь другой порядок размещения элементов в массиве, лучше все же позволить PHP самостоятельно заниматься их расстановкой.

Пример 6.2. Добавление в массив элементов с конкретным указанием их мест

```
<?php
  $paper[0] = "Copier";
  $paper[1] = "Inkjet";
  $paper[2] = "Laser";
```

```
  $paper[3] = "Photo";

  print_r($paper);
?>
```

Этот пример выведет такую же информацию, как и предыдущий, но на разрабатываемом сайте вы вряд ли будете пользоваться функцией `print_r`, поэтому в примере 6.3 показано, как с помощью цикла можно распечатать сведения о различных типах бумаги, предлагаемых на сайте.

Пример 6.3. Добавление элементов в массив и извлечение их из массива

```
<?php
  $paper[] = "Copier";
  $paper[] = "Inkjet";
  $paper[] = "Laser";
  $paper[] = "Photo";

  for ($j = 0 ; $j < 4 ; ++$j)
    echo "$j: $paper[$j]<br>";
?>
```

Этот пример выведет следующую информацию:

```
0: Copier
1: Inkjet
2: Laser
3: Photo
```

Итак, вы увидели два способа добавления элементов к массиву и один из способов ссылки на них, но PHP предлагает и много других способов, на которых я кратко остановлюсь в дальнейшем. Сначала рассмотрим другой тип массива.

Ассоциативные массивы

Конечно, можно отслеживать элементы массива по индексам, но тогда придется помнить, какой именно номер на какой товар ссылается. Кроме того, за вашим кодом трудно будет уследить другим программистам.

Самое время обратиться к ассоциативным массивам. Использование этих массивов позволяет ссылаться на элементы массива по именам, а не по номерам. В примере 6.4 приводится расширенная версия предыдущего кода, где каждому элементу массива дается имя для идентификации и более длинное и информативное строковое значение.

Пример 6.4. Добавление элементов к ассоциативному массиву и их извлечение

```
<?php
  $paper['copier'] = "Copier & Multipurpose";
  $paper['inkjet'] = "Inkjet Printer";
  $paper['laser']  = "Laser Printer";
```

```
  $paper['photo']  = "Photographic Paper";

  echo $paper['laser'];
?>
```

Теперь у каждого элемента вместо числа (не содержащего никакой полезной информации, кроме позиции элемента в массиве) имеется уникальное имя, по которому на него можно сослаться где-нибудь в другом месте, как в случае с инструкцией `echo`, выводящей на экран `Laser Printer`. Имена (`copier`, `inkjet` и т. д.) называются *индексами*, или *ключами*, а присвоенные им элементы (например, `Laser Printer`) — *значениями*.

Это весьма мощное свойство PHP часто применяется при извлечении информации из кода XML и HTML. Например, HTML-парсер, используемый в поисковой системе, может помещать все элементы веб-страницы в ассоциативный массив, имена которого отображают структуру страницы:

```
$html['title'] = "Моя веб-страница";
$html['body']  = "... тело веб-страницы ...";
```

Вполне вероятно, что программа разобьет все найденные на странице ссылки и поместит их в другой массив, а все заголовки и подзаголовки — еще в один массив. При использовании ассоциативных, а не числовых массивов код, ссылающийся на все эти элементы, проще будет создавать и отлаживать.

Присваивание с использованием ключевого слова array

Мы уже видели, как элементам массива присваиваются значения путем последовательного добавления к этому массиву новых элементов. Но это слишком затянутый процесс, независимо от того, что при этом происходит: вы определяете ключи, числовые идентификаторы или позволяете PHP неявным образом заниматься присваиванием числовых идентификаторов. Есть более краткий и быстрый способ присваивания значений с использованием ключевого слова `array`. В примере 6.5 показаны оба массива — числовой и ассоциативный, значения которым присваиваются именно этим способом.

Пример 6.5. Добавление элементов к массиву с использованием ключевого слова array

```
<?php
  $p1 = array("Copier", "Inkjet", "Laser", "Photo");

  echo "Элемент массива p1: " . $p1[2] . "<br>";

  $p2 = array('copier' => "Copier & Multipurpose",
              'inkjet' => "Inkjet Printer",
              'laser'  => "Laser Printer",
              'photo'  => "Photographic Paper");

  echo "Элемент массива p2: " . $p2['inkjet'] . "<br>";
?>
```

В первой части этого кодового фрагмента массиву `$p1` присваивается старое, укороченное описание товара. Здесь используются четыре элемента, поэтому они занимают позиции от 0 до 3. Инструкция `echo` выводит следующий текст:

```
Элемент массива p1: Laser
```

Во второй части кода массиву `$p2` присваиваются ассоциативные идентификаторы и сопутствующие им длинные описания товаров. Для этого применяется формат *`индекс => значение`*. Применение оператора `=>` похоже на использование простого оператора присваивания `=`, за исключением того, что значение присваивается *индексу*, а не *переменной*. После этого индекс приобретает неразрывную связь с этим значением до тех пор, пока ему не будет присвоено другое значение. Поэтому команда `echo` выводит следующий текст:

```
Элемент массива p2: Inkjet Printer
```

В том, что `$p1` и `$p2` принадлежат к разным типам массивов, можно убедиться, если вставить в код две следующие команды, вызывающие ошибку неопределенного индекса или ошибку неопределенного смещения, поскольку для каждого из массивов используется неподходящий идентификатор:

```
echo $p1['inkjet'];  // Неопределенный индекс
echo $p2['3'];       // Неопределенное смещение
```

Цикл foreach...as

Создатели PHP постарались сделать этот язык простым в использовании. Поэтому они не остановились на уже имеющихся структурах организации цикла, а добавили еще одну структуру, специально предназначенную для массивов, — цикл `foreach...as`. Используя этот цикл, можно поочередно перебрать все элементы массива и произвести с ними какие-нибудь действия.

Процесс начинается с первого элемента и заканчивается последним, поэтому вам даже не нужно знать, сколько элементов присутствует в массиве.

В примере 6.6 показано, как цикл `foreach...as` может использоваться для переписывания кода примера 6.3.

Пример 6.6. Последовательный перебор элементов числового массива с использованием цикла foreach...as

```
<?php
  $paper = array("Copier", "Inkjet", "Laser", "Photo");
  $j = 0;

  foreach ($paper as $item)
  {
    echo "$j: $item<br>";
    ++$j;
  }
?>
```

Когда PHP встречает инструкцию `foreach`, он извлекает первый элемент массива и помещает его значение в переменную, указанную после ключевого слова `as`, и при каждом возвращении управления инструкции `foreach` в эту переменную помещается значение следующего элемента массива. В данном случае переменной `$item` присваиваются по очереди все четыре значения, хранящиеся в массиве `$paper`. Как только будут использованы все значения, выполнение цикла завершается. Этот код выводит точно такую же информацию, что и код примера 6.3.

Теперь посмотрим, как `foreach` работает с ассоциативным массивом. В примере 6.7 переписана вторая часть примера 6.5.

Пример 6.7. Последовательный перебор элементов ассоциативного массива с использованием цикла foreach...as

```
<?php
  $paper = array('copier' => "Copier & Multipurpose",
                 'inkjet' => "Inkjet Printer",
                 'laser'  => "Laser Printer",
                 'photo'  => "Photographic Paper");

  foreach ($paper as $item => $description)
    echo "$item: $description<br>";
?>
```

Вспомним, что ассоциативным массивам не требуются числовые индексы, поэтому переменная `$j` в данном примере не используется. Вместо этого каждый элемент массива `$paper` вводится в пару «ключ — значение», представленную переменными `$item` и `$description`, из которых эта пара выводится на экран в следующем виде:

```
copier: Copier & Multipurpose
inkjet: Inkjet Printer
laser: Laser Printer
photo: Photographic Paper
```

В качестве альтернативы синтаксису `foreach...as` можно воспользоваться функцией `list` в сочетании с функцией `each` (пример 6.8).

Пример 6.8. Последовательный перебор элементов ассоциативного массива с помощью функций each и list

```
<?php
  $paper = array('copier' => "Copier & Multipurpose",
                 'inkjet' => "Inkjet Printer",
                 'laser'  => "Laser Printer",
                 'photo'  => "Photographic Paper");

  while (list($item, $description) = each($paper))
    echo "$item: $description<br>";
?>
```

В этом примере организуется цикл `while`, который будет продолжать работу до тех пор, пока функция `each` не вернет значение `FALSE`. Функция `each` ведет себя как `foreach`: она возвращает из массива `$paper` массив, содержащий пару «ключ — значение», а затем перемещает встроенный указатель на следующую пару в ис-

ходном массиве. Когда возвращать становится нечего, функция `each` возвращает значение `FALSE`.

Функция `list` в качестве аргументов принимает массив (в данном случае пару «ключ — значение», возвращенную функцией `each`), а затем присваивает значения массива переменным, перечисленным внутри круглых скобок.

Лучше понять работу функции `list` можно из примера 6.9, где массив создается из двух строк — `Alice` и `Bob`, а затем передается функции `list`, которая присваивает эти строки переменным `$a` и `$b`.

Пример 6.9. Использование функции list

```
<?php
  list($a, $b) = array('Alice', 'Bob');
  echo "a=$a b=$b";
?>
```

Этот код выводит следующий текст:

```
a=Alice b=Bob
```

Итак, для перебора элементов массива можно применять различные подходы. Можно воспользоваться конструкцией `foreach...as` для создания цикла, извлекающего значения в переменную, которая следует за `as`, или функцией `each` и создать собственную систему циклической обработки.

Многомерные массивы

Простая конструктивная особенность синтаксиса массивов PHP позволяет создавать массивы более чем с одним измерением. Фактически можно создавать массивы любой размерности (хотя приложения редко нуждаются в массивах с размерностью больше трех).

Эта особенность заключается в возможности включать целый массив в состав другого массива, а также делать это снова и снова, как в старом стишке про блох, которых кусают другие блохи поменьше, а тех, в свою очередь, кусают свои блохи, и так до бесконечности.

Рассмотрим, как это работает, для чего возьмем ассоциативный массив из предыдущего примера и расширим его (пример 6.10).

Пример 6.10. Создание многомерного ассоциативного массива

```
<?php
  $products = array(

    'paper' => array(

      'copier' => "Copier & Multipurpose",
      'inkjet' => "Inkjet Printer",
```

```
      'laser'  => "Laser Printer",
      'photo'  => "Photographic Paper"),

    'pens' => array(

      'ball'   => "Ball Point",
      'hilite' => "Highlighters",
      'marker' => "Markers"),

    'misc' => array(

      'tape'   => "Sticky Tape",
      'glue'   => "Adhesives",
      'clips'  => "Paperclips"
    )
  );

  echo "<pre>";

  foreach ($products as $section => $items)
    foreach ($items as $key => $value)
      echo "$section:\t$key\t($value)<br>";

  echo "</pre>";
?>
```

Чтобы упростить понимание начинающего разрастаться кода, я переименовал часть элементов. Например, поскольку предыдущий массив `$paper` стал лишь подразделом более крупного массива, главный массив теперь называется `$products`. В этом массиве присутствуют три элемента: бумага — `paper`, ручки — `pens` и разные товары — `misc`, и каждый из них содержит другой массив, состоящий из пар «ключ — значение».

При необходимости эти подмассивы могут содержать другие массивы. Например, элемент шариковые ручки — `ball` — может содержать множество типовых и цветовых решений этого товара, имеющихся в интернет-магазине. Но пока я ограничил код глубиной в два измерения.

После присваивания массивам данных для вывода различных значений я воспользовался парой вложенных циклов `foreach...as`. Внешний цикл извлекает из верхнего уровня массива основные разделы, а внутренний цикл извлекает для категорий в каждом разделе пары «ключ — значение».

Если вспомнить, что все уровни массива работают одинаково (являясь парой «ключ — значение»), можно без особого труда создать код для доступа к любому элементу на любом уровне.

В инструкции `echo` используется управляющий символ PHP `\t`, который выводит знак табуляции.

Хотя знаки табуляции для браузеров, как правило, ничего не значат, я использовал их в разметке, применив теги `<pre>...</pre>`, которые предписывают браузеру

форматировать текст с сохранением предварительного формата и фиксированной ширины и *не* игнорировать неотображаемые символы вроде знаков табуляции и переводов строки. Текст, выводимый этим кодом, будет иметь следующий вид:

```
paper: copier (Copier & Multipurpose)
paper: inkjet (Inkjet Printer)
paper: laser  (Laser Printer)
paper: photo  (Photographic Paper)
pens:  ball   (Ball Point)
pens:  hilite (Highlighters)
pens:  marker (Markers)
misc:  tape   (Sticky Tape)
misc:  glue   (Adhesives)
misc:  clips  (Paperclips)
```

Непосредственный доступ к конкретному элементу массива можно получить, добавив квадратные скобки:

```
echo $products['misc']['glue'];
```

Этот код выводит значение `Adhesives`.

Можно также создать числовой многомерный массив, непосредственный доступ к элементам которого можно будет получать по индексам, а не по буквенно-цифровым идентификаторам. В примере 6.11 создается шахматная доска с фигурами на исходных позициях.

Пример 6.11. Создание многомерного числового массива

```
<?php
  $chessboard = array(
    array('r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'),
    array('p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array('P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'),
    array('R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R')
  );

  echo "<pre>";

  foreach ($chessboard as $row)
  {
    foreach ($row as $piece)
      echo "$piece ";

    echo "<br>";
  }

  echo "</pre>";
?>
```

В этом примере буквы в нижнем регистре представляют собой черные фигуры, а в верхнем регистре — белые. Используются следующие обозначения: `r` — rook

(ладья), n — knight (конь), b — bishop (слон), k — king (король), q — queen (ферзь) и p — pawn (пешка). Для последовательного перебора массива и демонстрации его содержимого снова используется пара вложенных циклов foreach...as. Внешний цикл обрабатывает каждую горизонталь и помещает ее в переменную $row, которая сама по себе является массивом, поскольку для каждой горизонтали массив шахматной доски — $chessboard — задействует подмассив. В этом цикле используются две инструкции, поэтому они заключены в фигурные скобки.

Внутренний цикл обрабатывает каждую клетку горизонтали, выводя хранящийся в ней символ ($piece), за которым следует пробел (чтобы выводимый текст имел форму шахматной доски). У этого цикла одна инструкция, которую не нужно заключать в фигурные скобки. Теги <pre> и </pre> обеспечивают правильную форму выводимого текста:

```
r n b q k b n r
p p p p p p p p
P P P P P P P P
R N B Q K B N R
```

Используя квадратные скобки, можно получить непосредственный доступ к любому элементу этого массива:

```
echo $chessboard[7][3];
```

Эта инструкция выведет букву Q в верхнем регистре, которая является значением восьмого вниз по вертикали и четвертого по горизонтали элемента (следует помнить, что индексы массива начинаются с нуля, а не с единицы).

Использование функций для работы с массивами

С функциями list и each вы уже знакомы, но в PHP имеется множество других функций, предназначенных для работы с массивами. Их полный перечень представлен в документации (http://tinyurl.com/arraysinphp). Но некоторые из этих функций играют настолько важную роль в программировании на PHP, что мы изучим их подробнее.

is_array

Массивы и переменные используют одно и то же пространство имен. Это означает, что нельзя иметь строковую переменную по имени $fred и массив, который также называется $fred. Если есть сомнения и в коде программы нужно проверить, является ли переменная массивом, можно воспользоваться функцией is_array:

```
echo (is_array($fred)) ? "Это массив" : "Это не массив";
```

Заметьте, что переменной $fred не присвоено никакого значения, поэтому будет выведено сообщение о неопределенной переменной — Undefined variable.

count

Несмотря на то что функция `each` и структура организации цикла `foreach...as` предоставляют отличные способы последовательного перебора всего содержимого массива, иногда нужно точно знать, сколько элементов содержится в вашем массиве, особенно если вы будете обращаться к ним напрямую. Для подсчета всех элементов на верхнем уровне массива используется следующая команда:

```
echo count($fred);
```

Если нужно узнать, сколько всего элементов содержится в многомерном массиве, можно воспользоваться следующей инструкцией:

```
echo count($fred, 1);
```

Второй параметр является необязательным и устанавливает режим использования. Он может иметь либо нулевое значение, чтобы ограничить подсчет только верхним уровнем, либо единичное — для принудительного включения рекурсивного подсчета еще и всех элементов, содержащихся в подмассивах.

sort

Сортировка является настолько распространенной операцией, что PHP предоставляет для нее встроенную функцию. В наипростейшей форме ее можно использовать следующим образом:

```
sort($fred);
```

В отличие от некоторых других функций, сортировка будет работать непосредственно с предоставленным ей массивом, а не возвращать новый массив с отсортированными элементами. Вместо этого она вернет значение `TRUE` при успешном выполнении сортировки и `FALSE` — при возникновении ошибки. Эта функция поддерживает также несколько флагов. Основные два, которые вам могут пригодиться, предписывают проведение либо числовой, либо строковой сортировки:

```
sort($fred, SORT_NUMERIC);
sort($fred, SORT_STRING);
```

Массив можно также отсортировать в обратном порядке, воспользовавшись функцией `rsort`:

```
rsort($fred, SORT_NUMERIC);
rsort($fred, SORT_STRING);
```

shuffle

Иногда, например при создании игры или при игре в карты, требуется, чтобы элементы массива располагались в случайном порядке:

```
shuffle($cards);
```

Как и функция `sort`, функция `shuffle` работает непосредственно с предоставленным ей массивом и возвращает значение `TRUE` в случае успешного завершения работы и `FALSE` — при возникновении ошибки.

explode

Это очень полезная функция, позволяющая взять строку, содержащую несколько элементов, отделенных друг от друга одиночным символом (или строкой символов), а затем поместить каждый из этих элементов в массив. В примере 6.12 показан один из случаев полезного применения этой функции, который заключается в разбиении предложения на слова и помещении всех слов, из которого оно состоит, в массив.

Пример 6.12. Извлечение слов из строки в массив с использованием пробелов

```
<?php
  $temp = explode(' ', "Это предложение из пяти слов");
  print_r($temp);
?>
```

Этот пример выводит следующую информацию (которая при просмотре в браузере будет отображена в одной строке):

```
Array
(
  [0] => Это
  [1] => предложение
  [2] => из
  [3] => пяти
  [4] => слов
)
```

Первый параметр — разделитель — необязательно должен быть пробелом или даже одиночным символом. В примере 6.13 показан этот же код в несколько измененном виде.

Пример 6.13. Извлечение слов, разделенных символами ***, из строки в массив

```
<?php
  $temp = explode('***', "Это***предложение***со***звездочками");
  print_r($temp);
?>
```

Код примера 6.13 выводит следующую информацию:

```
Array
(
  [0] => Это
  [1] => предложение
  [2] => со
  [3] => звездочками
)
```

extract

Иногда бывает удобно превратить пары «ключ — значение» из массива в переменные PHP. Один из таких случаев — это обработка переменных `$_GET` или `$_POST`, отправленных формой сценарию PHP.

Когда форма передается через Интернет, веб-сервер распаковывает переменные и помещает их в глобальный массив, предназначенный для сценария PHP. Если переменные были отправлены методом GET, они будут помещены в ассоциативный массив `$_GET`, а при отправке методом POST — в ассоциативный массив `$_POST`.

Разумеется, можно перебрать все элементы этих ассоциативных массивов, воспользовавшись уже рассмотренными в этой главе способами. Но иногда нужно лишь сохранить отправленные значения в переменных для дальнейшего использования. В таком случае можно заставить PHP проделать эту работу за вас в автоматическом режиме:

```
extract($_GET);
```

Таким образом, к примеру, если параметр строки запроса `q` отправлен сценарию PHP наряду со связанным с ним значением `Hi there`, будет создана новая переменная по имени `$q`, которой будет присвоено это значение.

Но к описанному подходу нужно относиться осторожно, поскольку если какие-нибудь извлекаемые переменные конфликтуют с уже определенными переменными, то существующие переменные будут переписаны. Чтобы избежать этого, можно воспользоваться одним из многих дополнительных параметров, доступных в данной функции:

```
extract($_GET, EXTR_PREFIX_ALL, 'fromget');
```

В этом случае имена всех новых переменных будут начинаться с заданного строкового префикса, за которым следует символ подчеркивания, в результате чего `$q` превратится в `$fromget_q`. Я настоятельно рекомендую при обработке массивов `$_GET` и `$_POST` или любого другого массива, ключи которого могут контролироваться пользователем, применять именно эту версию функции. Ведь злоумышленники могут отправлять ключи, специально подобранные для того, чтобы переписать переменные с часто используемыми именами и таким образом угрожать вашему сайту.

compact

Иногда нужно воспользоваться функцией `compact`, являющейся противоположностью функции `extract`, чтобы создать массив из переменных и их значений. Применение этой функции показано в примере 6.14.

Пример 6.14. Использование функции compact

```
<?php
  $fname         = "Doctor";
```

```
  $sname         = "Who";
  $planet        = "Gallifrey";
  $system        = "Gridlock";
  $constellation = "Kasterborous";

  $contact = compact('fname', 'sname', 'planet', 'system', 'constellation');

  print_r($contact);
?>
```

В результате запуска кода из примера 6.14 будет выведена следующая информация:

```
Array
(
  [fname] => Doctor
  [sname] => Who
  [planet] => Gallifrey
  [system] => Gridlock
  [constellation] => Kasterborous
)
```

Обратите внимание на то, что функции `compact` нужны имена переменных, стоящие в кавычках и не содержащие начального символа `$`. Причина заключается в том, что функция `compact` ищет список имен переменных, а не их значений.

Эту функцию можно использовать также для отладки, когда нужно быстро просмотреть несколько переменных вместе с их значениями, как в примере 6.15.

Пример 6.15. Использование функции compact для отладки программы

```
<?php
  $j       = 23;
  $temp    = "Hello";
  $address = "1 Old Street";
  $age     = 61;

  print_r (compact (explode (' ','j temp address age')));
?>
```

Работа примера основана на использовании функции `explode` для извлечения всех слов из строки в массив, который затем передается функции `compact`, а она, в свою очередь, возвращает массив функции `print_r`, которая в итоге показывает его содержимое.

Если скопировать и вставить строку кода, содержащую вызов функции `print_r`, то в ней нужно будет лишь изменить имена переменных, чтобы быстро вывести группу их значений. В этом примере выводимая информация будет иметь следующий вид:

```
Array
(
  [j] => 23
  [temp] => Hello
  [address] => 1 Old Street
  [age] => 61
)
```

reset

Когда с помощью конструкции `foreach...as` или функции `each` осуществляется последовательный перебор элементов массива, они перемещают внутренний указатель PHP, который показывает, какой из элементов массива нужно извлечь в следующий раз. Если коду программы понадобится вернуться к началу массива, то можно воспользоваться функцией `reset`, а она к тому же вернет значение элемента, на котором остановился указатель. Эта функция может быть использована следующим образом:

```
reset($fred);           // Отбрасывание возвращаемого значения
$item = reset($fred);   // Сохранение первого элемента массива
                        // в переменной $ item
```

end

Можно также переместить внутренний указатель элемента массива PHP на последний элемент, воспользовавшись для этого функцией `end`, которая, кроме этого, возвращает значение элемента и может быть использована следующим образом:

```
end($fred);
$item = end($fred);
```

В этой главе завершается введение в основы PHP. Теперь, используя приобретенные навыки, вы должны справиться с написанием довольно сложных программ. В следующей главе будет рассмотрено применение PHP для решения наиболее распространенных практических задач.

Вопросы

1. В чем разница между числовым и ассоциативным массивом?
2. Каковы основные преимущества использования ключевого слова `array`?
3. В чем разница между `foreach` и `each`?
4. Как создается многомерный массив?
5. Как определить количество элементов в массиве?
6. Каково назначение функции `explode`?
7. Как вернуть внутренний указатель элемента массива PHP на его первый элемент?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

7 Практикум по программированию на PHP

В предыдущих главах рассматривались элементы языка PHP. А эта глава предназначена для приобретения навыков программирования в процессе решения типовых, но тем не менее важных практических задач. Здесь будут представлены лучшие способы обработки строк, позволяющие получить вполне понятный и лаконичный код, который включает усовершенствованное управление отображением даты и времени и демонстрируется браузерами в точном соответствии с вашими желаниями. Вы также узнаете о создании и разнообразных способах изменения файлов, включая файлы, выложенные на сайт пользователями.

Функция printf

Ранее нам уже встречались функции `print` и `echo`, которые использовались для простого вывода текста в браузер. Но существует намного более мощная функция `printf`, управляющая форматом выводимых данных путем вставки в строку специальных форматирующих символов. Функция `printf` ожидает, что для каждого форматирующего символа будет предоставлен аргумент, который будет отображаться с использованием заданного формата. Например, в следующем фрагменте применяется спецификатор преобразования `%d`, чтобы значение `3` отображалось в виде десятичного числа:

```
printf("В вашей корзине находится %d покупки", 3);
```

Если заменить `%d` на `%b`, значение `3` будет отображено в виде двоичного числа (`11`). В табл. 7.1 показаны поддерживаемые функцией спецификаторы преобразования.

Таблица 7.1. Спецификаторы преобразования, используемые в функции printf

Спецификатор	Преобразование, осуществляемое с аргументом arg	Пример (для arg, имеющего значение 123)
%	Отображение символа % (аргументы не требуются)	%
b	Отображение arg в виде двоичного целого числа	1111011
c	Отображение ASCII-символа с кодом, содержащимся в arg	{
d	Отображение arg в виде целого десятичного числа со знаком	123
e	Отображение arg с использованием научной формы записи	1.23000e+2
f	Отображение arg в виде числа с плавающей точкой	123.000000
o	Отображение arg в виде восьмеричного целого числа	173
s	Отображение arg в виде строки	123
u	Отображение arg в виде беззнакового десятичного числа	123
x	Отображение arg в виде шестнадцатеричного числа с символами в нижнем регистре	7b
X	Отображение arg в виде шестнадцатеричного числа с символами в верхнем регистре	7B

В функции `printf` можно использовать любое количество спецификаторов, если им передается соответствующее количество аргументов и если каждый спецификатор предваряется символом `%`. Поэтому следующий код имеет вполне допустимую форму и выводит предложение: `Меня зовут Симон. Мне 33 года, то есть 21 в шестнадцатеричном представлении`:

```
printf("Меня зовут %s. Мне %d года, то есть %X в шестнадцатеричном
  представлении", 'Симон', 33, 33);
```

Если пропустить какой-нибудь аргумент, то будет получена ошибка синтаксического разбора, информирующая о том, что правая круглая скобка (`)`) была встречена в неожиданном месте.

Более полезный с практической точки зрения пример использования функции `printf` устанавливает цвета в коде HTML, используя для этого десятичные числа. Предположим, к примеру, что вам нужен цвет, составленный из трех значений: 65 для красного, 127 для зеленого и 245 для синего цвета, но вам не хочется самостоятельно переводить эти числа в шестнадцатеричный формат. Для этого есть более простое решение:

```
printf("<span style='color:#%X%X%X'>Привет</span>", 65, 127, 245);
```

Тщательно разберитесь с цветовой спецификацией, которая заключена в апострофы (`''`). Сначала ставится знак решетки (`#`), ожидаемый в цветовой спецификации.

Затем следуют три форматирующие спецификации `%X`, по одной для каждого из ваших чисел. В результате эта команда выдаст такой текст:

```
<span style='color:#417FF5'>Привет</span>
```

Обычно представляется удобным в качестве аргументов `printf` использовать переменные или выражения. Например, если значения для цветового решения хранятся в трех переменных: `$r`, `$g` и `$b`, то более темный оттенок можно получить с помощью выражения:

```
printf("<span style='color:#%X%X%X'>Привет</span>", $r-20, $g-20, $b-20);
```

Настройка представления данных

Можно указать не только тип преобразования, но и точность отображаемого результата. Например, суммы в валюте отображаются, как правило, с точностью до двух цифр. Но после вычисления значение может иметь более высокую точность (например, если разделить 123,42 на 12, то получится 10,285). Чтобы обеспечить правильное внутреннее хранение таких значений, но при этом организовать их отображение с точностью только до двух цифр, можно между символом `%` и спецификатором преобразования вставить строку `".2"`:

```
printf("Результат: $%.2f", 123.42 / 12);
```

Эта команда выводит следующий текст:

```
Результат: $10.29
```

Но доступные средства управления на этом не заканчиваются, потому что можно также указать, где и чем — нулями или пробелами — дополнить выводимый текст, поставив перед спецификатором соответствующие значения. В примере 7.1 показаны пять возможных комбинаций.

Пример 7.1. Настройка представления данных точности

```
<?php
  echo "<pre>"; // Тег, позволяющий отображать все пустые пространства

  // Дополнение пробелами до 15 знакомест
  printf("Результат равен $%15f\n", 123.42 / 12);

  // Дополнение нулями до 15 знакомест
  printf("Результат равен $%015f\n", 123.42 / 12);

  // Дополнение пробелами до 15 знакомест и вывод с точностью
  // до двух десятичных знаков
  printf("Результат равен $%15.2f\n", 123.42 / 12);

  // Дополнение нулями до 15 знакомест и вывод с точностью
  // до двух десятичных знаков
```

```
  printf("Результат равен $%015.2f\n", 123.42 / 12);

  // Дополнение символами # до 15 знакомест и вывод с точностью
  // до двух десятичных знаков
  printf("Результат равен $%'#15.2f\n", 123.42 / 12);
?>
```

Этот пример выводит следующий текст:

```
Результат равен $      10.285000
Результат равен $00000010.285000
Результат равен $          10.29
Результат равен $000000000010.29
Результат равен $##########10.29
```

Проследить работу спецификатора проще, если изучать его слева направо (табл. 7.2). Обратите внимание на следующие моменты.

- Крайним справа символом спецификатора преобразования в данном случае является `f`, означающий преобразование в число с плавающей точкой.
- Если сразу же перед спецификатором преобразования стоит сочетание точки и числа, значит этим числом указана точность выводимой информации.
- Независимо от присутствия спецификатора точности, если в общем спецификаторе есть число, оно представляет собой количество знакомест, выделяемых под выводимую информацию. В предыдущем примере это количество равно 15. Если выводимая информация уже равна количеству выделяемых знако-мест или превышает его, то данный аргумент игнорируется.
- После крайнего слева символа `%` разрешается поставить символ `0`, который игнорируется, если не указано количество выделяемых знакомест. Если это количество указано, то вместо пробелов дополнение производится нулями. Если нужно, чтобы пустующие знако-места заполнялись не нулями или пробелами, а каким-нибудь другим символом, то можно выбрать любой символ, поставив перед ним одинарную кавычку: `'#`.
- В левой части спецификатора ставится символ `%`, с позиции которого и начинается преобразование.

Таблица 7.2. Компоненты спецификатора преобразования

Начало преобразования	Дополняющий символ	Количество дополняющих символов	Точность отображения	Спецификатор преобразования	Примеры
%		15		f	10.285000
%	0	15	.2	f	000000000010.29
%	'#	15	.4	f	########10.2850

Дополнение строк

Дополнить до требуемой длины можно не только числа, но и строки, выбирая для этого различные дополняющие символы и даже левую или правую границы выравнивания. Возможные варианты показаны в примере 7.2.

Пример 7.2. Дополнение строк

```
<?php
  echo "<pre>"; // Тег, позволяющий отображать все пустые пространства

  $h = 'Rasmus';

  printf("[%s]\n",         $h);  // Стандартный вывод строки
  printf("[%12s]\n",       $h);  // Выравнивание пробелами по правому краю
                                 // до ширины 12
  printf("[%-12s]\n",      $h);  // Выравнивание пробелами по левому краю
  printf("[%012s]\n",      $h);  // Дополнение нулями
  printf("[%'#12s]\n\n",   $h);  // Использование специально выбранного
                                 // символа дополнения '#'
  $d = 'Rasmus Lerdorf';

  printf("[%12.8s]\n",     $d);  // Выравнивание по правому краю с усечением
                                 // до 8 символов
  printf("[%-12.12s]\n",   $d);  // Выравнивание по левому краю с усечением
                                 // до 12 символов
  printf("[%-'@12.10s]\n", $d);  // Выравнивание по левому краю, дополнение
                                 // символом '@', усечение до 10 символов
?>
```

Обратите внимание на то, что для получения нужной разметки на веб-странице я воспользовался HTML-тегом `<pre>`, который оставляет нетронутыми все пустые пространства и после каждой отображаемой строки выводит на экран символ новой строки `\n`. В этом примере выводится следующий текст:

```
[Rasmus]
[      Rasmus]
[Rasmus      ]
[000000Rasmus]
[######Rasmus]

[    Rasmus L]
[Rasmus Lerdo]
[Rasmus Ler@@]
```

Если при указании количества знако-мест длина строки уже равна этому количеству или превышает его, это указание будет проигнорировано, *если только* заданное количество символов, до которого нужно усечь строку, не будет меньше указанного количества знако-мест.

В табл. 7.3 показаны спецификаторы преобразования строки, разложенные на компоненты.

Таблица 7.3. Компоненты спецификаторов преобразования строки

Начало преобразования	Выравнивание по левому или по правому краю	Дополняющий символ	Количество дополняющих символов	Усечение	Спецификатор преобразования	Примеры
%					s	[Rasmus]
%	–		10		s	[Rasmus]
%		'#	8	.4	s	[####Rasm]

Функция sprintf

Зачастую результат преобразования нужно не выводить на экран, а использовать в самом коде программы. Для этого предназначена функция `sprintf`. Она позволяет не отправлять выходную информацию браузеру, а присваивать ее какой-нибудь переменной.

Функции `sprintf` можно использовать для преобразования, возвращающего шестнадцатеричное строковое значение для цветового сочетания RGB 65, 127, 245, которое присваивается переменной `$hexstring`:

```
$hexstring = sprintf("%X%X%X", 65, 127, 245);
```

Или же она может пригодиться для сохранения выходной информации, которую нужно будет вывести на экран чуть позже:

```
$out = sprintf("Результат: $%.2f", 123.42 / 12);
echo $out;
```

Функции даты и времени

Для отслеживания даты и времени в PHP используются стандартные отметки времени UNIX, представляющие собой простое количество секунд, прошедших с начала отсчета — 1 января 1970 года. Для определения текущей отметки времени можно воспользоваться функцией `time`:

```
echo time();
```

Поскольку значение хранится в секундах, для получения метки времени ровно через неделю можно воспользоваться следующим выражением, в котором к возвращаемому значению прибавляется 7 дней × 24 часа × 60 минут × 60 секунд:

```
echo time() + 7 * 24 * 60 * 60;
```

Если нужно получить отметку времени для заданной даты, можно воспользоваться функцией `mktime`. Она выводит отметку времени `946684800` для первой секунды первой минуты первого часа первого дня 2000 года:

```
echo mktime(0, 0, 0, 1, 1, 2000);
```

Этой функции передаются следующие параметры (слева направо):

- количество часов (0–23);
- количество минут (0–59);
- количество секунд (0–59);
- номер месяца (1–12);
- номер дня (1–31);
- год (1970–2038, или 1901–2038 при использовании PHP 5.1.0 + 32-разрядной системы со знаком числа).

Вы можете спросить: почему годы ограничены отрезком с 1970-го до 2038-го? Причина в том, что разработчики первой версии UNIX выбрали 1970 год в качестве начала отсчета времени, опускаться ниже которого не понадобится ни одному программисту.

К счастью, благодаря тому что PHP, начиная с версии 5.1.0, поддерживает системы, использующие 32-разрядные целые числа со знаком, в нем разрешается применение дат от 1901 и до 2038 года. Но второе ограничение еще хуже первого и обусловлено тем, что разработчики UNIX также решили, что по прошествии 70 лет никто уже не будет пользоваться их системой, и поэтому они были уверены, что для хранения отметки времени им вполне хватит 32-разрядного значения, которое будет вмещать даты только до 19 января 2038 года.

Это ограничение вызовет сбой, известный как Y2K38 (очень похожий на проблему 2000 года, которая была вызвана тем, что года хранились в виде значений из двух цифр, и также требовала своего решения). Для решения этой проблемы в PHP версии 5.2 был введен класс DateTime, но он работает только в 64-разрядной архитектуре, на основе которой в наше время построено большинство компьютеров (но перед использованием этого класса нужно все же убедиться в наличии такой архитектуры).

Для отображения даты используется функция `date`, поддерживающая множество настроек форматирования, которые позволяют выводить дату любым желаемым способом. Эта функция имеет следующий синтаксис:

```
date($format, $timestamp);
```

Параметр `$format` должен быть строкой, в которой содержатся спецификаторы форматирования, подробно описанные в табл. 7.4, а параметр `$timestamp` должен быть отметкой времени в стандарте UNIX. Полный перечень спецификаторов приведен в документации по адресу http://php.net/manual/en/function.date.php. Следующая команда выведет текущее время и дату в формате `Thursday July 6th, 2017 – 1:38pm`:

```
echo date("l F jS, Y - g:ia", time0);
```

Таблица 7.4. Основные спецификаторы формата, использующиеся в функции date

Формат	Описание	Возвращаемое значение
Спецификаторы дня		
d	День месяца, две цифры с лидирующими нулями	От 01 до 31
D	День недели, составленный из трех букв	От Mon до Sun
j	День месяца без лидирующих нулей	От 1 до 31
l	День недели полностью	От Sunday до Saturday
N	День недели, число, от понедельника до воскресенья	От 1 до 7
S	Суффикс для дня месяца (пригодится в сочетании со спецификатором j)	st, nd, rd или th
w	День недели, число, от воскресенья до субботы	От 0 до 6
z	День года	От 0 до 365
Спецификатор недели		
W	Номер недели в году	От 01 до 52
Спецификаторы месяца		
F	Название месяца	От January до December
m	Номер месяца с лидирующими нулями	От 01 до 12
M	Название месяца, составленное из трех букв	От Jan до Dec
n	Номер месяца без лидирующих нулей	От 1 до 12
t	Количество дней в заданном месяце	28, 29, 30 или 31
Спецификаторы года		
L	Високосный год	1 — Да, 0 — Нет
Y	Год, четыре цифры	От 0000 до 9999
y	Год, две цифры	От 00 до 99
Спецификаторы времени		
a	До или после полудня, в нижнем регистре	am или pm
A	До или после полудня, в верхнем регистре	AM или PM
g	Час суток, 12-часовой формат без лидирующих нулей	От 1 до 12
G	Час суток, 24-часовой формат без лидирующих нулей	От 00 до 23
h	Час суток, 12-часовой формат с лидирующими нулями	От 01 до 12
H	Час суток, 24-часовой формат с лидирующими нулями	От 00 до 23
i	Минуты с лидирующими нулями	От 00 до 59
s	Секунды с лидирующими нулями	От 00 до 59

Константы, связанные с датами

Существуют полезные константы, которые можно использовать с командами, связанными с датами, для того чтобы они вернули дату в определенном формате. Например, `date(DATE_RSS)` возвращает текущую дату и время в формате, который применяется в RSS-потоке. Наиболее часто используются следующие константы.

- `DATE_ATOM` — формат для потоков Atom. PHP-формат имеет вид `Y-m-d\TH:i:sP`, а выводимая информация — `2022-10-22T 12:00:00+0000`.
- `DATE_COOKIE` — формат для cookie, устанавливаемый веб-сервером или JavaScript. PHP-формат имеет вид `l, d-M-y H:i:s T`, а выводимая информация — `Wednesday, 26-Oct-22 12:00:00 UTC`.
- `DATE_RSS` — формат для потоков RSS. PHP-формат имеет вид `D, d M Y H:i:s T`, а выводимая информация — `Wed,216 Oct 2022 12:00:00 UTC`.
- `DATE_W3C` — формат для Консорциума Всемирной паутины, World Wide Web Consortium. PHP-формат имеет вид `Y-m-d\TH:i:sP`, а выводимая информация — `2022-10-26T12:00:00+00:00`.

Полный перечень приведен по адресу http://php.net/manual/en/class.datetime.php.

Функция checkdate

Как отобразить допустимую дату в различных форматах, вы уже видели. А как проверить, что пользователь передал такую дату вашей программе? Нужно передать месяц, день и год функции `checkdate`, которая вернет значение `TRUE`, если ей передана допустимая дата, или значение `FALSE` в противном случае.

Например, если введена дата 31 сентября любого года, то она всегда будет недопустимой. В примере 7.3 показан код, который можно использовать для этой цели. В данных условиях он признает указанную дату недопустимой.

Пример 7.3. Проверка допустимости даты

```
<?php
  $month = 9;     // Сентябрь (в котором только 30 дней)
  $day   = 31;    // 31-е
  $year  = 2022;  // 2022

  if (checkdate($month, $day, $year)) echo "Допустимая дата";
  else echo "Недопустимая дата";
?>
```

Работа с файлами

При всех своих достоинствах MySQL не является единственным (или самым лучшим) способом хранения всех данных на веб-сервере. Иногда бывает быстрее и удобнее обращаться непосредственно к файлам, хранящимся на диске.

Это может потребоваться при изменении изображений, например выложенных пользователями аватаров, или файлов регистрационных журналов, требующих обработки.

Прежде всего следует упомянуть об именах файлов. Если создается код, который может использоваться на различных установках PHP, то узнать о том, чувствительна система к регистру букв или нет, практически невозможно. Например, имена файлов в Windows и macOS нечувствительны к регистру, а в Linux и UNIX — чувствительны. Поэтому нужно принять за основу то, что система чувствительна к регистру, и придерживаться соглашения о присваивании файлам имен в нижнем регистре.

Проверка существования файла

Чтобы проверить факт существования файла, можно воспользоваться функцией `file_exists`, которая возвращает либо `TRUE`, либо `FALSE` и применяется следующим образом:

```
if (file_exists("testfile.txt")) echo "Файл существует";
```

Создание файла

В данный момент файла `testfile.txt` не существует, поэтому создадим его и запишем в него несколько строк. Наберите код, показанный в примере 7.4, и сохраните его под именем `testfile.php`.

Пример 7.4. Создание простого текстового файла

```
<?php // testfile.php
  $fh = fopen("testfile.txt", 'w') or die("Создать файл не удалось");

  $text = <<<_END
Строка 1
Строка 2
Строка 3
_END;

  fwrite($fh, $text) or die("Сбой записи файла");
  fclose($fh);
  echo "Файл 'testfile.txt' записан успешно";
?>
```

Если в программе выполняется вызов функции `die`, открытый файл автоматически закрывается в рамках общего завершения программы.

Если этот код будет запущен через браузер, то при его успешном выполнении появится следующее сообщение: `Файл 'testfile.txt' записан успешно`. Если будет выведено сообщение об ошибке, значит на диске недостаточно свободного места или, что более вероятно, отсутствует разрешение на создание файла или на запись в этот файл. В таком случае нужно изменить атрибуты папки назначения в соответствии

с требованиями вашей операционной системы. Если все обойдется без ошибки, то файл `testfile.txt` попадет в ту же папку, где был сохранен программный файл `testfile.php`. Если открыть файл в текстовом или программном редакторе, в нем будет следующее содержимое:

```
Строка 1
Строка 2
Строка 3
```

В этом простом примере показана последовательность работы со всеми файлами.

1. Все начинается с открытия файла с помощью вызова функции `fopen`.
2. После этого можно вызывать другие функции. В данном случае в файл велась запись (`fwrite`), но можно также читать данные из уже существующего файла (`fread` или `fgets`) и осуществлять с ним другие действия.
3. Работа завершается закрытием файла (`fclose`). Хотя программа перед завершением своей работы делает это за вас, но все же вы должны удостовериться в том, что по окончании работы с файлом он будет закрыт.

Каждому открытому файлу требуется файловый ресурс, чтобы PHP-программа могла к нему обращаться и им управлять. В предыдущем примере переменной `$fh` (которую я выбрал в качестве описателя файла) присваивается значение, возвращаемое функцией `fopen`. После этого каждой функции обработки файла, которая получает к нему доступ, например `fwrite` или `fclose`, в качестве параметра должна быть передана переменная `$fh`, чтобы идентифицировать обрабатываемый файл. Интересоваться содержимым переменной `$fh` не стоит, это всего лишь номер, используемый PHP для ссылки на внутреннюю информацию о файле. Данная переменная применяется только для передачи другим функциям.

В случае сбоя функция `fopen` возвращает значение `FALSE`. В предыдущем примере показан простой способ перехвата управления и реакции на сбой: в нем вызывается функция `die`, которая завершает программу и выдает пользователю сообщение об ошибке. Этот упрощенный способ выхода подходит лишь для наших учебных программ, а выходить с его помощью из веб-приложения не следует ни в коем случае (вместо этого нужно создать веб-страницу с сообщением об ошибке).

Обратите внимание на второй параметр, используемый в вызове функции `fopen`. Это символ `w`, предписывающий функции открыть файл для записи. Если такого файла нет, то он будет создан. Применять эту функцию следует с оглядкой: если файл уже существует, то параметр режима работы `w` заставит функцию `fopen` удалить все его прежнее содержимое (даже если в него не будет записано ничего нового!).

В табл. 7.5 перечислены различные параметры режима работы, которые могут быть использованы при вызове этой функции. Режимы, включающие символ `+`, будут рассмотрены далее в разделе «Обновление файлов».

Таблица 7.5. Режимы работы, поддерживаемые функцией fopen

Режим	Действие	Описание
'r'	Чтение с начала файла	Открытие файла только для чтения; установка указателя файла на его начало. Возвращение FALSE, если файла не существует
'r+'	Чтение с начала файла с возможностью записи	Открытие файла для чтения и записи; установка указателя файла на его начало. Возвращение FALSE, если файла не существует
'w'	Запись с начала файла с усечением его размера	Открытие файла только для записи; установка указателя файла на его начало и сокращение размера файла до нуля. Если файла не существует, попытка его создания
'w+'	Запись с начала файла с усечением его размера и возможностью чтения	Открытие файла для чтения и записи; установка указателя файла на его начало и сокращение его размера до нуля. Если файла не существует, попытка его создания
'a'	Добавление к концу файла	Открытие файла только для записи; установка указателя файла на его конец. Если файла не существует, попытка его создания
'a+'	Добавление к концу файла с возможностью чтения	Открытие файла для чтения и записи; установка указателя файла на его конец. Если файла не существует, попытка его создания

Чтение из файлов

Проще всего прочитать текстовый файл, извлекая из него всю строку целиком, для чего, как в примере 7.5, используется функция `fgets` (последняя буква s в названии функции означает string — «строка»).

Пример 7.5. Чтение файла с помощью функции fgets

```
<?php
  $fh = fopen("testfile.txt", 'r') or
    die("Файл не существует, или вы не обладаете правами на его открытие");

  $line = fgets($fh);
  fclose($fh);
  echo $line;
?>
```

Если используется файл, созданный кодом из примера 7.4, будет получена первая строка:

```
Строка 1
```

Можно также извлечь из файла сразу несколько строк или фрагменты строк, воспользовавшись функцией `fread`, как показано в примере 7.6.

Пример 7.6. Чтение файла с помощью функции fread

```
<?php
  $fh = fopen("testfile.txt", 'r') or
    die("Файл не существует, или вы не обладаете правами на его открытие");

  $text = fread($fh, 3);
  fclose($fh);
  echo $text;
?>
```

При вызове функции `fread` было запрошено чтение трех символов, поэтому программа отобразит следующий текст:

```
Стр
```

Функция `fread` обычно применяется для чтения двоичных данных. Но если она используется для чтения текстовых данных объемом более одной строки, следует брать в расчет символы новой строки.

Копирование файлов

Попробуем создать клон нашего файла `testfile.txt`, воспользовавшись PHP-функцией `copy`. Наберите текст примера 7.7 и сохраните его в файле `copyfile.php`, а затем вызовите программу через браузер.

Пример 7.7. Копирование файла

```
<?php // copyfile.php
  copy('testfile.txt', 'testfile2.txt') or die("Копирование невозможно");
  echo "Файл успешно скопирован в 'testfile2.txt'";
?>
```

Если еще раз проверить содержимое вашей папки, в ней окажется новый файл `testfile2.txt`. Кстати, если вам не нужно, чтобы программа завершала свою работу после неудачной попытки копирования, можно воспользоваться другим вариантом синтаксиса, который показан в примере 7.8. В качестве быстрой и краткой записи в нем задействуется оператор `! (NOT)`. Его наличие перед выражением означает применение оператора `NOT` ко всему выражению, следовательно, эквивалентным утверждением становится «Если копирование невозможно...».

Пример 7.8. Альтернативный синтаксис для копирования файла

```
<?php // copyfile2.php
  if (!copy('testfile.txt', 'testfile2.txt')) echo "Копирование невозможно";
  else echo "Файл успешно скопирован в 'testfile2.txt"';
?>
```

Перемещение файла

Для перемещения файла его необходимо переименовать, как показано в примере 7.9.

Пример 7.9. Перемещение файла

```
<?php // movefile.php
  if (!rename('testfile2.txt', 'testfile2.new'))
    echo "Переименование невозможно";
  else echo "Файл успешно переименован в 'testfile2.new'";
?>
```

Функцию переименования можно применять и к каталогам. Чтобы избежать предупреждений при отсутствии исходных файлов, сначала для проверки факта их существования можно вызвать функцию `file_exists`.

Удаление файла

Для удаления файла из файловой системы достаточно, как показано в примере 7.10, воспользоваться функцией `unlink`.

Пример 7.10. Удаление файла

```
<?php // deletefile.php
  if (!unlink('testfile2.new')) echo "Удаление невозможно";
  else echo "Файл 'testfile2.new' удален успешно";
?>
```

При непосредственном доступе к файлам на жестком диске нужна гарантия того, что ваша файловая система не будет поставлена под угрозу. Например, при удалении файла на основе введенной пользователем информации нужно быть абсолютно уверенным в том, что этот файл может быть удален без ущерба безопасности системы и что пользователю разрешено удалять его.

В данном случае, как и при операции перемещения, если файла с таким именем не существует, будет выведено предупреждение. Его появления можно избежать, если использовать функцию `file_exists` для проверки существования файла перед вызовом функции `unlink`.

Обновление файлов

Довольно часто возникает потребность добавлять к сохраненному файлу дополнительные данные, для чего существует множество способов. Можно воспользоваться одним из режимов добавления данных (см. табл. 7.5) или же задействовать режим, поддерживающий запись, и просто открыть файл для чтения и записи и переместить указатель файла в то место, с которого необходимо вести запись в файл или чтение из файла.

Указатель файла — это позиция внутри файла, с которой будет осуществлен очередной доступ к файлу при чтении или записи. Его не следует путать с *описателем* файла (который в примере 7.4 хранился в переменной `$fh`), содержащим сведения о том файле, к которому осуществляется доступ.

Если набрать код, показанный в примере 7.11, сохранить его в файле `update.php`, а затем вызвать его из своего браузера, то можно увидеть работу указателя.

Пример 7.11. Обновление файла

```
<?php // update.php
  $fh = fopen("testfile.txt", 'r+') or die("Сбой открытия файла");
  $text = fgets($fh);

  fseek($fh, 0, SEEK_END);
  fwrite($fh, "$text") or die("Сбой записи в файл");
  fclose($fh);

  echo "Файл 'testfile.txt' успешно обновлен";
?>
```

Эта программа открывает файл `testfile.txt` для чтения и записи, для чего задается режим работы `'+r'`, в котором указатель устанавливается в самое начало файла. Затем используется функция `fgets`, с помощью которой из файла считывается одна строка (до встречи первого символа перевода строки). После этого вызывается функция `fseek`, чтобы переместить указатель файла в самый конец, куда затем добавляется строка, которая была извлечена из начала файла (и сохранена в переменной `$text`), после чего файл закрывается. Получившийся в итоге файл имеет следующий вид:

```
Строка 1
Строка 2
Строка 3
Строка 1
```

Первая строка была успешно скопирована, а затем добавлена в конец файла.

В данном примере функции `fseek` кроме описателя файла `$fh` были переданы еще два параметра — `0` и `SEEK_END`. Параметр `SEEK_END` предписывает функции переместить указатель файла в его конец, а параметр `0` показывает, на сколько позиций нужно вернуться назад из этой позиции. В примере 7.11 используется значение `0`, потому что указатель должен оставаться в конце файла.

С функцией `fseek` можно задействовать еще два режима установки указателя: `SEEK_SET` и `SEEK_CUR`. Режим `SEEK_SET` предписывает функции установку указателя файла на конкретную позицию, заданную предыдущим параметром. Поэтому в следующем примере указатель файла перемещается на позицию 18:

```
fseek($fh, 18, SEEK_SET);
```

Режим `SEEK_CUR` приводит к установке указателя файла на позицию, которая смещена от текущей позиции на заданное значение. Если в данный момент указатель файла находится на позиции 18, то следующий вызов функции переместит его на позицию 23:

```
fseek($fh, 5, SEEK_CUR);
```

Без особой надобности это не рекомендуется делать, но таким образом даже текстовые файлы (с фиксированной длиной строк) можно использовать в качестве простых неструктурированных баз данных. В этом случае ваша программа может использовать функцию `fseek` для перемещения в обе стороны по такому файлу с целью извлечения, обновления существующих и добавления новых записей. Записи также могут удаляться путем их перезаписи нулевыми символами и т. д.

Блокирование файлов при коллективном доступе

Веб-программы довольно часто вызываются многими пользователями в одно и то же время. Когда одновременно предпринимается попытка записи в файл более чем одним пользователем, файл может быть поврежден. А когда один пользователь вводит в него запись, а другой считывает из него данные, с файлом ничего не случится, но читающий может получить весьма странные результаты.

Чтобы обслужить сразу несколько одновременно обращающихся к файлу пользователей, нужно обратиться к функции блокировки файла `flock`. Эта функция ставит в очередь все другие запросы на доступ к файлу до тех пор, пока ваша программа не снимет блокировку. Когда ваши программы обращаются к файлу, который может быть одновременно доступен нескольким пользователям, с намерением произвести в него запись, к коду нужно также добавлять задание на блокировку файла, как в примере 7.12, который является обновленной версией примера 7.11.

Пример 7.12. Обновление файла с использованием блокировки

```
<?php
  $fh = fopen("testfile.txt", 'r+') or die("Сбой открытия файла");
  $text = fgets($fh);

  if (flock($fh, LOCK_EX))
  {
    fseek($fh, 0, SEEK_END);
    fwrite($fh, "$text") or die("Сбой записи в файл");
    flock($fh, LOCK_UN);
  }

  fclose($fh);
  echo "Файл 'testfile.txt' успешно обновлен";
?>
```

При блокировке файла для посетителей вашего сайта нужно добиться наименьшего времени отклика: блокировку следует ставить непосредственно перед внесением изменений в файл и снимать ее сразу же после их внесения. Блокировка файла на более длительный период приведет к неоправданному замедлению работы приложения. Поэтому в примере 7.12 функция `flock` вызывается непосредственно до и после вызова функции `fwrite`.

При первом вызове `flock` с помощью параметра `LOCK_EX` устанавливается эксклюзивная блокировка того файла, ссылка на который содержится в переменной `$fh`:

```
flock($fh, LOCK_EX);
```

С этого момента и далее никакой другой процесс не может осуществлять не только запись, но даже чтение файла до тех пор, пока блокировка не будет снята с помощью передачи функции параметра `LOCKJJN`:

```
flock($fh, LOCKJJN);
```

Как только блокировка будет снята, другие процессы снова получат возможность доступа к файлу. Это одна из причин, по которой необходимо заново обращаться к нужному месту в файле при каждом чтении или записи данных: со времени последнего обращения к нему другой процесс мог внести в этот файл изменения.

Кстати, вы заметили, что вызов с требованием эксклюзивной блокировки вложен в структуру инструкции `if`? Дело в том, что `flock` поддерживается не во всех системах, и поэтому есть смысл проверить успешность установки блокировки, так как известно, что некоторые системы на это не способны.

Следует также принять во внимание, что действия функции `flock` относятся к так называемой *рекомендательной* блокировке. Это означает, что блокируются только те процессы, которые вызывают эту функцию. Если есть код, который действует напрямую и изменяет файлы, не блокируя их с помощью `flock`, он всегда сможет обойти блокировку и внести хаос в ваши файлы.

Кстати, если в каком-то кодовом фрагменте заблокировать файл, а затем по рассеянности забыть его разблокировать, это может привести к ошибке, которую будет очень трудно обнаружить.

Функция flock не будет работать в сетевой файловой системе NFS и во многих других файловых системах, основанных на применении сетей. Не стоит полагаться на flock и при использовании многопоточных серверов типа ISAPI, потому что она не защитит файлы от доступа из кода PHP-сценариев, запущенных в параллельных потоках на том же физическом сервере. Кроме того, flock не поддерживается в любых системах, использующих устаревшую файловую систему FAT, например в устаревших версиях Windows.

Если на этот счет есть сомнения, можно попробовать установить в начале программы быструю блокировку на тестовый файл, чтобы посмотреть, можно ли получить блокировку на файл. Не забудьте после проверки снять с него блокировку (и, возможно, удалить его, если он больше не нужен).

Не следует также забывать, что любой вызов функции die приводит к автоматическому снятию блокировки и закрытию файла в рамках общего завершения программы.

Чтение всего файла целиком

Для чтения целиком всего файла без применения описателей файлов можно воспользоваться очень удобной функцией `file_get_contents`. Она проста в применении, о чем свидетельствует код примера 7.13.

Пример 7.13. Использование функции file_get_contents

```
<?php
   echo "<pre>";   // Тег, позволяющий отображать переводы строк
   echo file_get_contents("testfile.txt");
   echo "</pre>";  // Прекращение действия тега <pre>
?>
```

Но эту функцию можно использовать и с большей пользой. С ее помощью можно извлечь файл с сервера через Интернет. В примере 7.14 показан запрос кода HTML с главной страницы сайта O'Reilly с последующим ее отображением, как при обычном переходе на саму веб-страницу. Полученный результат будет похож на копию страницы, приведенную на рис. 7.1.

Пример 7.14. Захват главной страницы сайта O'Reilly

```
<?php
  echo file_get_contents("http://oreilly.com");
?>
```

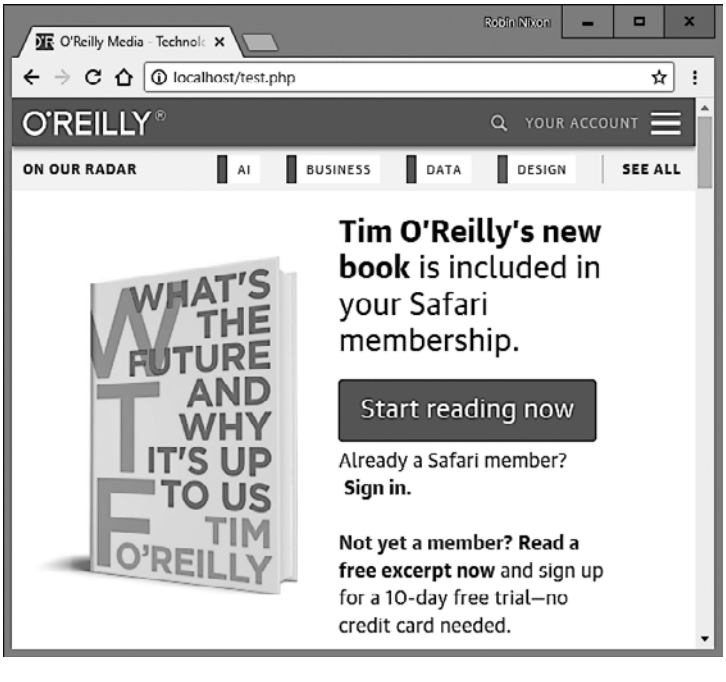

Рис. 7.1. Главная страница сайта O'Reilly, захваченная с помощью функции file_get_contents

Загрузка файлов на веб-сервер

Загрузка файлов на веб-сервер вызывает затруднения у многих пользователей, но сделать этот процесс еще проще, чем он есть на самом деле, не представляется возможным. Для загрузки файла из формы нужно лишь выбрать специальный тип кодировки, который называется `multipart/form-data`, а все остальное сделает ваш браузер. Чтобы увидеть этот процесс в работе, наберите программу, представленную в примере 7.15, и сохраните ее в файле под именем `upload.php`. Когда этот файл будет запущен, в браузере появится форма, позволяющая загружать на сервер любой выбранный файл.

Пример 7.15. Программа для загрузки изображений, хранящаяся в файле upload.php

```
<?php // upload.php
  echo <<<_END
    <html><head><title>PHP-форма для загрузки файлов
    на сервер</title></head><body> <form method='post'
    action='upload.php' enctype='multipart/form-data'>
    Выберите файл: <input type='file' name='filename' size='10'>
    <input type='submit' value='Загрузить'>
    </form>
_END;

  if ($_FILES)
  {
  $name = $_FILES['filename']['name'];
  move_uploaded_file($_FILES['filename']['tmp_name'], $name);
  echo "Загружаемое изображение '$name'<br><img src='$name'>";
}

  echo "</body></html>";
?>
```

Проанализируем программу по блокам. В первой строке многострочной инструкции `echo` задается начало HTML-документа, отображается заголовок, а затем начинается тело документа.

Далее идет форма, для передачи содержимого которой выбран метод POST, задается предназначение всех отправляемых программе `upload.php` (то есть самой нашей программе) данных и указывается браузеру на то, что отправляемые данные должны быть закодированы с использованием типа содержимого `multipart/form-data`.

Для подготовки формы в следующих строках задается отображение приглашения `Выберите файл`, а затем дважды запрашивается пользовательский ввод. Сначала от пользователя требуется указать файл. В параметрах ввода задаются тип вводимой информации — `input type`, в качестве которого указан файл — `file`, имя — `name`, в качестве которого определено имя файла — `filename`, а также размер поля ввода — `size`, в качестве которого указана ширина поля, составляющая 10 символов.

Затем от пользователя требуется ввести команду на отправку данных формы, для чего служит кнопка с надписью `Загрузить` (эта надпись заменяет текст, исполь-

зуемый по умолчанию, — `Submit Query`, что означает «Отправить запрос»). После этого форма закрывается.

В этой небольшой программе показана весьма распространенная технология веб-программирования, в которой одна и та же программа вызывается дважды: один раз при первом посещении страницы, а второй — когда пользователь нажимает кнопку отправки формы.

PHP-код, предназначенный для приема загружаемых данных, предельно прост, поскольку все загружаемые на сервер файлы помещаются в ассоциативный системный массив `$_FILES`. Поэтому для установки факта отправки пользователем файла достаточно проверить, есть ли у массива `$_FILES` хоть какое-нибудь содержимое. Эта проверка осуществляется с помощью инструкции `if ($_FILES)`.

При первом посещении страницы пользователем, которое происходит еще до загрузки файла, массив `$_FILES` пуст, поэтому программа пропускает этот блок кода. Когда пользователь загружает файл, программа запускается еще раз и обнаруживает присутствие элемента в массиве `$_FILES`.

Когда программа обнаружит, что файл был загружен, его имя, каким оно было прочитано из компьютера, занимавшегося загрузкой, извлекается и помещается в переменную `$name`. Теперь нужно только переместить файл из временного места, где PHP хранит загруженные файлы, в постоянное место хранения. Это делается с помощью функции `move_uploaded_file`, которой передается исходное имя файла, сохраняемого в текущем каталоге.

И наконец, загруженное на сервер изображение отображается путем помещения его имени в тег `IMG`. Возможный результат показан на рис. 7.2.

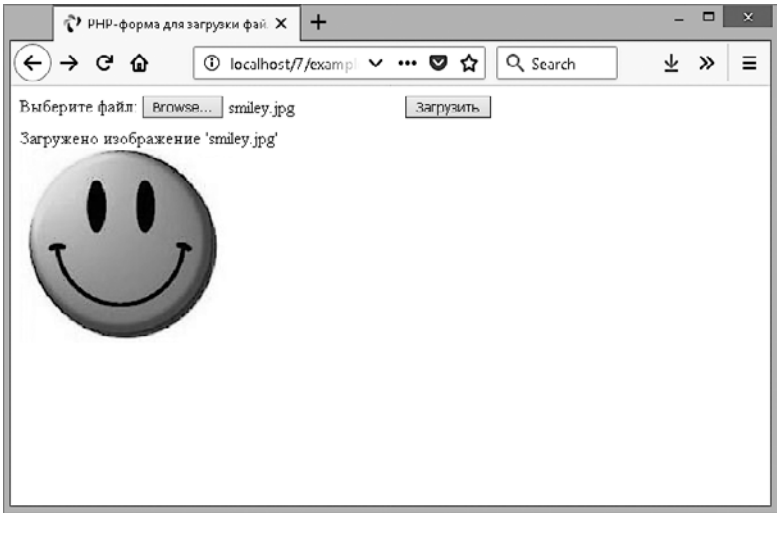

Рис. 7.2. Загрузка изображения с помощью формы данных

Если при запуске программы в ответ на вызов функции move_uploaded_file будет получено предупреждение об отсутствии прав доступа — Permission denied, значит у вас нет права на доступ к папке, из которой запущена программа.

Использование массива $_FILES

При загрузке файла на сервер в массиве `$_FILES` сохраняются пять элементов, показанных в табл. 7.6 (где используется загружаемый файл, имя которого предоставляется отправляемой серверу формой).

Таблица 7.6. Содержимое массива $_FILES

Элемент массива	Содержимое
$_FILES[*'файл'*][*'имя'*]	Имя загруженного файла (например, smiley.jpg)
$_FILES[*'файл'*][*'тип'*]	Тип содержимого файла (например, image/jpeg)
$_FILES[*'файл'*][*'размер'*]	Размер файла в байтах
$_FILES[*'файл'*]['*временное_имя*']	Имя временного файла, сохраненного на сервере
$_FILES[*'файл'*][*'ошибка'*]	Код ошибки, получаемый после загрузки файла

Типы содержимого обычно называли *MIME-типами* (Multipurpose Internet Mail Extension — многоцелевые почтовые расширения в Интернете). Но поскольку позже они были распространены на все виды передаваемой через Интернет информации, то теперь их часто называют *типами информации, используемой в Интернете* (Internet Media Types). В табл. 7.7 показаны некоторые из наиболее часто используемых типов, появляющиеся в элементе массива `$_FILES['файл']['тип']`.

Таблица 7.7. Некоторые наиболее распространенные типы информации, используемой в Интернете

application/pdf	image/gif	multipart/form-data	text/xml
application/zip	image/jpeg	text/css	video/mpeg
audio/mpeg	image/png	text/html	video/mp4
audio/x-wav	image/tiff	text/plain	video/quicktime

Проверка допустимости

Надеюсь, что не нужно говорить (хотя я все равно это сделаю), насколько важно проверять допустимость присланных формой данных, что обусловлено существующей для пользователей возможностью взломать ваш сервер.

Вдобавок к проверке вредоносности введенных данных нужно также проверить, был ли файл получен, и если он получен, то был ли отправлен правильный тип данных.

С учетом всего этого программа `upload.php` была превращена в программу `upload2.php`, показанную в примере 7.16.

Пример 7.16. Более безопасная версия upload.php

```
<?php // upload2.php
  echo <<<_END
    <html><head><title>PHP-форма для загрузки файлов
    на сервер</title></head><body>
    <form method='post' action='upload2.php' enctype='multipart/form-data'>
    Выберите файл с расширением JPG, GIF, PNG или TIF:
    <input type='file' name='filename' size='10'>
    <input type='submit' value='Загрузить'></form>
_END;

  if ($_FILES)
  {
    $name = $_FILES['filename']['name'];

    switch($_FILES['filename']['type'])
    {
      case 'image/jpeg': $ext = 'jpg'; break;
      case 'image/gif':  $ext = 'gif'; break;
      case 'image/png':  $ext = 'png'; break;
      case 'image/tiff': $ext = 'tif'; break;
      default:           $ext = '';    break;
    }
    if ($ext)
    {
      $n = "image.$ext";
      move_uploaded_file($_FILES['filename']['tmp_name'], $n);
      echo "Загружено изображение '$name' под именем '$n':<br>";
      echo "<img src='$n'>";
    }
    else echo "'$name' - неприемлемый файл изображения";
  }
  else echo "Загрузки изображения не произошло";

  echo "</body></html>";
?>
```

Блок, не содержащий HTML-кода, был расширен, и теперь вместо шести строк примера 7.15 в нем содержится 20 строк, начиная с `if ($_FILES)`.

Как и в предыдущей версии, в этой строке `if` выполняется проверка факта отправки данных, но теперь у этой инструкции ближе к концу программы есть и соответствующая ей инструкция `else`, которая выводит на экран сообщение о том, что загрузки изображения не произошло.

В теле инструкции `if` переменной `$name` присваивается значение имени файла, полученное (как и прежде) от загружающего компьютера, но на этот раз мы не полагаемся на то, что пользователь отправил нам приемлемые данные. Вместо этого используется инструкция `switch`, предназначенная для проверки соответствия типа загружаемого контекста четырем типам изображений, которые поддерживаются этой программой. При обнаружении соответствия переменной `$ext` присваивается трехсимвольное расширение имени файла, относящееся к этому типу. Если соответствие не обнаружится, значит загруженный файл не относится к приемлемому типу и переменной `$ext` будет присвоена пустая строка `""`.

В следующем блоке кода проверяется, содержит ли переменная `$ext` строку, и при положительном ответе в переменной `$n` создается новое имя файла, составленное из основы `image` и расширения, сохраненного в переменной `$ext`. Это означает, что программа полностью контролирует имя создаваемого файла и этим именем может быть только одно из следующих: `image.jpg`, `image.gif`, `image.png` или `image.tif`.

Поскольку программе больше ничто не угрожает, остальной PHP-код похож на код предыдущей версии. Он перемещает загруженное временное изображение на его новое место, затем выводит его на экран, а вместе с ним отображает старое и новое имена изображения.

Об удалении временного файла, созданного PHP в процессе загрузки, беспокоиться не стоит, поскольку, если файл не был перемещен или переименован, он будет удален автоматически, как только программа завершит свою работу.

Когда по условию инструкции `if` произойдет переход к инструкции `else`, которая выполняется только в том случае, если загружен неподдерживаемый тип изображения, программа выведет сообщение об ошибке.

Я настоятельно рекомендую применить такой же подход и использовать заранее подобранные имена и места для загружаемых файлов, когда вы будете создавать собственную программу загрузки. Тогда будут исключены любые попытки добавления к используемым переменным каких-нибудь других путевых имен и других данных, способных нанести вред. Если подразумевается, что несколько пользователей могут загружать файл с одним и тем же именем, то такие файлы можно снабжать префиксами, представляющими собой имена пользователей, или сохранять их в отдельных папках, созданных для каждого пользователя.

Но если нужно использовать предоставленное имя файла, его требуется обезвредить, разрешив применение только буквенно-цифровых символов и точки, что можно сделать с помощью следующей команды, использующей регулярное выражение (см. главу 17) для осуществления поиска и замены символов в значении переменной `$name`:

```
$name = preg_replace("[^A-Za-z0-9.]/", "", $name);
```

Эта команда оставляет в строковой переменной `$name` только символы A–Z, a–z, 0–9 и точку, а прочие символы удаляет.

Для обеспечения работы своей программы во всех системах, независимо от их чувствительности к регистру букв, стоит воспользоваться другой командой, которая одновременно с предыдущими действиями переводит все символы верхнего регистра в нижний:

```
$name = strtolower(ereg_replace("[^A-Za-z0-9.]", "", $name));
```

Иногда можно встретить тип содержимого image/pjpeg, который служит признаком прогрессивного JPEG-формата. Этот тип можно без лишних опасений добавить к вашему коду в качестве альтернативы для image/jpeg:

```
case 'image/pjpeg':
case 'image/jpeg': $ext = 'jpg'; break;
```

Системные вызовы

Иногда функцию для осуществления конкретного действия можно найти не в PHP, а в операционной системе, под управлением которой запущен этот язык. В таком случае для выполнения задачи можно применить системный вызов `exec`.

Например, для быстрого просмотра содержимого текущего каталога можно воспользоваться программой, показанной в примере 7.17. В процессе работы в системе Windows она не потребует изменений и задействует Windows-команду `dir`. В Linux, UNIX или macOS нужно будет закомментировать или удалить первую строку и убрать символы комментария из второй строки, чтобы применить системную команду `ls`. При желании можете набрать текст этой программы, сохранить его как `exec.php` и вызвать из своего браузера.

Пример 7.17. Выполнение системной команды

```
<?php // exec.php
  $cmd = "dir";   // Windows
  // $cmd = "ls"; // Linux, Unix & Mac

  exec(escapeshellcmd($cmd), $output, $status);

  if ($status) echo "Команда exec не выполнена";
  else
  {
     echo "<pre>";
     foreach($output as $line) echo htmlspecialchars("$line\n");
     echo "</pre>";
  }
?>
```

Функция `htmlspecialchars` вызывается с целью превращения любых специальных символов, возвращаемых системой, в символы, которые могут быть восприняты и правильно отображены как код HTML, упорядочивая тем самым вывод.

В зависимости от рабочей системы в результате запуска этой программы будет выведена следующая информация (полученная при использовании Windows-команды `dir`):

```
Volume in drive C is Hard Disk
  Volume Serial Number is DC63-0E29

  Directory of C:\Program Files (x86)\Ampps\www

11/04/2018 11:58   <DIR>        .
11/04/2018 11:58   <DIR>        ..
28/01/2018 16:45   <DIR>        5th_edition_examples
08/01/2018 10:34   <DIR>        cgi-bin
08/01/2018 10:34   <DIR>        error
29/01/2018 16:18           1,150 favicon.ico
             1 File(s)      1,150 bytes
             5 Dir(s)   1,611,387,486,208 bytes free
```

Функция `exec` воспринимает три аргумента.

- Саму команду (в предыдущем случае это `$cmd`).
- Массив, в который система поместит информацию, получаемую в результате выполнения команды (в предыдущем случае это `$output`).
- Переменную для хранения возвращаемого статуса вызова (в предыдущем случае это `$status`).

При желании параметры `$output` и `$status` можно опустить, но тогда ничего не будет известно ни о выходной информации, созданной в результате вызова, ни даже о том, насколько успешен был сам вызов.

Обратите внимание также на применение функции `escapeshellcmd`. Желательно выработать привычку постоянно использовать эту функцию при вызове функции `exec`, поскольку она обезвреживает содержимое командной строки, предотвращая выполнение случайных команд в том случае, если пользователю предоставляется возможность их ввода.

Как правило, функции системных вызовов на веб-хостах общего пользования запрещены как представляющие угрозу системе безопасности. По возможности все задачи нужно стараться решить средствами PHP и обращаться к системе напрямую только при крайней необходимости. Кроме того, вы должны знать, что обращение к системе выполняется довольно медленно, и, если приложение рассчитано на запуск как в Windows, так и в Linux/UNIX, для него следует создавать две реализации вызова.

XHTML или HTML5?

Поскольку документы XHTML должны иметь строго заданное оформление, их парсинг может проводиться с использованием стандартных XML-парсеров, в отличие от документов HTML, для которых требуется менее привередливый, специально приспособленный под HTML парсер (в качестве которого, на нашу удачу, работают все наиболее популярные браузеры). В итоге XHTML так и не завоевал популярности, и, когда настало время разработать новый стандарт, W3C отдал предпочтение не новому стандарту XHTML2, а поддержке HTML5.

HTML5 обладает множеством свойств как HTML4, так и XHTML, но при этом он намного проще в использовании и менее строг к проверке, и, к нашему удовольствию, теперь имеется только один тип документа, объявляемый в заголовке документа HTML5 (вместо ранее требуемых разнообразных строгих, переходных и кадрированных типов):

```
<!DOCTYPE html>
```

Простого слова `html` достаточно, чтобы сообщить браузеру, что ваша веб-страница разработана для HTML5, и, поскольку все самые последние версии наиболее популярных браузеров, начиная приблизительно с 2011 года, поддерживают большинство HTML5-спецификаций, этот тип документа, как правило, является единственно необходимым, если, конечно, не сделать выбор в пользу обслуживания устаревших браузеров.

Для всех целей и намерений при написании HTML-документов веб-разработчики могут спокойно игнорировать старые типы и синтаксис XHTML-документов (например, использование `<br />` вместо простого тега `<br>`). Но если придется обслуживать очень старые браузеры или какое-нибудь необычное приложение, основанное на XHTML, то информацию о том, как это сделать, можно найти по адресу http://xhtml.com.

Вопросы

1. Какой спецификатор преобразования следует использовать в функции `printf` для отображения числа с плавающей точкой?
2. Какая инструкция `printf` может быть использована для приема строки `"Happy Birthday"` и вывода строки `"**Happy"`?
3. Какой альтернативной функцией следует воспользоваться для выдачи информации из `printf` не в браузер, а в переменную?
4. Как создать отметку времени UNIX для времени и даты, представленных в виде `7:11am May 2nd, 2016`?

5. Какой режим доступа к файлу следует использовать в функции `fopen`, чтобы открыть файл в режиме чтения и записи с усечением его размера и установкой указателя на начало файла?
6. Какую PHP-команду нужно применить для удаления файла `file.txt`?
7. Какая PHP-функция используется для чтения целиком всего файла и даже для извлечения его из Всемирной паутины?
8. В какой суперглобальной переменной PHP содержатся сведения о загруженных на сервер файлах?
9. Какая PHP-функция позволяет запускать системные команды?
10. Какой из следующих стилей тегов в HTML5 предпочтительнее: `<hr>` или `<hr />`?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

8 Введение в MySQL

Более чем 10 млн установленных на компьютерах копий MySQL позволяют, наверное, считать ее наиболее популярной системой управления базами данных для веб-серверов. Она была разработана в середине 1990-х годов и теперь превратилась в полноценную технологию, входящую в состав многих современных наиболее посещаемых интернет-ресурсов.

Одна из причин такого успеха — то, что она, как и PHP, является продуктом свободного пользования. Кроме того, это очень мощная и исключительно быстрая система, способная работать даже на самом скромном оборудовании, не отнимая слишком много системных ресурсов.

Система MySQL обладает также хорошей масштабируемостью, то есть база данных может увеличиваться в объеме вместе с вашим сайтом (последние сравнительные показатели можно посмотреть по адресу http://mysql.com/why-mysql/benchmarks).

Основные характеристики MySQL

База данных — это структурированная коллекция записей или данных, хранящихся в компьютерной системе и организованных так, что можно осуществлять быстрый поиск и извлечение нужной информации.

В названии MySQL составляющая SQL означает *Structured Query Language — язык структурированных запросов*. Если характеризовать его в общих чертах, то это язык, основанный на словах английского языка и используемый также в других системах управления базами данных, например Oracle и Microsoft SQL Server. Он разработан для предоставления возможности создания простых запросов к базе данных посредством команд следующего вида:

```
SELECT title FROM publications WHERE author = 'Charles Dickens';
```

В базе данных MySQL имеются одна или несколько *таблиц*, каждая из которых состоит из *записей*, или *строк*. Внутри строк находятся разные *столбцы* или *поля*, в которых и содержатся данные. В табл. 8.1 показана учебная база данных, в которой присутствует информация о пяти книгах, структурированная по авторам, названиям, категориям и годам изданий.

Таблица 8.1. Пример простой базы данных

Author (автор)	Title (название)	Type (категория)	Year (год)
Mark Twain (Марк Твен)	The Adventures of Tom Sawyer («Приключения Тома Сойера»)	Fiction (Художественная)	1876
Jane Austen (Джейн Остин)	Pride and Prejudice («Гордость и предубеждение»)	Fiction (Художественная)	1811
Charles Darwin (Чарльз Дарвин)	The Origin of Species («Происхождение видов»)	Non-Fiction (Научная)	1856
Charles Dickens (Чарльз Диккенс)	The Old Curiosity Shop («Лавка древностей»)	Fiction (Художественная)	1841
William Shakespeare (Вильям Шекспир)	Romeo and Juliet («Ромео и Джульетта»)	Play (Пьеса)	1594

Каждая строка таблицы подобна строке в таблице MySQL, столбец в таблице соответствует столбцу в MySQL, и каждый элемент в строке подобен полю MySQL.

Чтобы однозначно идентифицировать эту базу данных, в последующих примерах я буду ссылаться на нее как на базу данных `publications` (издания). Как вы уже заметили, все эти издания относятся к классической литературе, поэтому таблицу в базе данных, содержащую сведения о них, я буду называть `classics`.

Сводка понятий, используемых в базах данных

Основными понятиями, с которыми следует ознакомиться на данном этапе, являются:

- *база данных* — контейнер для всей коллекции данных MySQL;
- *таблица* — вложенный в базу данных контейнер, в котором хранятся сами данные;
- *строка* — отдельная запись, в которой могут содержаться несколько полей;
- *столбец* — имя поля внутри строки.

Следует заметить, что я не пытаюсь воспроизвести точную терминологию, используемую в учебной литературе по реляционным базам данных, а хочу лишь дать простые, обычные определения, помогающие быстро усвоить основные понятия и приступить к работе с базой данных.

Доступ к MySQL из командной строки

Работать с MySQL можно тремя основными способами: используя командную строку, применяя веб-интерфейс наподобие phpMyAdmin и задействуя такой язык программирования, как PHP. Третий из перечисленных способов будет рассмотрен в главе 10, а сейчас изучим первые два способа.

Начало работы с интерфейсом командной строки

В следующих подразделах даны соответствующие инструкции для Windows, macOS и Linux.

Для пользователей Windows

Если у вас в соответствии с инструкциями, изложенными в главе 2, обычным способом установлена среда AMPPS, то доступ к исполняемой программе MySQL можно получить из следующего каталога:

```
C:\Program Files (x86)\Ampps\mysql\bin
```

Если AMPPS установлена в любое другое место, то нужно воспользоваться тем каталогом, в котором она установлена.

По умолчанию начальным для MySQL будет пользователь по имени `root`, у которого будет исходный пароль `mysql`.

Чтобы войти в интерфейс командной строки MySQL, следует выбрать команду Пуск ▸ Выполнить и в окне запуска ввести команду `CMD`, после чего нажать клавишу Enter. В результате будет вызвано командное окно Windows. Находясь в этом окне, нужно ввести следующую команду (внося в нее соответствующие коррективы):

```
cd C:\"Program Files (x86)\Ampps\mysql\bin"
mysql -u root -pmysql
```

Первая команда меняет текущий каталог на каталог MySQL, а вторая предписывает MySQL зарегистрировать вас как пользователя `root` с паролем `mysql`. Теперь вы должны оказаться в среде MySQL и сможете приступить к вводу команд.

Чтобы убедиться в том, что все работает должным образом, введите следующую команду, результат выполнения которой должен быть похож на показанный на рис. 8.1:

```
SHOW databases;
```

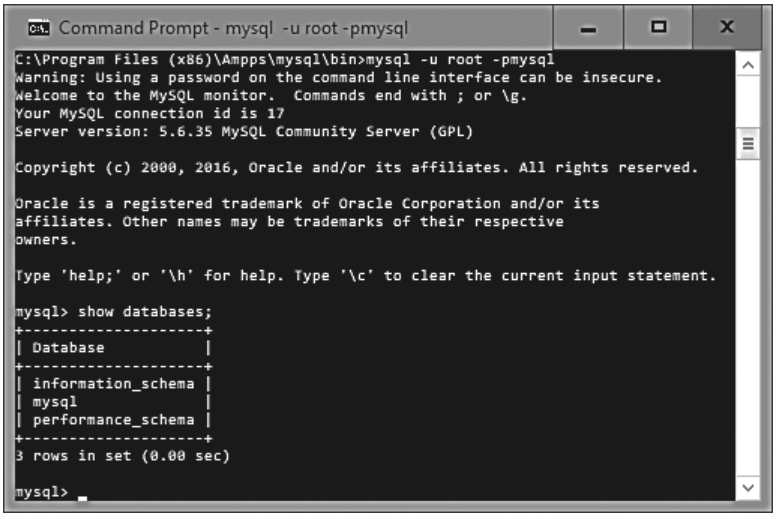

Рис. 8.1. Доступ к MySQL из командной строки Windows

Теперь можете перейти к подразделу «Использование интерфейса командной строки».

Для пользователей macOS

Чтобы иметь возможность выполнять то, о чем говорится в этой главе, нужно в соответствии с инструкциями, изложенными в главе 2, установить среду AMPPS. Следует также иметь работающий веб-сервер с запущенным сервером MySQL.

Для входа в интерфейс командной строки MySQL необходимо запустить программу Terminal (доступную в меню Utilities программы Finder). Затем вызвать программу MySQL, которая должна быть установлена в каталоге `/Applications/ampps/mysql/bin`.

По умолчанию исходным пользователем для MySQL будет пользователь по имени `root`, и пароль у него будет `mysql`. Поэтому для запуска программы наберите следующую команду:

```
/Applications/ampps/mysql/bin/mysql -u root -pmysql
```

Эта команда предпишет MySQL зарегистрировать вас как пользователя `root` с использованием пароля `mysql`. Чтобы проверить, что все в порядке, наберите следующую команду (результаты ее выполнения показаны на рис. 8.2):

```
SHOW databases;
```

Если будет получено сообщение об ошибке, не позволяющей подключиться к локальному серверу MySQL (`Can't connect to local MySQL server through socket`),

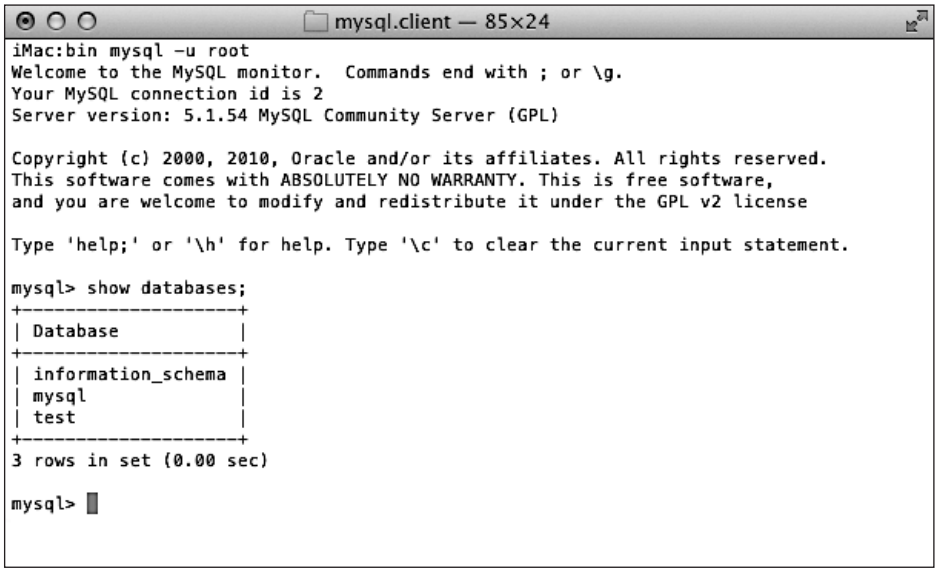

Рис. 8.2. Доступ к MySQL из программы Terminal в macOS

то сначала вам придется запустить сервер MySQL в соответствии с описанием, приведенным в главе 2.

Теперь вы готовы перейти к подразделу «Использование интерфейса командной строки».

Для пользователей Linux

На машинах, работающих под управлением UNIX-подобных операционных систем, к которым относится и Linux, PHP и MySQL практически всегда будут заранее установлены и запущены, позволяя вводить текст примеров следующего раздела (если же этого не случилось, нужно выполнить процедуры, которые рассматривались в главе 2, когда шел разговор об установке AMPPS). Сначала требуется войти в систему MySQL, введя следующую команду:

```
mysql -u root -p
```

Эта команда предписывает MySQL зарегистрировать вас под именем `root` и запросить пароль. Если у вас есть пароль, то нужно его ввести, если пароль отсутствует — просто нажать клавишу Enter.

После входа в систему нужно проверить работоспособность программы, набрав следующую команду, примерный результат выполнения которой показан на рис. 8.3:

```
SHOW databases;
```

```
You may also use sysinstall(8) to re-enter the installation and
configuration utility.  Edit /etc/motd to change this login announcement.

robnix# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4377812
Server version: mysql-server-5.0.51a

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.02 sec)

mysql>
```

Рис. 8.3. Доступ к MySQL в Linux

Если по каким-то причинам эта команда не сработает, обратитесь, пожалуйста, к главе 2, чтобы убедиться в том, что MySQL установлена должным образом.

Если команда будет выполнена, можете перейти к подразделу «Использование интерфейса командной строки».

MySQL на удаленном сервере

При получении доступа к MySQL на удаленном сервере, который, скорее всего, будет работать под управлением операционной системы семейства типа Linux/FreeBSD/UNIX, нужно выйти на удаленную машину с помощью безопасного протокола SSH (использования небезопасного протокола Telnet нужно избегать всеми силами). После подключения к удаленной машине можно встретиться с незначительными вариациями в порядке работы, зависящими от настроек сервера, которые выполнены системным администратором, особенно если этот сервер предназначен для коллективного пользования. Поэтому следует убедиться в доступности MySQL и в том, что у вас есть имя пользователя и пароль. Получив эти сведения, можно набрать следующую команду, в которой вместо *имя_пользователя* нужно вставить предоставленное имя пользователя:

```
mysql -u имя_пользователя -p
```

После появления приглашения необходимо ввести пароль. Затем можно попробовать ввести следующую команду, примерный результат выполнения которой показан на рис. 8.3:

```
SHOW databases;
```

В перечне баз данных могут присутствовать и другие ранее созданные базы, среди которых базы `test` может и не оказаться.

Следует также понимать, что все находится под полным контролем системного администратора и вы можете столкнуться с некоторыми неожиданными настройками. Например, может оказаться, что вам следует ставить перед именами всех создаваемых вами баз данных уникальную идентификационную строку, обеспечивающую их бесконфликтную работу с базами данных, созданными другими пользователями.

При возникновении любых проблем нужно переговорить с системным администратором, который должен с ними разобраться. У него нужно запросить имя пользователя и пароль, а также можно попросить дать вам возможность создавать новые базы данных или как минимум попросить создать для вас хотя бы одну готовую к работе базу данных. Тогда в этой базе можно будет создать все необходимые таблицы.

Использование интерфейса командной строки

Для всего, что изложено далее в тексте главы, нет никакой разницы, из какой именно системы — Windows, macOS или Linux — вы получаете непосредственный доступ к MySQL, поскольку все используемые команды (и сообщения об ошибках, которые могут быть получены) абсолютно одинаковы.

Точка с запятой

Начнем с самого простого. Набирая команду, вы, наверное, заметили точку с запятой (`;`) в конце `SHOW databases;`? Этот символ используется в MySQL для завершения команд или отделения их друг от друга. Если забыть поставить этот символ, MySQL выдаст приглашение и будет ожидать от вас его ввода. Запрашиваемая точка с запятой стала частью синтаксиса, позволяющего вводить длинные команды, разбивая их на несколько строк. Она также позволяет вводить сразу несколько команд, после каждой из которых стоит точка с запятой. После нажатия вами клавиши Enter интерпретатор получит все эти команды в едином пакете и выполнит их в порядке следования.

Вместо результата введенной команды довольно часто появляется приглашение MySQL. Это означает, что вы забыли поставить завершающую точку с запятой. В таком случае нужно просто ввести точку с запятой, нажать клавишу Enter, и вы получите желаемый результат.

На экране могут появляться шесть разных приглашений MySQL (табл. 8.2), позволяющих определить, на каком именно этапе многострочного ввода вы находитесь.

Таблица 8.2. Шесть приглашений к вводу команды MySQL

Приглашение MySQL	Значение
mysql>	Готова к работе и ждет ввода команды
–>	Ожидание следующей строки команды
'>	Ожидание следующей строки строкового значения, которое начиналось с одинарной кавычки
">	Ожидание следующей строки строкового значения, которое начиналось с двойной кавычки
`>	Ожидание следующей строки строкового значения, которое начиналось с символа засечки (`)
/*>	Ожидание следующей строки комментария, который начинался с символов /*

Отмена команды

Если, набрав часть команды, вы решили, что ее вообще не следует выполнять, то ни в коем случае *не пользуйтесь* сочетанием Ctrl+C! Его нажатие приведет к закрытию программы. Вместо этого можно ввести символы `\c` и нажать клавишу Enter. Порядок использования этой команды показан в примере 8.1.

Пример 8.1. Отмена ввода строки

```
бессмысленная для mysql строка \c
```

При наборе этой строки MySQL проигнорирует все ранее введенные символы и выдаст новое приглашение. Без `\c` программа выведет сообщение об ошибке. Но этой парой символов нужно пользоваться с оглядкой: если у вас уже есть открытая строка или комментарий, то прежде, чем применить `\c`, вам придется их закрыть, иначе MySQL примет `\c` за часть строки. В примере 8.2 показано, как в таком случае следует задействовать `\c`.

Пример 8.2. Отмена ввода из строки

```
это "бессмысленная для mysql строка" \c
```

Следует также заметить, что комбинация `\c` после точки с запятой не отменит предыдущую команду, поскольку это уже будет новая инструкция.

Команды MySQL

Нам уже приходилось встречаться с командой `SHOW`, которая выводит список таблиц, баз данных и многих других элементов. В табл. 8.3 приведен перечень наиболее востребованных команд.

Таблица 8.3. Наиболее востребованные команды MySQL

Команда	Действие
ALTER	Внесение изменений в базу данных или таблицу
BACKUP	Создание резервной копии таблицы
\c	Отмена ввода
CREATE	Создание базы данных
DELETE	Удаление строки из таблицы
DESCRIBE	Описание столбцов таблиц
DROP	Удаление базы данных или таблицы
EXIT (Ctrl+C)	Выход
GRANT	Изменение привилегий пользователя
HELP (\h, \?)	Отображение подсказки
INSERT	Вставка данных
LOCK	Блокировка таблицы (таблиц)
QUIT (\q)	То же самое, что и EXIT
RENAME	Переименование таблицы
SHOW	Список сведений об объектах
SOURCE	Выполнение команд из файла
STATUS (\s)	Отображение текущего состояния
TRUNCATE	Опустошение таблицы
UNLOCK	Снятие блокировки таблицы (таблиц)
UPDATE	Обновление существующей записи
USE	Использование базы данных

Многие из представленных команд будут рассмотрены по мере изучения этой главы, но сначала следует запомнить два важных положения, касающихся команд MySQL.

- Команды и ключевые слова SQL нечувствительны к регистру. Все три команды — `CREATE`, `create` и `CrEaTe` — абсолютно идентичны по смыслу. Но, чтобы было понятнее, для команд лучше использовать буквы верхнего регистра.
- Имена таблиц нечувствительны к регистру в Windows, но чувствительны к регистру в Linux и macOS. Поэтому из соображений переносимости нужно всегда выбирать буквы одного из регистров и пользоваться только ими. Для имен таблиц рекомендуется использовать буквы нижнего регистра или комбинацию из букв верхнего и нижнего регистра.

Создание базы данных

Если вы работаете на удаленном сервере, у вас только одна учетная запись пользователя и вы имеете допуск только к одной созданной для вас базе данных, то можете перейти к изучению пункта «Создание таблицы» далее. А если это не так, то продолжим, введя следующую команду для создания новой базы данных по имени `publications`:

```
CREATE DATABASE publications;
```

При успешном выполнении команды будет выведено сообщение, пока не имеющее для нас особого смысла, — `Query OK, 1 row affected (0.38 sec)` (Запрос выполнен, обработана 1 строка за 0,38 с), но вскоре все станет на свои места. После создания базы данных с ней нужно будет работать, поэтому даем следующую команду:

```
USE publications;
```

Теперь должно быть выведено сообщение об изменении текущей базы данных (`Database changed`), и после этого база будет готова к продолжению работы со следующими примерами.

Организация доступа пользователей

Теперь, когда вы уже убедились в том, насколько просто пользоваться MySQL, и создали свою первую базу данных, настало время посмотреть, как происходит организация доступа пользователей, поскольку, вполне вероятно, вам не захочется предоставлять PHP-сценариям привилегированный доступ (`root`) к MySQL, что грозит большими неприятностями в том случае, если кому-то вздумается взломать ваш сайт.

Для создания нового пользователя выдается команда предоставления прав — `GRANT`, которая принимает следующую форму (не вздумайте все это набирать, поскольку это еще не команда):

```
GRANT ПРАВА ON база_данных.объект TO 'имя_пользователя@имя_хоста'
  IDENTIFIED BY 'пароль';
```

Эта форма не должна вызвать каких-либо затруднений, быть может, за исключением фрагмента *`база_данных.объект`*. Это ссылка на саму базу данных и на содержащиеся в ней объекты, например на таблицы (табл. 8.4).

Таблица 8.4. Примерные параметры для команды GRANT

Параметр	Значение
**	Все базы данных и все их объекты
база_данных.*	Только база данных с именем *база_данных* и все ее объекты
база_данных.объект	Только база данных с именем *база_данных* и ее объект с именем *объект*

Итак, создадим пользователя, который получит доступ только к новой базе данных `publications` и ко всем ее объектам, и введем для этого следующую команду (заменив в ней имя пользователя `jim` и пароль `mypasswd` выбранными вами именем и паролем):

```
GRANT ALL ON publications.* TO 'jim'@'localhost'
  IDENTIFIED BY 'mypasswd';
```

Эта команда предоставляет пользователю `jim@localhost` полный доступ к базе данных `publications` при использовании пароля `mypasswd`. Работоспособность этой установки можно проверить, если ввести команду `quit` для выхода из системы, а затем перезапустить MySQL, воспользовавшись прежним способом запуска, но вместо `-u root -p` набрав `-u jim -p` или указав в этой строке созданное вами имя пользователя. В табл. 8.5 показаны команды, соответствующие используемой вами операционной системе, но если в вашей системе MySQL-клиент установлен в другой каталог, то в команду следует внести соответствующие коррективы.

Таблица 8.5. Запуск MySQL и вход в систему под именем jim@localhost

Операционная система	Пример команды
Windows	C:\"Program Files (x86)\Ampps\mysql\bin\mysql" -u jim -p
macOS	/Applications/ampps/mysql/bin/mysql -u jim -p
Linux	mysql -u jim -p

Теперь, как только появится приглашение, нужно лишь ввести свой пароль, и вход в систему будет открыт. Кстати, при желании можете поместить пароль сразу же после ключа `-p` (не используя никаких пробелов). Тем самым вы избежите его ввода после появления приглашения. Но такой подход не приветствуется, поскольку, если в вашей системе зарегистрировались и другие пользователи, они могут подсмотреть вводимую вами команду и получить доступ к вашему паролю.

Вы можете предоставлять только те права, которыми уже обладаете, и должны иметь право на ввод команд GRANT. Этим и ограничивается выбор предоставляемых вам прав, если только вам не предоставлены абсолютно все права. Если вас интересуют подробности использования команд GRANT и REVOKE, которые позволяют отозвать уже предоставленные права, обратитесь к документации по адресу http://tinyurl.com/mysqlgrant. Вам также следует знать, что, если при создании нового пользователя не будет указана инструкция IDENTIFIED BY, у пользователя не будет пароля, из-за чего возникнет неблагоприятная с точки зрения безопасности ситуация, которой лучше избегать.

Создание таблицы

В данный момент вы должны находиться в системе MySQL, обладать всеми (`ALL`) правами, выделенными для базы данных `publications` (или той базы данных, которая была для вас создана), и быть готовыми к созданию своей первой таблицы.

Поэтому нужно включить базу данных в работу, набрав следующую команду (и заменив `publications` именем своей базы данных, если оно у нее другое):

```
USE publications;
```

Теперь наберите построчно команды, которые приведены в примере 8.3.

Пример 8.3. Создание таблицы под названием classics

```
CREATE TABLE classics (
  author VARCHAR(128),
  title VARCHAR(128),
  type VARCHAR(16),
  year CHAR(4)) ENGINE InnoDB;
```

Последние два слова в этой команде требуют небольшого разъяснения. Обработка запросов внутри MySQL может осуществляться множеством различных способов, которые поддерживаются разными исполнительными механизмами. Начиная с версии 5.6, исходным механизмом работы с хранилищем MySQL является InnoDB, используемый нами из-за имеющейся в нем поддержки поиска в режиме FULLTEXT. Если у вас относительно новая версия MySQL, часть команды ENGINE InnoDB при создании таблицы можно опустить, но я ее сохранил для того, чтобы подчеркнуть, какой именно исполнительный механизм используется.

Если у вас запущена версия MySQL ниже 5.6, то механизм InnoDB не станет поддерживать индексы FULLTEXT и вам придется вместо InnoDB указать в команде MyISAM, чтобы показать, что нужно воспользоваться именно этим исполнительным механизмом (см. далее раздел «Создание индекса FULLTEXT»).

InnoDB в целом более эффективен и является рекомендуемым вариантом применения. Если у вас в соответствии с подробными инструкциями, изложенными в главе 2, установлен пакет программных средств AMPPS, вы будете пользоваться версией MySQL не ниже 5.6.35.

Предыдущую команду можно ввести одной строкой:

```
CREATE TABLE classics (author VARCHAR(128), title VARCHAR(128), type
VARCHAR(16), year CHAR(4)) ENGINE InnoDB;
```

но команды MySQL могут быть длинными и сложными, поэтому я рекомендую набирать их в формате, показанном в примере 8.3, до тех пор, пока вы не привыкнете к набору длинных строк.

После ввода команды MySQL должна выдать ответ: `Query OK, 0 rows affected`, а также показать время, затраченное на выполнение команды. Если вместо этого появится сообщение об ошибке, внимательно проверьте синтаксис команды. Должны быть на месте все скобки и запятые, а может быть, допущена какая-нибудь опечатка.

Чтобы проверить факт создания новой таблицы, наберите команду:

```
DESCRIBE classics;
```

Если все в порядке, то вы увидите последовательность команд и ответов, показанных в примере 8.4, и в ней особое внимание нужно обратить на отображение формата таблицы.

Пример 8.4. Сеанс работы с MySQL: создание и проверка формата новой таблицы

```
mysql> USE publications;
Database changed
mysql> CREATE TABLE classics (
    -> author VARCHAR(128),
    -> title VARCHAR(128),
    -> type VARCHAR(16),
    -> year CHAR(4)) ENGINE InnoDB;
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE classics;
+---------+----------------+-------+------+----------+-------+
| Field   | Type           | Null  | Key  | Default  | Extra |
+---------+----------------+-------+------+----------+-------+
| author  | varchar(128)   | YES   |      | NULL     |       |
| title   | varchar(128)   | YES   |      | NULL     |       |
| type    | varchar(16)    | YES   |      | NULL     |       |
| year    | char(4)        | YES   |      | NULL     |       |
+---------+----------------+-------+------+----------+-------+
4 rows in set (0.00 sec)
```

Команда `DESCRIBE` является неоценимым средством отладки, когда нужно убедиться в успешном создании таблицы MySQL. Этой командой можно воспользоваться также для того, чтобы просмотреть имена полей или столбцов таблицы и типы данных в каждом из них. Рассмотрим подробнее все заголовки:

- `Field` — имя каждого из полей или столбцов таблицы;
- `Type` — тип данных, сохраняемых в поле;
- `Null` — заголовок, который показывает, может ли поле содержать значение `NULL`;
- `Key` — примененный тип ключа, если таковой имеется (*ключи*, или *индексы* в MySQL позволяют ускорить просмотр и поиск данных);
- `Default` — исходное значение, присваиваемое полю, если при создании новой строки не указано никакого значения;
- `Extra` — дополнительная информация, например о настройке поля на автоматическое приращение его значения.

Типы данных

В примере 8.3 можно было заметить, что для трех полей таблицы объявлены типы данных `VARCHAR`, а для одного — тип данных `CHAR`. Термин `VARCHAR` означает *VARiable length CHARacter string* — строка символов переменной длины, а команда «видит» числовое значение, указывающее максимальную длину, разрешенную для строки, хранящейся в этом поле.

Оба типа данных (и CHAR, и VARCHAR) принимают строки текста, ограничивая их длину размером поля. Разница между ними состоит в том, что каждая строка в поле CHAR имеет указанный размер. Если поместить в него строку меньшего размера, она будет дополнена пробелами. В поле VARCHAR дополнения текста не происходит; его размер может изменяться таким образом, чтобы в него помещался вставленный текст. Но при использовании поля VARCHAR требуется идти на небольшие издержки, чтобы отслеживать размер каждого значения. Поэтому CHAR больше подходит для тех случаев, когда данные во всех записях имеют одинаковый размер, а VARCHAR эффективнее применять, когда размеры могут сильно отличаться друг от друга и возрастать. Но за это приходится расплачиваться тем, что доступ к данным типа VARCHAR осуществляется несколько медленнее, чем к данным типа CHAR.

Еще одной особенностью символьных и текстовых столбцов, играющей важную роль для сегодняшнего глобального Интернета, является *набор символов*. Вполне очевидно, что для английского и русского языков используются разные наборы символов. При создании символьных или текстовых столбцов им можно присвоить тот или иной набор символов.

Тип данных VARCHAR очень удобен в нашем примере, поскольку позволяет разместить имена авторов и названия различной длины, помогая MySQL планировать размер базы данных и эффективнее осуществлять просмотр и поиск данных. Но есть у него и недостаток: если присвоить строковое значение длиннее позволенного, оно будет усечено до максимальной длины, объявленной в определении таблицы.

Но у поля `year` (год) вполне предсказуемые значения, поэтому вместо VARCHAR для него используется более подходящий тип данных — `CHAR(4)`. Параметр 4 позволяет выделить под него 4 байта данных, поддерживающих все года от –999 до 9999; байт содержит 8 бит и может иметь значения от 00000000 до 11111111, что в десятичном представлении означает от 0 до 255.

Можно было бы, конечно, сохранять год и в значении, состоящем из двух цифр, но если данные не утратят своей актуальности и в следующем столетии или показатель лет каким-то образом опять вернется к нулевому значению, то эту проблему нужно решать в первоочередном порядке, поскольку она очень похожа на «проблему 2000 года», из-за которой даты начиная с 1 января 2000 года во многих крупных компьютерных системах могли быть отнесены к 1900 году.

Тип данных YEAR не выбран для хранения года в таблице classics потому, что он поддерживает только 0000 год и диапазон лет с 1901 по 2155. MySQL из соображений эффективности хранит значение года в одном байте, а это значит, что храниться могут только 256 значений, в то время как книги в таблице classics изданы задолго до 1901 года.

Тип данных CHAR

В табл. 8.6 перечислены типы символьных данных CHAR. Оба типа предлагают указать параметр, устанавливающий максимальную (или точную) длину строки, которая может быть помещена в поле. Из таблицы следует, что у каждого типа есть присущее ему максимальное значение длины. Для данных типа VARCHAR длиной от 0 до 255 байт требуется еще один байт в хранилище, а для данных длиной более 256 байт требуется еще два байта.

Таблица 8.6. Типы данных CHAR, используемые в MySQL

Тип данных	Количество байтов	Примеры
CHAR(*n*)	В точности равное *n* (<= 256)	CHAR(5) Hello использует 5 байт CHAR(57) Goodbye использует 57 байт
VARCHAR(*n*)	Вплоть до *n* (<= 65 536)	VARCHAR(7) Hello использует 5 байт VARCHAR(100) Goodbye использует 7 байт

Тип данных BINARY

Тип данных BINARY (табл. 8.7) применяется для хранения строк байтов, не имеющих связанного с ними набора символов. Например, тип данных BINARY можно использовать для хранения изображения в формате GIF.

Таблица 8.7. Типы данных BINARY, используемые в MySQL

Тип данных	Количество байтов	Примеры
BINARY(*n*)	В точности равное *n* (<= 256)	Похож на CHAR, но содержит двоичные данные
VARBINARY(*n*)	Вплоть до *n* (<= 65 536)	Похож на VARCHAR, но содержит двоичные данные

Типы данных TEXT

Символьные данные могут быть также сохранены в одном из наборов полей TEXT. Различия между этими полями и полями VARCHAR невелики.

- До выхода версии 5.0.3 MySQL удалял из полей VARCHAR все начальные и замыкающие пробелы.
- В полях типа TEXT не может быть исходных значений.
- В столбце TEXT MySQL индексирует только первые *n* символов (*n* задается при создании индекса).

Это означает, что VARCHAR является более приемлемым и быстрее обрабатываемым типом данных, если нужно вести поиск по всему содержимому поля. Если поиск

никогда не будет вестись более чем в конкретном количестве начальных символов хранящегося в поле значения, то, наверное, нужно остановить свой выбор на типе данных TEXT (табл. 8.8).

Таблица 8.8. Типы данных TEXT, используемые в MySQL

Тип данных	Количество байтов	Особенности
TINYTEXT(*n*)	Вплоть до *n* (<= 256)	Считается строкой с набором символов
TEXT(*n*)	Вплоть до *n* (<= 65 536)	Считается строкой с набором символов
MEDIUMTEXT(*n*)	Вплоть до *n* (<= 1,67e+7)	Считается строкой с набором символов
LONGTEXT(*n*)	Вплоть до *n* (<= 4,29e+9)	Считается строкой с набором символов

Использование типов данных, имеющих меньший максимальный размер, также повышает эффективность работы, поэтому следует использовать наиболее подходящий известный тип с наименьшим максимальным размером, способный вместить сохраняемую в поле строку.

Тип данных BLOB

Аббревиатура BLOB означает *Binary Large Object — большой двоичный объект*, и поэтому, как и можно было предположить, тип данных BLOB больше всего подходит для хранения двоичных данных, превышающих по объему 65 536 байт. Другим основным отличием BLOB от типа данных BINARY является то, что для столбцов типа BLOB нельзя задавать исходные значения. Типы данных BLOB перечислены в табл. 8.9.

Таблица 8.9. Типы данных BLOB, используемые в MySQL

Тип данных	Количество байтов	Особенности
TINYBLOB(*n*)	Вплоть до *n* (<= 256)	Считается не набором символов, а двоичными данными
BLOB(*n*)	Вплоть до *n* (<= 65 536)	Считается не набором символов, а двоичными данными
MEDIUMBLOB(*n*)	Вплоть до *n* (<= 1,67e+7)	Считается не набором символов, а двоичными данными
LONGBLOB(*n*)	Вплоть до *n* (<= 4,29e+9)	Считается не набором символов, а двоичными данными

Числовые типы данных

В MySQL поддерживаются различные числовые типы данных — от одиночного байта до чисел с плавающей точкой с удвоенной точностью. Хотя для числового поля можно использовать до 8 байт, лучше все же выбрать поле с самым скром-

ным типом данных, в котором способно уместиться наибольшее из ожидаемых вами значений. Тогда ваша база данных будет небольшой по объему и быстрой по доступу.

В табл. 8.10 перечислены числовые типы данных, поддерживаемые MySQL, и диапазоны значений, которые могут содержаться в их полях. Если вы не знакомы с терминологией, поясню, что число *со знаком* имеет диапазон возможных значений от отрицательного до нуля и от нуля до положительного значения, а число *без знака* может быть в диапазоне от нуля до положительного значения. Оба они могут иметь одинаковую величину, нужно лишь представить число со знаком, сдвинутым наполовину влево, с одной половиной в отрицательном, а с другой — в положительном диапазоне. Следует заметить, что значения с плавающей точкой (любой точности) могут быть только числами со знаком.

Таблица 8.10. Числовые типы данных, используемые в MySQL

Тип данных	Количество байтов	Минимальное значение		Максимальное значение	
		Со знаком	Без знака	Со знаком	Без знака
TINYINT	1	–128	0	127	255
SMALLINT	2	–32 768	0	32 767	65 535
MEDIUMINT	3	–8,38e+6	0	8,38e+6	1,67e+7
INT или INTEGER	4	–2,15e+9	0	2,15e+9	4,29e+9
BIGINT	8	–9,22e+18	0	9,22e+18	1,84e+19
FLOAT	4	–3,40e+38	Не бывает	3,40e+38	Не бывает
DOUBLE или REAL	8	–1,80e+308	Не бывает	1,80e+308	Не бывает

Чтобы указать, какой именно тип данных используется, со знаком или без знака, применяется спецификатор `UNSIGNED`. В следующем примере создается таблица по имени `tablename`, содержащая поле `fieldname` с типом данных `UNSIGNED INTEGER`:

```
CREATE TABLE tablename (fieldname INT UNSIGNED);
```

При создании числового поля можно также передать в качестве параметра необязательное число:

```
CREATE TABLE tablename (fieldname INT(4));
```

Но при этом следует помнить, что, в отличие от типов данных `BINARY` и `CHAR`, этот параметр не показывает количество байтов, выделяемых под хранение. Может быть, это противоречит интуитивному восприятию, но на самом деле это число обозначает отображаемую ширину данных в поле при его извлечении. Оно часто используется вместе со спецификатором `ZEROFILL`:

```
CREATE TABLE tablename (fieldname INT(4) ZEROFILL);
```

Этот спецификатор указывает на то, что все числа шириной меньше четырех символов дополняются одним или несколькими нулями, для того чтобы ширина отображаемого поля составляла четыре символа. Если поле уже занимает четыре и более символа, дополнение не производится.

Типы данных DATE и TIME

В табл. 8.11 показана еще одна важная категория типов данных, поддерживаемая MySQL, которая относится к дате и времени.

Таблица 8.11. Типы данных DATE и TIME, используемые в MySQL

Тип данных	Формат времени-даты
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (только годы 0000 и 1901–2155)

Значения, имеющие типы данных `DATETIME` и `TIMESTAMP`, отображаются одинаково. Основное различие в том, что у `TIMESTAMP` слишком узкий диапазон (от 1970 до 2037 года), а в `DATETIME` может храниться практически любая нужная дата, если только вы не интересуетесь античной историей или научной фантастикой.

Но `TIMESTAMP` также полезен, потому что, используя его, можно позволить MySQL установить для вас нужное значение. Если при добавлении строки не задавать значение для поля с этим типом данных, то в него автоматически будет вставлено текущее время. Можно также заставить MySQL обновлять столбец с типом данных `TIMESTAMP` при каждом изменении строки.

Атрибут AUTO_INCREMENT

Иногда нужно обеспечить уникальность каждой строки, имеющейся в базе данных. В вашей программе это можно сделать за счет тщательной проверки вводимых данных и обеспечения их различия хотя бы в одном из значений в любых двух строках. Но такой подход не защищен от ошибок и работает только в конкретных обстоятельствах. Например, в таблице один и тот же автор может появляться несколько раз. Точно так же, скорее всего, будет повторяться год издания и т. д. В таком случае гарантировать отсутствие продублированных строк будет довольно трудно.

В общем виде эта проблема решается за счет специально выделенного дополнительного столбца. Вскоре мы рассмотрим использование ISBN (International

Standard Book Number — международный стандартный книжный номер) издания, но сначала нужно представить вам тип данных с автоприращением — `AUTO_INCREMENT`.

В соответствии с названием столбца, которому назначен этот тип данных, его содержимому будет устанавливаться значение на единицу большее, чем значение записи в этом же столбце в предыдущей вставленной строке. В примере 8.5 показано, как нужно добавлять новый столбец по имени `id` к таблице `classics` и придавать ему свойства автоприращения.

Пример 8.5. Добавление столбца id с автоприращением

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

Здесь представлена команда `ALTER`, очень похожая на команду `CREATE`. Команда `ALTER` работает с уже существующей таблицей и может добавлять, изменять или удалять столбцы. В нашем примере добавляется столбец по имени `id`, имеющий следующие характеристики.

- `INT UNSIGNED` — делает столбец способным принять целое число, достаточно большое для того, чтобы в таблице могло храниться более 4 млрд записей.
- `NOT NULL` — обеспечивает наличие значения в каждой записи столбца. Многие программисты используют его в поле `NULL`, чтобы показать отсутствие в нем какого-либо значения. Но тогда могут появляться дубликаты, противоречащие самому смыслу существования этого столбца. Поэтому появление в нем значения `NULL` запрещено.
- `AUTO_INCREMENT` — заставляет MySQL установить для этого столбца уникальное значение в каждой строке, как было описано ранее. Фактически мы не управляем значением, которое будет появляться в каждой строке этого столбца, но это и не нужно: все, о чем мы беспокоимся, — гарантия уникальности этого значения.
- `KEY` — столбец с автоприращением полезно использовать в качестве ключа, поскольку вы будете стремиться искать строки на основе значений этого столбца. Пояснения будут даны в разделе «Индексы» далее.

Теперь каждая запись будет иметь уникальное число в столбце `id`, для первой записи это будет начальное число 1, а счет других записей будет вестись по нарастающей. Как только будет вставлена новая строка, в ее столбец `id` будет автоматически записано следующее по порядку число.

Этот столбец можно не добавлять после создания таблицы, а сразу включить в нее, слегка изменив формат команды `CREATE`. В данном случае команда из примера 8.3 должна быть заменена командой из примера 8.6. Обратите особое внимание на ее последнюю строку.

Пример 8.6. Добавление столбца id с автоприращением при создании таблицы

```
CREATE TABLE classics (
  author VARCHAR(128),
  title VARCHAR(128),
  type VARCHAR(16),
  year CHAR(4),
  id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE InnoDB;
```

Если хочется проверить, был ли добавлен столбец, нужно просмотреть имеющиеся в таблице столбцы и типы данных, воспользовавшись следующей командой:

```
DESCRIBE classics;
```

Теперь, когда мы закончили изучение этого типа данных, столбец `id` нам больше не нужен, поэтому, если вы его создали, воспользовавшись командой из примера 8.5, его нужно удалить, введя команду из примера 8.7.

Пример 8.7. Удаление столбца id

```
ALTER TABLE classics DROP id;
```

Добавление данных к таблице

Для добавления данных к таблице предназначена команда `INSERT`. Рассмотрим ее в действии, заполнив таблицу `classics` данными из табл. 8.1, многократно используя одну и ту же форму команды `INSERT` (пример 8.8).

Пример 8.8. Заполнение таблицы classics

```
INSERT INTO classics(author, title, type, year)
  VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');
INSERT INTO classics(author, title, type, year)
  VALUES('Jane Austen','Pride and Prejudice','Fiction','1811');
INSERT INTO classics(author, title, type, year)
  VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');
INSERT INTO classics(author, title, type, year)
  VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');
INSERT INTO classics(author, title, type, year)
  VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
```

После каждой второй строки вы должны увидеть сообщение об успешной обработке запроса — `Query OK`. Как только будут введены все строки, наберите следующую команду, которая отобразит содержимое таблицы:

```
SELECT * FROM classics;
```

Результат должен быть похож на тот, что показан на рис. 8.4.

Сейчас не стоит обращать внимания на команду `SELECT`, ее очередь наступит в подразделе «Создание запросов к базе данных MySQL» раздела «Индексы». Достаточно сказать, что в таком виде она отображает все только что введенные данные.

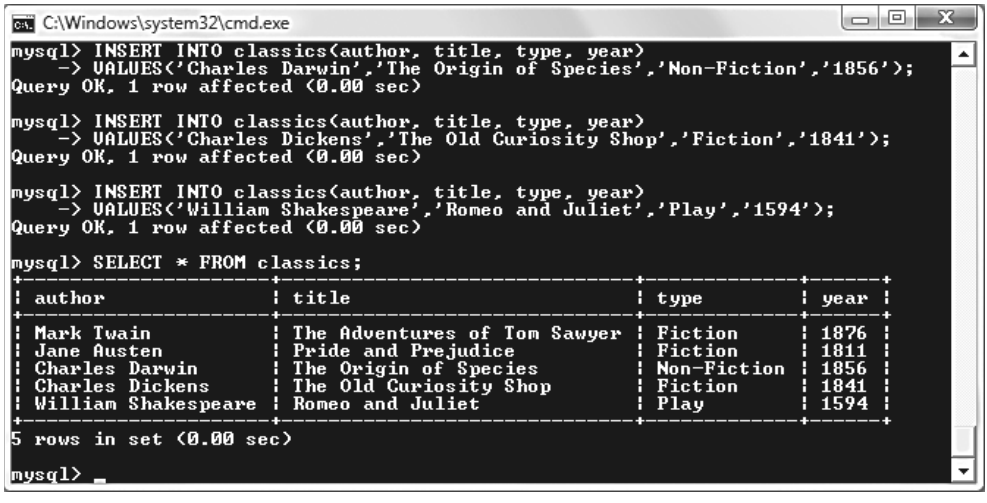

Рис. 8.4. Заполнение таблицы classics и просмотр ее содержимого

Теперь вернемся назад и посмотрим, как используется команда `INSERT`. Ее первая часть, `INSERT INTO classics`, сообщает MySQL, куда нужно вставлять следующие за ней данные. Затем в круглых скобках перечисляются четыре имени столбцов: `author`, `title`, `type` и `year`, которые отделяются друг от друга запятыми. Таким образом MySQL сообщается, что именно в эти четыре поля будут вставляться данные.

Во второй строке каждой команды `INSERT` содержится ключевое слово `VALUES`, за которым следуют четыре строковых значения, взятых в кавычки и отделенных друг от друга запятыми. Они обеспечивают MySQL теми четырьмя значениями, которые будут вставлены в четыре ранее указанных столбца. (Как и во всех остальных примерах, разбиение команды на строки было моим собственным решением, придерживаться которого не обязательно.)

Каждый элемент данных будет вставлен по порядку в соответствующие столбцы. Если порядок перечисления столбцов и данных будет случайно перепутан, данные попадут не в те столбцы. А количество указанных столбцов должно соответствовать количеству элементов данных. (Есть более безопасные способы использования `INSERT`, которые вскоре будут рассмотрены.)

Переименование таблиц

Переименование таблиц, как и любые другие изменения ее структуры или метаданных, осуществляется посредством команды `ALTER`. Поэтому, чтобы, к примеру, изменить имя таблицы `classics` на `pre1900`, воспользуйтесь следующей командой:

```
ALTER TABLE classics RENAME pre1900;
```

Если применить эту команду, то потом, чтобы без изменений работали все последующие примеры текущей главы, вам придется вернуть таблице ее прежнее имя, для чего нужно будет ввести такую команду:

```
ALTER TABLE pre1900 RENAME classics;
```

Изменение типа данных столбца

Для изменения типа данных столбца также используется команда `ALTER`, но в этом случае вместе с ней применяется ключевое слово `MODIFY`. Поэтому для изменения типа данных столбца `year` с `CHAR(4)` на `SMALLINT` (для которого потребуется только 2 байта памяти, что способствует экономии дискового пространства) нужно ввести следующую команду:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

После этого, если для MySQL есть смысл конвертировать тип данных, система автоматически изменит данные, сохраняя их значение. В этом случае она заменит каждое строковое значение сопоставимым с ним целым числом, пока строку можно будет распознать как отображение целого числа.

Добавление нового столбца

Предположим, что таблица создана и заполнена большим объемом данных и тут выяснилось, что нужен еще один столбец. Не стоит расстраиваться. Посмотрите, как можно добавить к таблице новый столбец `pages`, который будет использоваться для хранения количества страниц, имеющихся в книге:

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Эта команда добавляет новый столбец по имени `pages`, в котором используется тип данных `UNSIGNED SMALLINT`, подходящий для хранения значений вплоть до 65 535. Этого наверняка более чем достаточно для любой когда-либо изданной книги!

И если запросить у MySQL описание обновленной таблицы, воспользовавшись показанной далее командой `DESCRIBE`, то можно будет увидеть внесенные в нее изменения (рис. 8.5):

```
DESCRIBE classics;
```

Переименование столбца

Посмотрев еще раз на рис. 8.5, можно заметить, что наличие в таблице столбца `type` приводит к путанице, поскольку такое же имя используется MySQL для идентификации типа данных. Но это не проблема — изменим имя этого столбца на `category`:

```
ALTER TABLE classics CHANGE type category VARCHAR(16);
```

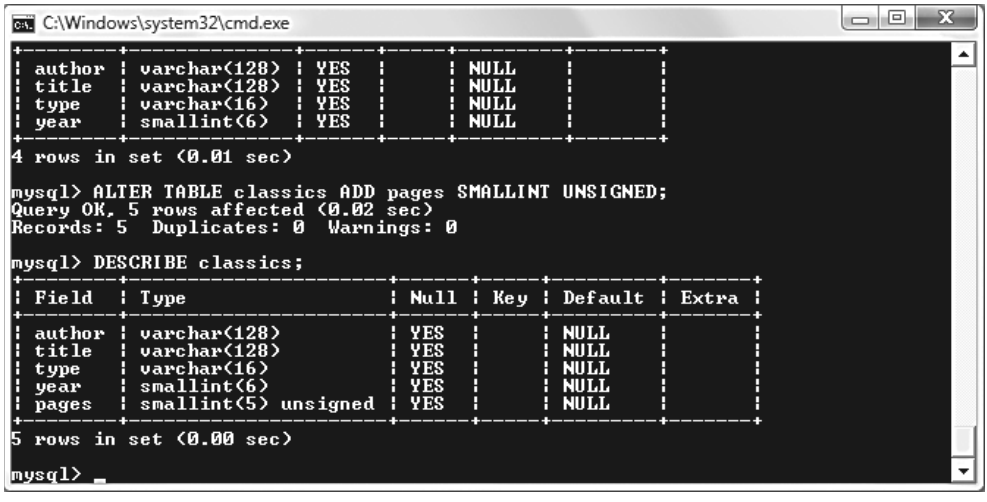

Рис. 8.5. Добавление нового столбца pages и просмотр таблицы

Обратите внимание на добавление `VARCHAR(16)` в конце этой команды. Это связано с тем, что ключевое слово `CHANGE` требует указания типа данных даже в том случае, если вы не собираетесь его изменять, и `VARCHAR(16)` — тот самый тип данных, который был указан при создании столбца `type`.

Удаление столбца

Поразмыслив, можно прийти к выводу, что столбец `pages`, в котором хранится количество страниц, не представляет для этой базы данных особой ценности, поэтому его можно удалить, используя ключевое слово `DROP`:

```
ALTER TABLE classics DROP pages;
```

Учтите, что ключевое слово DROP нужно применять с особой осторожностью, поскольку его действие носит необратимый характер и по недоразумению можно удалить целые таблицы (и даже базы данных)!

Удаление таблицы

Удалить таблицу очень просто. Но поскольку я не хочу заставлять вас заново вводить все данные в таблицу `classics`, мы ее удалять не станем. Вместо этого просто создадим новую таблицу, проверим факт ее существования, а затем удалим ее, набрав команду, приведенную в примере 8.9. Результат выполнения всех четырех команд показан на рис. 8.6.

Пример 8.9. Создание, просмотр и удаление таблицы

```
CREATE TABLE disposable(trash INT);
DESCRIBE disposable;
DROP TABLE disposable;
SHOW tables;
```

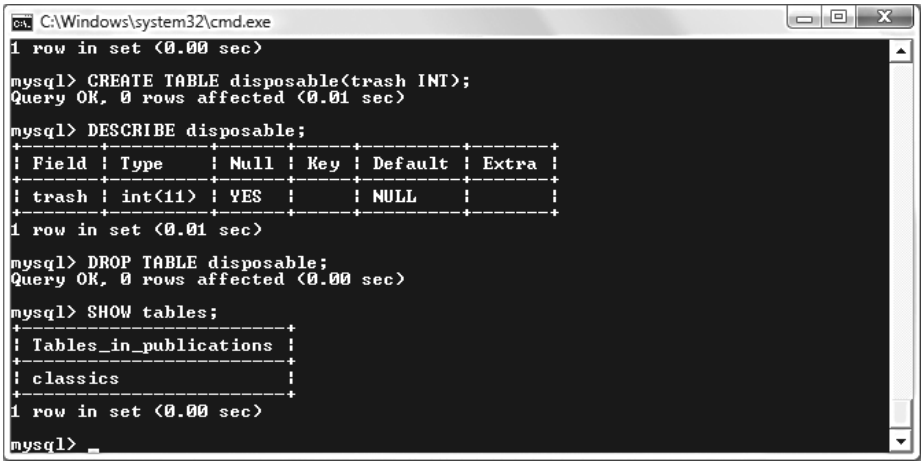

Рис. 8.6. Создание, просмотр и удаление таблицы

Индексы

В данный момент у нас есть действующая таблица `classics`, в которой можно будет без труда, пользуясь средствами MySQL, отыскать нужную информацию. Но все так просто лишь до тех пор, пока она не разрастется до пары сотен строк. Тогда с каждой добавленной строкой доступ к базе данных будет становиться все медленнее и медленнее, поскольку MySQL при обработке запроса придется вести поиск в каждой строке. Это похоже на поиск нужной информации в каждой книге, имеющейся в библиотеке.

Разумеется, вам не придется вести поиск в библиотеках подобным образом, поскольку в них есть либо обычная картотека, либо, что более вероятно, собственная база данных. То же самое относится и к MySQL, поскольку ценой небольших затрат оперативной памяти и дискового пространства можно создать «картотеку» для таблицы, которая будет использоваться MySQL для выполнения мгновенного поиска.

Создание индекса

Возможности быстрого поиска можно добиться путем добавления *индекса* либо при создании таблицы, либо в любое время впоследствии. Но сделать это не так-то просто. Существуют, к примеру, различные типы индексов, такие как `INDEX`, `PRIMARY`

KEY или FULLTEXT. Кроме того, необходимо решить, каким столбцам нужен индекс, а для этого нужно спрогнозировать, по каким данным каждого из столбцов будет осуществляться поиск. Индексы можно также усложнять, комбинируя в одном индексе данные из нескольких столбцов. И даже когда вы все это поймете, у вас будет возможность сократить размер индекса за счет ограничения объема данных каждого индексируемого столбца.

Если представить себе поисковые операции применительно к таблице `classics`, становится ясно, что поиск может осуществляться во всех столбцах. Но если бы не был удален столбец `pages`, созданный в пункте «Добавление нового столбца» выше, то он, наверное, не понадобился бы для индекса, поскольку большинство людей вряд ли стали бы искать книги по количеству страниц. Давайте все же продолжим и добавим индекс к каждому столбцу, воспользовавшись командами, приведенными в примере 8.10.

Пример 8.10. Добавление индексов к таблице classics

```
ALTER TABLE classics ADD INDEX(author(20));
ALTER TABLE classics ADD INDEX(title(20));
ALTER TABLE classics ADD INDEX(category(4));
ALTER TABLE classics ADD INDEX(year);
DESCRIBE classics;
```

Первые две команды создают индексы для столбцов авторов и названий — `author` и `title`, ограничивая каждый индекс только первыми 20 символами. Например, когда MySQL индексирует название:

```
The Adventures of Tom Sawyer
```

на самом деле в индексе будут сохранены только первые 20 символов:

```
The Adventures of To
```

Это делается для сокращения размера индекса и для оптимизации скорости доступа к базе данных. Я выбрал 20 символов, поскольку их должно быть достаточно для обеспечения уникальности большинства строк, встречающихся в данных столбцах. Если MySQL обнаружит два индекса с одинаковым содержимым, ей нужно будет понапрасну потратить время на обращение к самой таблице и на проверку проиндексированного столбца, для того чтобы определить, какая именно строка действительно соответствует условиям поиска.

Что касается столбца категории — `category`, то на данный момент, чтобы идентифицировать уникальность строки, достаточно только первого символа (F для Fiction, N для Non-Fiction и P для Play), но я выбрал индекс из четырех символов, чтобы дать возможность в будущем вводить такие категории, у которых первые три символа могут быть одинаковыми. Если позже набор категорий усложнится еще больше, этот столбец можно будет переиндексировать. И наконец, я не стал задавать ограничения на индекс столбца года издания — `year`, поскольку он имеет четко определенную длину в четыре символа.

Результат ввода этих команд (и команды DESCRIBE, позволяющей убедиться в том, что они работают) можно увидеть на рис. 8.7, который показывает наличие ключа MUL для каждого столбца. Это означает, что в этом столбце может многократно присутствовать одно и то же значение, что, собственно, нам и нужно, поскольку имена авторов могут встречаться многократно, одни и те же названия книг могут использоваться множеством авторов и т. д.

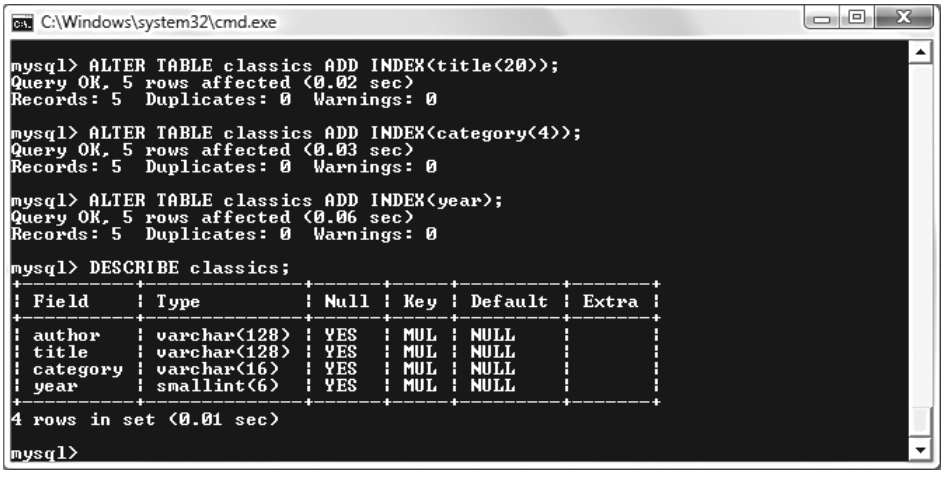

Рис. 8.7. Добавление индексов к таблице classics

Использование команды CREATE INDEX

Индекс можно добавить не только командой ALTER TABLE, но и командой CREATE INDEX. Эти две команды являются равнозначными, за исключением того, что CREATE INDEX не может использоваться для создания индекса типа первичного ключа — PRIMARY KEY (см. далее пункт «Первичные ключи»). Формат этой команды показан во второй строке примера 8.11.

Пример 8.11. Эти две команды эквивалентны

```
ALTER TABLE classics ADD INDEX(author(20));
CREATE INDEX author ON classics (author(20));
```

Добавление индексов при создании таблиц

Чтобы добавить индекс, не нужно выжидать какое-то время после создания таблицы. Это может отнять много времени, поскольку добавление индекса к большой таблице — длительный процесс. Поэтому рассмотрим команду, создающую таблицу classics с уже имеющимися индексами.

Пример 8.12 является переработкой примера 8.3, в котором одновременно с таблицами создаются индексы. Учтите, что для включения всех изменений, выполненных в данной главе, в этой версии используется новое имя столбца `category` вместо прежнего имени `type`, а для столбца `year` указан тип данных `SMALLINT`, а не `CHAR(4)`. При желании попробовать эту команду в работе без предварительного удаления текущей таблицы `classics` замените слово `classics` в первой строке каким-нибудь другим словом, например `classics1`, а после завершения работы удалите таблицу `classics1`.

Пример 8.12. Создание таблицы classics с индексами

```
CREATE TABLE classics (
  author VARCHAR(128),
  title VARCHAR(128),
  category VARCHAR(16),
  year SMALLINT,
  INDEX(author(20)),
  INDEX(title(20)),
  INDEX(category(4)),
  INDEX(year)) ENGINE InnoDB;
```

Первичные ключи

В данный момент у нас создана таблица `classics` и за счет добавления индексов обеспечен быстрый поиск, но кое-что все же упущено. Можно вести поиск по всем имеющимся в таблице изданиям, но нет единого уникального ключа для каждого издания, обеспечивающего мгновенный доступ к строке. Важность наличия ключа с уникальным значением для каждой строки проявится, когда мы станем комбинировать данные из разных таблиц.

В пункте «Атрибут AUTO_INCREMENT» подраздела «Типы данных» предыдущего раздела, где рассматривался создаваемый столбец `id` с автоприращением, было сказано, что он может быть использован в качестве первичного ключа для этой таблицы. Но я захотел возложить эту задачу на более подходящий столбец: признанный во всем мире номер ISBN.

Продолжим работу с таблицей и создадим новый столбец для этого ключа. Теперь, помня о том, что номер ISBN состоит из 13 символов, можно решить, что с задачей справится следующая команда:

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Но это не так. Если запустить эту команду на выполнение, будет получено сообщение об ошибке, связанной с дубликатом записи для ключа 1: `Duplicate entry`. Причина в том, что таблица уже заполнена данными, а эта команда пытается добавить столбец со значением `NULL` к каждой строке, что запрещено, поскольку все столбцы, использующие первичный ключ, должны иметь уникальное значение.

Но если бы таблица была пуста, то эта команда была бы выполнена без проблем, как и при добавлении первичного ключа сразу же после создания таблицы.

В сложившейся ситуации нужно немного схитрить: создать новый столбец без индекса, заполнить его данными, а затем добавить индекс ретроспективно, воспользоваться командой из примера 8.13. К счастью, в этом наборе данных каждый год имеет уникальное значение, поэтому для идентификации каждой обновляемой строки можно воспользоваться столбцом `year`. Учтите, что в этом примере применяются команда `UPDATE` и ключевое слово `WHERE`, которые более подробно будут рассмотрены в подразделе «Создание запросов к базе данных MySQL» далее.

Пример 8.13. Заполнение столбца isbn данными и использование первичного ключа

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
ALTER TABLE classics ADD PRIMARY KEY(isbn);
DESCRIBE classics;
```

После ввода этих команд будет получен результат, похожий на копию экрана, показанную на рис. 8.8. Обратите внимание на то, что в синтаксисе команды `ALTER TABLE` ключевое слово `INDEX` заменено ключевыми словами `PRIMARY KEY` (сравните примеры 8.10 и 8.13).

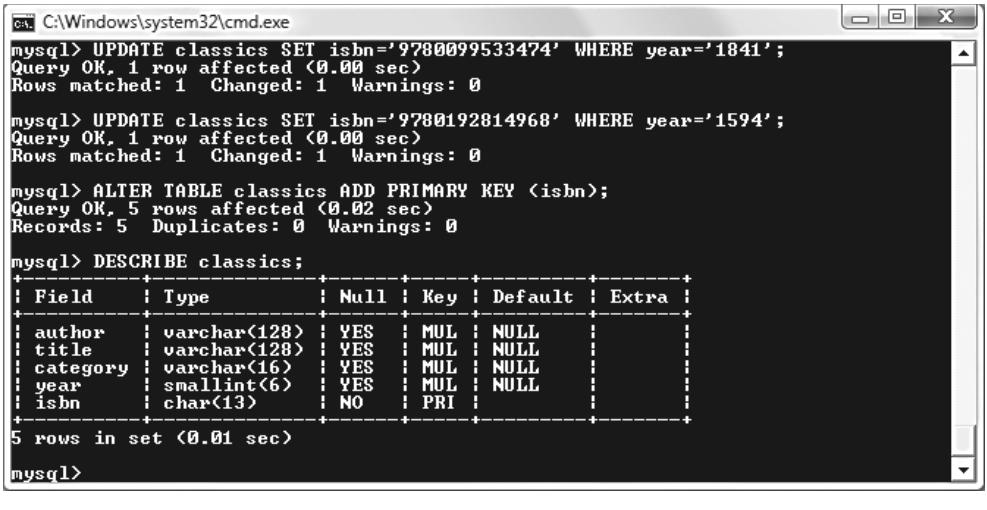

Рис. 8.8. Ретроспективное добавление первичного ключа к таблице classics

Чтобы создать первичный ключ при создании таблицы `classics`, можно воспользоваться командой, показанной в примере 8.14. И в этом случае, если вы хотите

испробовать эту команду в работе, нужно заменить имя `classics` в строке 1 каким-нибудь другим, а затем удалить проверочную таблицу.

Пример 8.14. Создание таблицы classics с первичным ключом

```
CREATE TABLE classics (
  author VARCHAR(128),
  title VARCHAR(128),
  category VARCHAR(16),
  year SMALLINT,
  isbn CHAR(13),
  INDEX(author(20)),
  INDEX(titTe(20)),
  INDEX(category(4)),
  INDEX(year),
  PRIMARY KEY (isbn)) ENGINE InnoDB;
```

Создание индекса FULLTEXT

В отличие от обычного индекса, имеющийся в MySQL индекс `FULLTEXT` позволяет осуществлять сверхбыстрый поиск целых столбцов текста. Он сохраняет каждое слово каждой строки данных в специальном индексе, в котором можно вести поиск, используя «естественный язык» наподобие того, что применяется в поисковом механизме.

Вообще-то утверждение о том, что система MySQL хранит все слова в индексе FULLTEXT, не вполне соответствует действительности, поскольку в ней имеется встроенный список более чем из 500 слов, которые она предпочитает игнорировать в силу их широкой распространенности и практической бесполезности при любом поиске. Этот список, называемый стоповыми словами — stopwords, включает слова the, as, is, of и т. д. Список помогает MySQL работать при FULLTEXT-поиске намного быстрее и не раздувать размеры базы данных. Полный список стоповых слов приведен в приложении B.

Рассмотрим особенности индексов `FULLTEXT`, о которых нужно знать.

- С выходом MySQL 5.6 появилась возможность использования индексов `FULLTEXT` с таблицами InnoDB, но прежде эти индексы могли применяться только с таблицами типа MyISAM. Если нужно привести таблицу к типу MyISAM, можно применить команду MySQL:

  ```
  ALTER TABLE tablename ENGINE = MyISAM;
  ```

- Индексы `FULLTEXT` могут создаваться только для столбцов с типами данных `CHAR`, `VARCHAR` и `TEXT`.
- Определение индекса `FULLTEXT` может быть дано в инструкции `CREATE TABLE` при создании таблицы или добавлено позже с использованием инструкции `ALTER TABLE` (или `CREATE INDEX`).

- *Намного* быстрее будет загрузить большие наборы данных в таблицу, не имеющую индекса FULLTEXT, а затем создать индекс, чем загружать их в таблицу, у которой уже имеется индекс FULLTEXT.

Чтобы создать индекс FULLTEXT, примените его к одной или нескольким записям, как в примере 8.15, в котором индекс FULLTEXT добавляется к двум столбцам — author и title, принадлежащим таблице classics (этот индекс является дополнением к тем, что уже были созданы, и не влияет на их работу).

Пример 8.15. Добавление индекса FULLTEXT к таблице classics

```
ALTER TABLE classics ADD FULLTEXT(author,title);
```

Теперь в этой паре столбцов можно вести поиск с использованием индекса FULLTEXT. Такая возможность могла бы проявиться в полную силу, если бы вы могли теперь ввести весь текст этих книг в базу данных (учитывая, что они не защищены авторскими правами), и тогда они были бы полностью доступны для поиска. Поисковые операции с использованием индекса FULLTEXT рассмотрены далее в пункте «MATCH...AGAINST» подраздела «Создание запросов к базе данных MySQL» далее.

Если система MySQL станет при доступе к вашей базе данных работать медленнее, чем вы от нее ожидали, то проблема скорее всего заключается в ваших индексах. Либо у вас нет индекса там, где он нужен, либо индексы составлены неоптимальным образом. Зачастую данная проблема решается за счет тонкой настройки индексов таблиц. Производительность не входит в тематику этой книги, но в главе 9 я дам несколько подсказок, чтобы вы знали, что именно нужно искать.

Создание запросов к базе данных MySQL

Итак, мы создали базу данных MySQL и таблицы, заполнили их данными и добавили к ним индексы, чтобы ускорить поиск. Теперь настало время посмотреть, как именно ведется этот поиск и какие для этого имеются команды и спецификаторы.

SELECT

На рис. 8.4 уже было показано, что команда SELECT используется для извлечения данных из таблицы. В том разделе я воспользовался ее наипростейшей формой для выбора всех данных и их отображения, что вам вряд ли когда-нибудь пригодится, разве что для просмотра самых маленьких таблиц, поскольку все данные будут прокручиваться на экране и скрываться в нечитаемой области. В качестве альтернативного варианта, на компьютерах под управлением Unix/Linux, MySQL можно заставить выполнить постраничный вывод данных на величину экрана, запустив команду

```
pager less;
```

Эта команда направит весь вывод на экран через канал программы less. Чтобы восстановить стандартный вывод и выключить разбиение на страницы, можно запустить команду

```
nopager;
```

А теперь рассмотрим команду `SELECT` более подробно.

Ее основной синтаксис имеет следующий вид:

```
SELECT что-нибудь FROM имя_таблицы;
```

Этим *что-нибудь*, как вы уже видели, может быть символ звездочки (*), означающий «каждый столбец», вместо него можно указать какие-нибудь конкретные столбцы. В примере 8.16 показано, как выбрать только автора и название (`author` и `title`) и только название и ISBN. Результат выполнения этих команд приведен на рис. 8.9.

Пример 8.16. Две разные инструкции

```
SELECT author,title FROM classics;
SELECT title,isbn FROM classics;
```

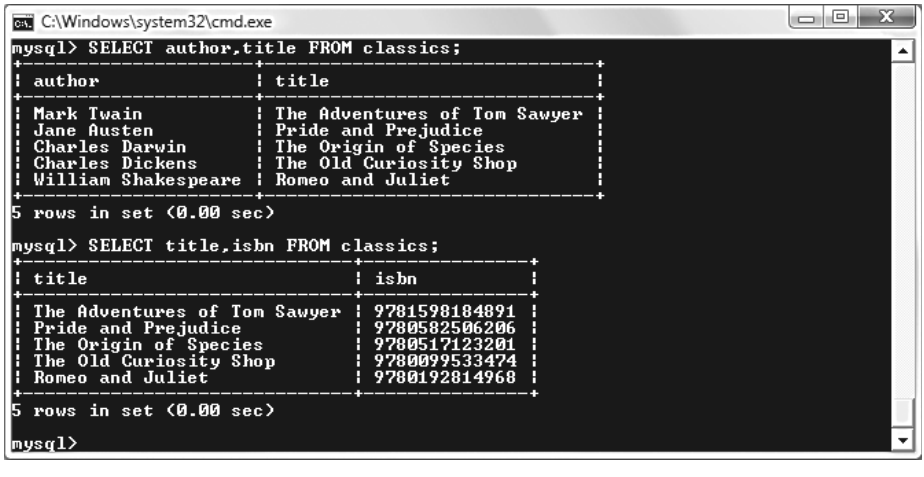

Рис. 8.9. Вывод, полученный в результате выполнения двух разных инструкций SELECT

SELECT COUNT

Другой заменой параметра *что-нибудь* является функция `COUNT`, которая может быть использована множеством способов. В примере 8.17 она отображает количество строк в таблице за счет передачи ей в качестве параметра символа звездочки (*), означающего «все строки». В соответствии с вашими ожиданиями будет возвращено число 5, поскольку в таблицу внесены сведения о пяти книгах.

Пример 8.17. Подсчет количества строк

```
SELECT COUNT(*) FROM classics;
```

SELECT DISTINCT

Спецификатор `DISTINCT` (и его синоним `DISTINCTROW`) позволяет исключать множество записей, имеющих одинаковые данные. Предположим, к примеру, что вам нужно получить список всех авторов, фигурирующих в таблице. Если просто выбрать столбец `author` из таблицы, содержащей несколько книг одного и того же автора, то будет отображен длинный список с одинаковыми именами авторов, повторяющимися снова и снова. Но за счет добавления ключевого слова `DISTINCT` можно показать каждого автора всего лишь один раз. Проверим этот спецификатор, добавив еще одну строку, в которой повторяется один из уже имеющихся авторов (пример 8.18).

Пример 8.18. Дублирование данных

```
INSERT INTO classics(author, title, category, year, isbn)
  VALUES('Charles Dickens','Little Dorrit','Fiction','1857', '9780141439969');
```

Теперь, когда Чарльз Диккенс появляется в таблице дважды, мы можем сравнить результаты использования команды `SELECT` со спецификатором `DISTINCT` и без него. В примере 8.19 и на рис. 8.10 показано, что при вводе простой команды `SELECT` Диккенс будет показан дважды, а команда со спецификатором `DISTINCT` выводит его только один раз.

Пример 8.19. Команда SELECT со спецификатором DISTINCT и без него

```
SELECT author FROM classics;
SELECT DISTINCT author FROM classics;
```

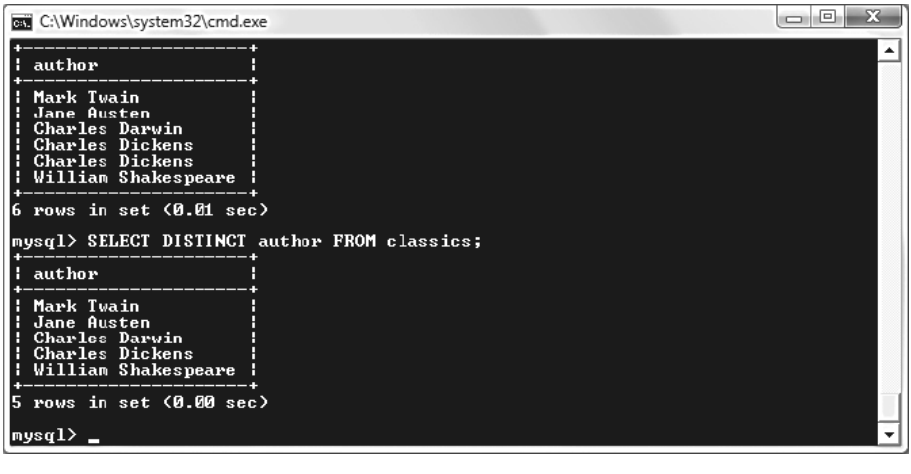

Рис. 8.10. Выбор данных с помощью команды DISTINCT и без нее

DELETE

Когда нужно удалить строку из таблицы, применяется команда `DELETE`. Ее синтаксис похож на синтаксис команды `SELECT`, она позволяет сузить диапазон удаляемой

информации до конкретной строки или строк путем использования таких спецификаторов, как WHERE и LIMIT.

Теперь, если вы вводили команду, приведенную в примере 8.18, и изучали работу спецификатора DISTINCT, нужно удалить Little Dorrit путем ввода команды, показанной в примере 8.20.

Пример 8.20. Удаление новой записи

```
DELETE FROM classics WHERE title='Little Dorrit';
```

В этом примере команда DELETE выдается для всех строк, в столбце title которых содержится строковое значение Little Dorrit.

Ключевое слово WHERE обладает большими возможностями, и очень важно, чтобы оно было правильно набрано. Ошибка может навести команду не на те строки (или вообще ни к чему не привести в том случае, если условию WHERE не будет найдено ни одного соответствия). Поэтому теперь нужно уделить немного внимания этому условию, играющему очень важную роль в языке SQL.

WHERE

Ключевое слово WHERE позволяет сузить диапазон действия запроса, возвращая только те данные, в отношении которых конкретное выражение возвращает истинное значение. За счет использования оператора равенства = код в примере 8.20 возвращает только те строки, в которых значение столбца title в точности соответствует строке Little Dorrit. В примере 8.21 показаны еще два фрагмента, в которых WHERE используется с оператором =.

Пример 8.21. Использование ключевого слова WHERE

```
SELECT author,title FROM classics WHERE author="Mark Twain";
SELECT author,title FROM classics WHERE isbn="9781598184891";
```

Применительно к нашей таблице эти две команды отобразят один и тот же результат. Но мы можем без особого труда добавить еще несколько книг Марка Твена, и тогда команда в первой строке отобразит все названия книг, принадлежащих его перу, а команда во второй строке — прежний результат (потому что, как мы знаем, ISBN имеет уникальное значение) — The Adventures of Tom Sawyer. Иными словами, поисковые операции, использующие уникальный ключ, более предсказуемы, и новые доказательства этого вы увидите позже, при рассмотрении роли уникальных и первичных ключей.

При проведении поисковых операций можно также осуществлять проверку на соответствие шаблону, для чего применяется спецификатор LIKE, позволяющий вести поиск в разных частях строк. Этот спецификатор должен использоваться с символом % до или после некоторого текста. Если его поместить до текста, это будет означать *«что-нибудь до»*, а если после текста — *«что-нибудь после»*. В примере 8.22 показаны три разных запроса, один из которых предназначен для начала строки, другой — для конца, а третий — для любого места в строке. Результат выполнения этих команд приведен на рис. 8.11.

Пример 8.22. Использование спецификатора LIKE

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";
SELECT author,title FROM classics WHERE title LIKE "%Species";
SELECT author,title FROM classics WHERE title LIKE "%and%";
```

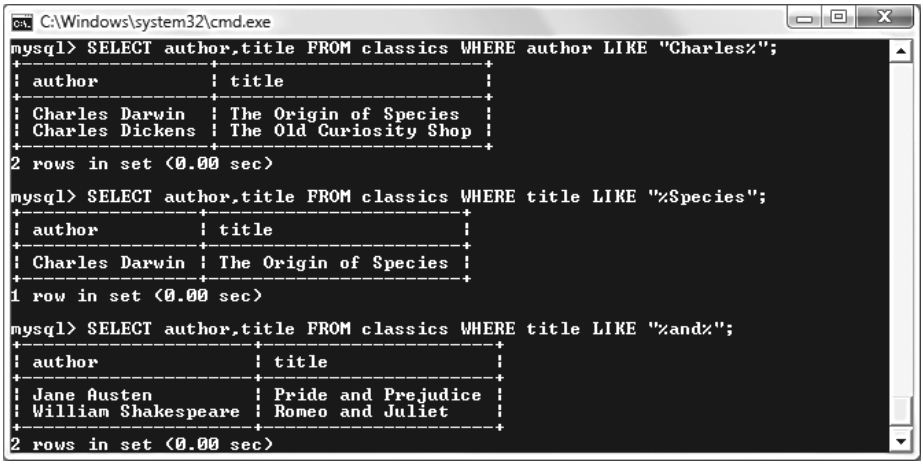

Рис. 8.11. Использование ключевого слова WHERE со спецификатором LIKE

Первая команда выведет книги, принадлежащие перу как Чарльза Дарвина, так и Чарльза Диккенса, потому что спецификатор `LIKE` был настроен на возвращение всего соответствующего строке `Charles`, за которой следует любой другой текст.

Затем будет возвращена информация о книге *The Origin of Species*, потому что есть только одна строка, столбец которой заканчивается строковым значением `Species`. И на последний запрос будет возвращена информация о книгах *Pride and Prejudice* и *Romeo and Juliet*, потому что обе записи соответствуют запросу строкового значения `and` в любом месте столбца.

Символ `%` также будет соответствовать пустому месту в той позиции, которую он занимает. Иными словами, он может соответствовать пустой строке.

LIMIT

Спецификатор `LIMIT` позволяет выбрать количество выводимых в запросе строк и место, с которого таблица начнет их возвращать. Когда передается один параметр, он указывает MySQL начать действие спецификатора с верхней части результатов и вернуть только то количество строк, которое задано этим параметром. Если передать спецификатору два параметра, то первый укажет смещение относительно начала результатов, которое MySQL должна учесть при их отображении, а второй укажет, сколько строк нужно вывести. Можно представить, что первый параметр сообщает: «Нужно пропустить это количество результатов, ведя счет сверху».

В пример 8.23 включены три команды. Первая возвращает три первые строки из таблицы. Вторая возвращает две строки, начиная с позиции 1 (пропуская первую строку). А последняя возвращает одну строку, начинающуюся с позиции 3 (пропуская первые три строки). Результаты выполнения всех трех команд показаны на рис. 8.12.

Пример 8.23. Ограничение количества возвращаемых результатов

```
SELECT author,title FROM classics LIMIT 3;
SELECT author,title FROM classics LIMIT 1,2;
SELECT author,title FROM classics LIMIT 3,1;
```

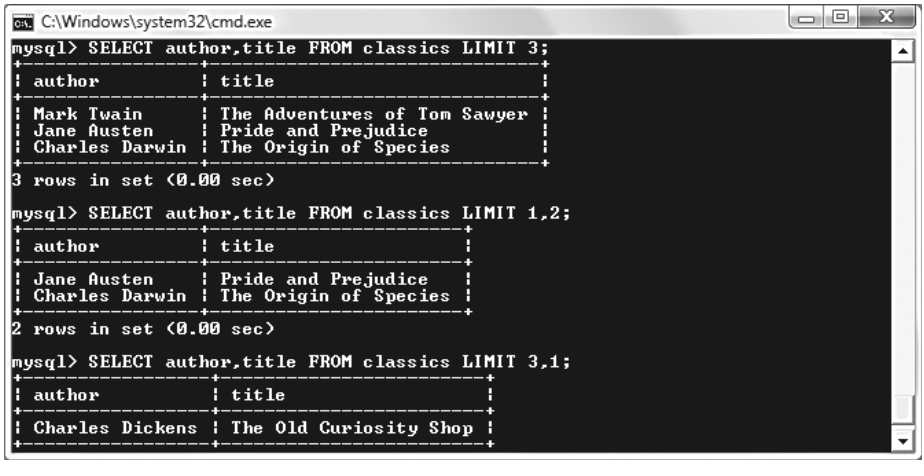

Рис. 8.12. Ограничение диапазона выводимых строк с помощью спецификатора LIMIT

Ключевое слово LIMIT требует особого внимания, поскольку смещение начинается с нулевой позиции, а количество возвращаемых строк — с единицы. Поэтому спецификатор LIMIT 1,3 означает возвращение трех строк, начиная со второй строки. Первый аргумент можно рассматривать как указание на то, сколько строк нужно пропустить, следовательно, говоря простым языком, инструкция должна звучать так: «Возвратить 3 строки, пропустив первые строки в количестве 1».

MATCH...AGAINST

Конструкция `MATCH...AGAINST` может быть применена к столбцу, для которого был создан индекс `FULLTEXT` (см. выше пункт «Создание индекса FULLTEXT»). Используя эту конструкцию, можно вести поиск, применяя в качестве критерия элементы обычного языка, как при работе с поисковыми механизмами Интернета. В отличие от конструкций `WHERE...=` или `WHERE...LIKE`, конструкция `MATCH...AGAINST` позволяет вводить в поисковый запрос несколько слов и проверять на их наличие все слова в столбцах, имеющих индекс `FULLTEXT`. Индексы `FULLTEXT` нечувствительны к регистру букв, поэтому неважно, какой именно регистр используется в ваших запросах.

Предположим, что вы добавили индекс FULLTEXT к столбцам author и title и ввели три запроса, показанные в примере 8.24. Первый из них требует вернуть любые строки, в которых содержится слово *and*. Поскольку *and* является стоповым словом, MySQL его проигнорирует и запрос всегда будет возвращать пустой набор независимо от того, что хранится в столбцах. Второй запрос требует вернуть любые строки, содержащие в любом месте и в любом порядке оба слова: *curiosity* и *shop*. Третий запрос применяет тот же вид поиска для слов *tom* и *sawyer*. Результаты выполнения этих запросов показаны на рис. 8.13.

Пример 8.24. Использование конструкции MATCH...AGAINST с индексами FULLTEXT

```
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('and');
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('curiosity shop');
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('tom sawyer');
```

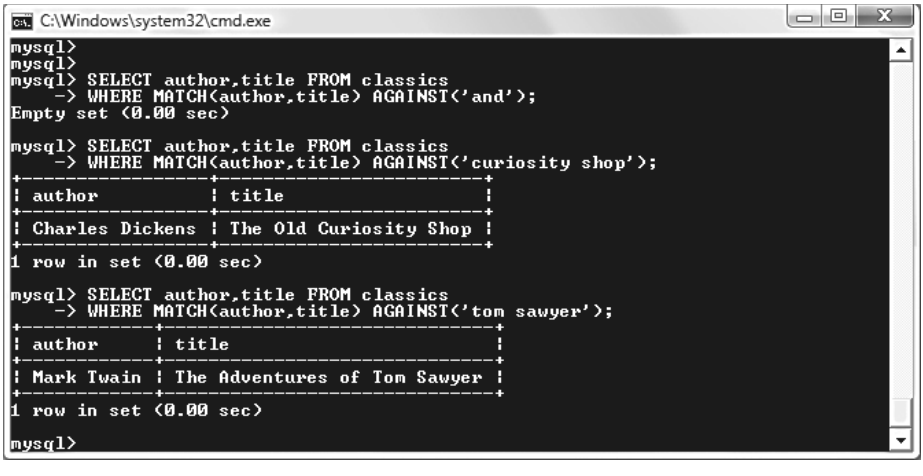

Рис. 8.13. Использование конструкции MATCH...AGAINST в индексе FULLTEXT

MATCH...AGAINST в булевом режиме

При желании придать своим запросам с конструкцией MATCH...AGAINST более широкие возможности нужно воспользоваться булевым режимом. Это изменение выражается в том, что стандартный запрос по индексу FULLTEXT ведет поиск любой комбинации искомых слов, не требуя наличия всех этих слов в тексте. Наличие отдельного слова в столбце приводит к тому, что поисковая операция возвращает строку.

Булев режим позволяет также ставить впереди искомых слов знак + или –, чтобы показать, что они должны быть включены или исключены. Если обычный булев

режим требует «искать присутствие любого из этих слов», то знак «плюс» означает, что «это слово обязательно должно присутствовать, иначе строку возвращать не нужно». Знак «минус» означает, что «этого слова быть не должно, а если оно присутствует, то строку возвращать не нужно».

В примере 8.25 показаны два запроса, использующие булев режим. Первый запрос требует вернуть все строки, в которых содержится слово *Charles* и нет слова *species*. Во втором запросе используются двойные кавычки, чтобы потребовать вернуть все строки, включающие в себя фразу *origin of*. На рис. 8.14 показаны результаты выполнения этих запросов.

Пример 8.25. Использование MATCH...AGAINST в булевом режиме

```
SELECT author,title FROM classics
  WHERE MATCH(author,title)
  AGAINST('+charles -species' IN BOOLEAN MODE);
SELECT author,title FROM classics
  WHERE MATCH(author,title)
  AGAINST('"origin of"' IN BOOLEAN MODE);
```

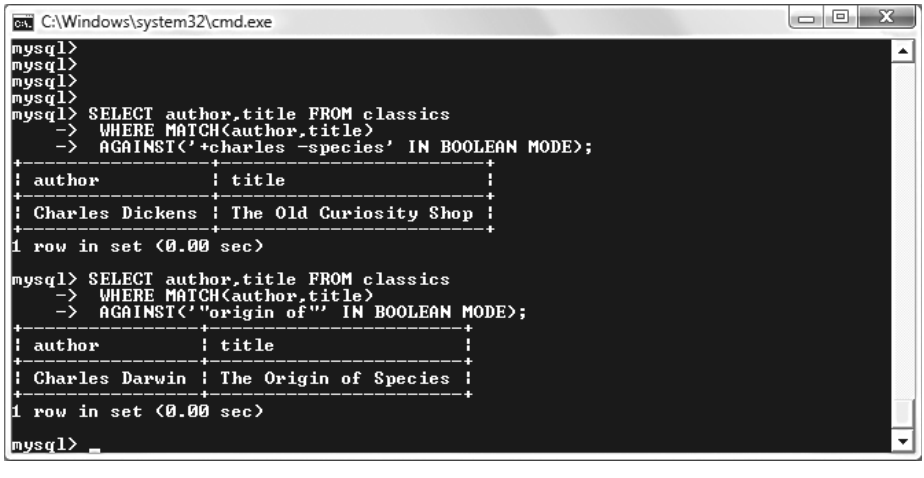

Рис. 8.14. Использование конструкции MATCH...AGAINST в булевом режиме

Как, наверное, и ожидалось, первый запрос вернет только запись о книге *The Old Curiosity Shop* Чарльза Диккенса. Запись о книге Чарльза Дарвина игнорируется, поскольку из результата должна быть исключена любая строка, содержащая слово *species*.

Во втором запросе есть кое-что интересное: частью искомой строки является стоповое слово of, но оно все же используется в поиске, поскольку двойные кавычки отменяют учет стоповых слов.

UPDATE...SET

Эта конструкция позволяет обновлять содержимое поля. Если нужно изменить содержимое одного или нескольких полей, сначала следует сузить область действия запроса до того поля или полей, которые будут подвергаться изменениям, практически тем же способом, который применялся в команде `SELECT`. В примере 8.26 показаны два разных способа использования `UPDATE...SET`. Копия экрана с результатами работы этих команд приведена на рис. 8.15.

Пример 8.26. Использование UPDATE...SET

```
UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'
  WHERE author='Mark Twain';
UPDATE classics SET category='Classic Fiction'
  WHERE category='Fiction';
```

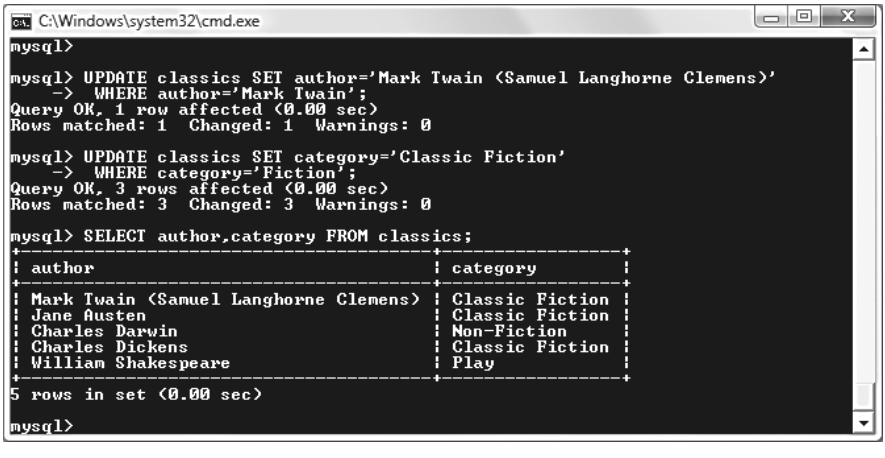

Рис. 8.15. Обновление столбцов в таблице classics

В первом запросе, действие которого затрагивает только одну строку, к литературному псевдониму *Mark Twain* добавляется настоящее имя писателя — *Samuel Langhorne Clemens*, заключенное в скобки. А вот второй запрос воздействует на три столбца, поскольку он заменяет все появления слова *Fiction* в столбце `category` термином *Classic Fiction*.

При выполнении обновления можно также воспользоваться такими уже приведенными здесь спецификаторами, как `LIMIT`, а также рассматриваемыми далее ключевыми словами `ORDER BY` и `GROUP BY`.

ORDER BY

Спецификатор `ORDER BY` позволяет отсортировать возвращаемые результаты по одному или нескольким столбцам в возрастающем или в убывающем порядке.

В примере 8.27 показаны два таких запроса, результаты работы которых можно увидеть на рис. 8.16.

Пример 8.27. Использование ORDER BY

```
SELECT author,title FROM classics ORDER BY author;
SELECT author,title FROM classics ORDER BY title DESC;
```

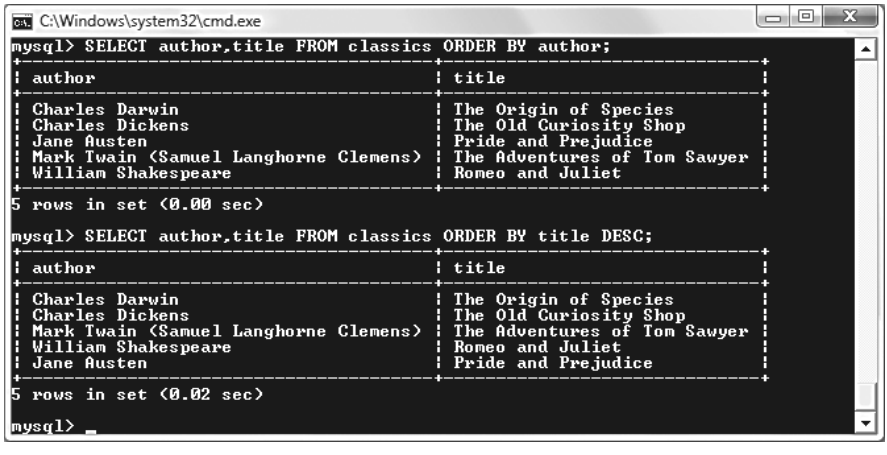

Рис. 8.16. Сортировка результатов запроса

Первый запрос возвращает издания, отсортированные по авторам в возрастающем алфавитном порядке (этот режим используется по умолчанию), а второй возвращает их отсортированными по названию в убывающем порядке.

Если нужно отсортировать все столбцы по авторам, а затем в убывающем порядке по году издания (чтобы сначала стояли самые последние), нужно ввести следующий запрос:

```
SELECT author,title,year FROM classics ORDER BY author,year DESC;
```

Здесь показано, что каждый спецификатор сортировки по возрастанию и по убыванию применяется к отдельному столбцу. Ключевое слово `DESC` применяется только к столбцу, который указан перед ним, — `year`. Поскольку для столбца `author` разрешено использовать порядок сортировки, применяемый по умолчанию, этот столбец сортируется в возрастающем порядке. Можно также указать порядок сортировки этого столбца по возрастанию и в явном виде, в результате будут получены аналогичные результаты:

```
SELECT author,title,year FROM classics ORDER BY author ASC,year DESC;
```

GROUP BY

Как и при использовании `ORDER BY`, можно сгруппировать результаты, возвращаемые запросом, с помощью спецификатора `GROUP BY`, который больше всего подходит для

извлечения информации о группе данных. Например, если нужно узнать, сколько изданий каждой категории присутствует в таблице `classics`, можно ввести запрос:

```
SELECT category,COUNT(author) FROM classics GROUP BY category;
```

который вернет следующую информацию:

```
+----------------+---------------+
| category       | COUNT(author) |
+----------------+---------------+
| Classic Fiction |             3 |
| Non-Fiction    |             1 |
| Play           |             1 |
+----------------+---------------+
3 rows in set (0.00 sec)
```

Объединение таблиц

Управление несколькими таблицами, содержащими различные виды информации в одной базе данных, считается вполне обычным делом. Рассмотрим, к примеру, таблицу клиентов — `customers`, для которой нужно обеспечить возможность использования перекрестных ссылок с приобретенными книгами из таблицы `classics`. Чтобы создать эту новую таблицу и поместить в нее информацию о трех клиентах и их покупках, введите команды из примера 8.28. Результаты показаны на рис. 8.17.

Пример 8.28. Создание и заполнение таблицы customers

```
CREATE TABLE customers (
  name VARCHAR(128),
  isbn VARCHAR(13),
  PRIMARY KEY (isbn)) ENGINE InnoDB;
INSERT INTO customers(name,isbn)
  VALUES('Joe Bloggs','9780099533474');
INSERT INTO customers(name,isbn)
  VALUES('Mary Smith','9780582506206');
INSERT INTO customers(name,isbn)
  VALUES('Jack Wilson','9780517123201');
SELECT * FROM customers;
```

Существует также быстрый способ для вставки сразу нескольких строк данных, как в примере 8.28, позволяющий заменить три отдельных запроса INSERT INTO одним, в котором перечисляются вставляемые данные, отделенные друг от друга запятыми:

```
INSERT INTO customers(name,isbn) VALUES
  ('Joe Bloggs','9780099533474'),
  ('Mary Smith','9780582506206'),
  ('Jack Wilson','9780517123201');
```

Разумеется, в настоящей таблице, содержащей сведения о покупателях, будут присутствовать также адреса, номера телефонов, адреса электронной почты и т. д., но на данном этапе изучения они для нас не представляют интереса.

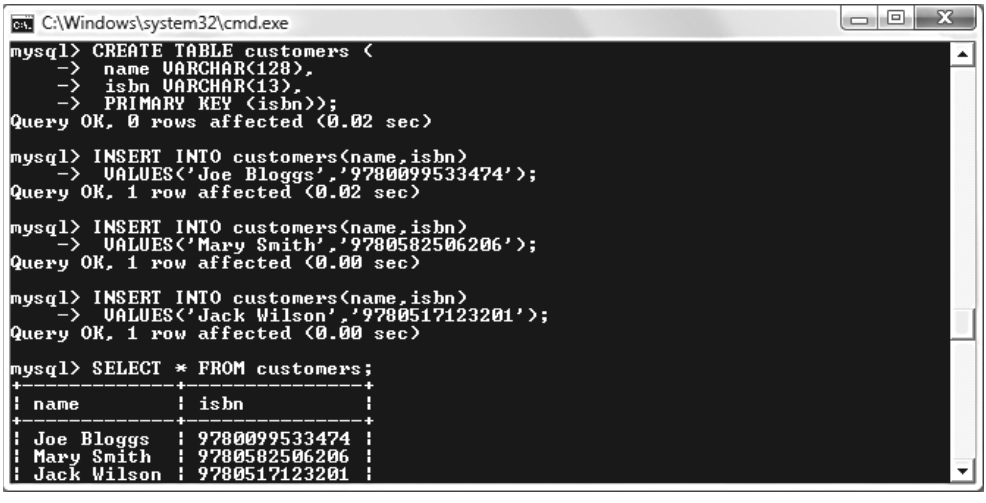

Рис. 8.17. Создание таблицы customers

При создании новой таблицы следует обратить внимание на то, что у нее есть кое-что общее с таблицей `classics`: столбец под названием `isbn`. Поскольку его предназначение в обеих таблицах совпадает (ISBN всегда является ссылкой на одну и ту же книгу), этот столбец можно использовать для связывания двух таблиц вместе в едином запросе, как в примере 8.29.

Пример 8.29. Объединение двух таблиц в одном запросе

```
SELECT SELECT name,author,title FROM customers,classics
  WHERE customers.isbn=classics.isbn;
```

В результате будет выведена следующая информация:

```
+-------------+----------------+-----------------------+
| name        | author         | title                 |
+-------------+----------------+-----------------------+
| Joe Bloggs  | Charles Dickens | The Old Curiosity Shop |
| Mary Smith  | Jane Austen     | Pride and Prejudice    |
| Jack Wilson | Charles Darwin  | The Origin of Species  |
+-------------+----------------+-----------------------+
3 rows in set (0.00 sec)
```

Видите, как этот запрос искусно связал вместе обе таблицы, чтобы продемонстрировать книги из таблицы `classics`, приобретенные покупателями из таблицы `customers`?

NATURAL JOIN

Используя конструкцию `NATURAL JOIN`, можно сократить количество вводимого текста и сделать запросы немного более понятными. В этом виде объединения участвуют две таблицы, в которых автоматически объединяются столбцы с одинаковыми

именами. Для получения тех же результатов, что и в примере 8.29, можно ввести следующий запрос:

```
SELECT name,author,title FROM customers NATURAL JOIN classics;
```

JOIN...ON

Если нужно указать столбец, по которому следует объединить две таблицы, используется конструкция `JOIN...ON`. Благодаря ей можно получить те же результаты, что и в примере 8.29:

```
SELECT name,author,title FROM customers
  JOIN classics ON customers.isbn=classics.isbn;
```

Использование ключевого слова AS

Можно сократить количество вводимого текста и улучшить читаемость запроса за счет создания псевдонимов с помощью ключевого слова `AS`. После имени таблицы нужно просто поставить `AS`, а затем используемый псевдоним. Следующий код идентичен по своей работе коду, приведенному в примере 8.29:

```
SELECT name,author,title from
  customers AS cust, classics AS class WHERE cust.isbn=class.isbn;
```

Результат выполнения этой операции имеет следующий вид:

```
+-------------+-----------------+-----------------------+
| name        | author          | title                 |
+-------------+-----------------+-----------------------+
| Joe Bloggs  | Charles Dickens | The Old Curiosity Shop |
| Mary Smith  | Jane Austen     | Pride and Prejudice    |
| Jack Wilson | Charles Darwin  | The Origin of Species  |
+-------------+-----------------+-----------------------+
3 rows in set (0.00 sec)
```

Ключевое слово `AS` можно также использовать для переименования столбца (независимо от того, объединяются таблицы или нет), например:

```
SELECT name AS customer FROM customers ORDER BY customer;
```

В качестве результата будут выведены следующие данные:

```
+-------------+
| customer    |
+-------------+
| Jack Wilson |
| Joe Bloggs  |
| Mary Smith  |
+-------------+
3 rows in set (0.00 sec)
```

Псевдонимы особенно полезны в длинных запросах, содержащих множественные ссылки на одни и те же имена таблиц.

Использование логических операторов

Для дальнейшего сужения пространства выбора в запросах MySQL, использующих ключевое слово `WHERE`, можно также задействовать логические операторы `AND`, `OR` и `NOT`. В примере 8.30 показаны варианты применения каждого из них, но их можно использовать в любых сочетаниях.

Пример 8.30. Использование логических операторов

```
SELECT author,title FROM classics WHERE
  author LIKE "Charles%" AND author LIKE "%Darwin";
SELECT author,title FROM classics WHERE
  author LIKE "%Mark Twain%" OR author LIKE "%Samuel Langhorne Clemens%";
SELECT author,title FROM classics WHERE
  author LIKE "Charles%" AND author NOT LIKE "%Darwin";
```

Первый запрос выбран потому, что Чарльз Дарвин может фигурировать в некоторых строках под своим полным именем — Чарльз Роберт Дарвин. А запрос возвращает сведения о книгах, для которых значение столбца `author` начинается с `Charles` и заканчивается `Darwin`. Второй запрос ищет книги, принадлежащие перу Марка Твена, используя для этого либо литературный псевдоним — Mark Twain, либо настоящее имя писателя — Samuel Langhorne Clemens. Третий запрос возвращает книги авторов, чье имя Charles, а фамилия не Darwin.

Функции MySQL

Стремление применять функции MySQL при таком обилии достаточно мощных функций PHP может вызвать недоумение. Ответ предельно прост: функции MySQL работают с данными непосредственно в самой базе. А при использовании PHP приходится сначала извлекать строку данных из MySQL, выполнять обработку, а затем выдавать первоначально задуманный запрос к базе данных.

Применение встроенных функций MySQL не только существенно сокращает время обработки сложных запросов, но и упрощает сами запросы. При желании подробные сведения обо всех доступных строковых функциях и функциях даты и времени можно найти в документации по следующим адресам: http://tinyurl.com/mysqlstrfuncs и http://tinyurl.com/mysqldatefuncs.

Первоначальные сведения о наборе наиболее востребованных функций изложены в приложении Г.

Работа с MySQL через phpMyAdmin

Для работы с MySQL, безусловно, важно изучить все представленные здесь основные команды и особенности их работы, но после того, как они уже изучены, для управления базами данных и таблицами будет намного проще и быстрее использовать программу phpMyAdmin.

Для этого при условии, что пакет программ AMPPS установлен в соответствии с инструкциями главы 2, наберите следующую команду, чтобы открыть программу (рис. 8.18):

```
http://localhost/phpmyadmin
```

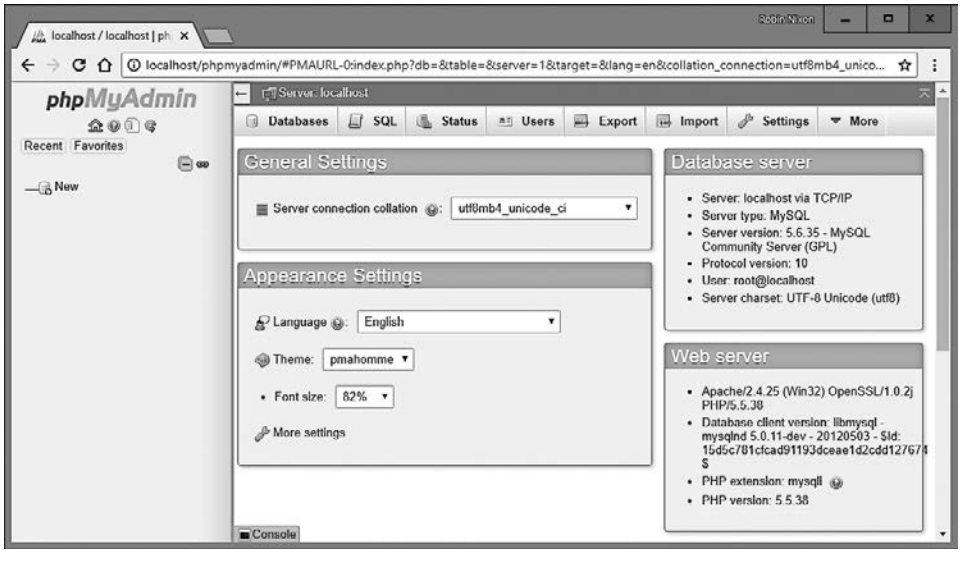

Рис. 8.18. Главный экран phpMyAdmin

На левой панели основного экрана phpMyAdmin щелчком можно выбрать любые таблицы, с которыми нужно поработать (хотя сначала, если такие таблицы отсутствуют, их придется создать). Для создания новой базы данных можно также щелкнуть на пункте меню New (Создать).

Находясь в окне этой программы, можно совершать все основные операции, например создавать новые базы данных, добавлять таблицы, создавать индексы и т. д. Дополнительные сведения о программе phpMyAdmin можно найти в документации по адресу https://docs/phpmyadmin.net.

Если вы совместно со мной прорабатывали все примеры, приведенные в данной главе, то я вас поздравляю с тем, что вы смогли выдержать весьма долгое путешествие, пройдя весь путь от изучения способа создания базы MySQL через выдачу

сложных запросов с задействованием сразу нескольких таблиц до использования булевых операторов и применения различных квалификаторов MySQL.

В следующей главе мы приступим к рассмотрению подходов к разработке рациональных баз данных, более совершенных SQL-технологий, а также функций и транзакций MySQL.

Вопросы

1. Для чего нужна точка с запятой в запросах MySQL?
2. Какие команды используются для просмотра доступных баз данных или таблиц?
3. Как на локальном хосте создается новый пользователь MySQL с именем `newuser` и паролем `newpass`, которому открыт доступ ко всему содержимому базы данных `newdatabase`?
4. Как просмотреть структуру таблицы?
5. Для чего нужен индекс в MySQL?
6. Какие преимущества дает индекс `FULLTEXT`?
7. Что такое стоповое слово?
8. Оба спецификатора, и `SELECT DISTINCT`, и `GROUP BY`, приводят к отображению только одной строки для каждого значения в столбце, даже если такое значение имеют несколько строк. В чем основное различие между `SELECT DISTINCT` и `GROUP BY`?
9. Как можно с помощью инструкции `SELECT...WHERE` вернуть только те строки, в которых в каком-нибудь месте столбца `author` таблицы `classics`, используемой в этой главе, содержится слово `Langhorne`?
10. Что должно быть определено в двух таблицах, чтобы появилась возможность их объединения?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

9 Освоение MySQL

В главе 8 была заложена хорошая основа для работы с реляционными базами данных с использованием SQL. Были рассмотрены создание баз данных и включаемых в них таблиц, а также вставка, поиск, изменение и удаление данных.

Теперь, вооружившись этими знаниями, нужно изучить проектирование баз данных, работать с которыми можно максимально быстро и эффективно. Например, научиться принимать решения о том, какие данные в какие таблицы помещать. За годы существования баз данных были разработаны руководства, следуя которым можно обеспечить эффективную работу с ними и возможность их масштабирования по мере наполнения все новыми и новыми данными.

Проектирование базы данных

Перед тем как создавать базу данных, очень важно ее удачно спроектировать, в противном случае, скорее всего, придется возвращаться назад и изменять ее структуру, разбивая одни и объединяя другие таблицы и перемещая различные столбцы из таблицы в таблицу с целью достижения рациональных связей, которыми MySQL будет легче воспользоваться.

Для начала было бы неплохо сесть за стол с листом бумаги и карандашом и набросать подборку тех запросов, которые, на ваш взгляд, чаще всего будут нужны вам и вашим пользователям. Для базы данных книжного интернет-магазина могут быть записаны следующие вопросы.

- Сколько авторов, книг и покупателей имеется в базе данных?
- Каким автором написана та или иная книга?
- Какие книги написаны тем или иным автором?
- Какая книга продается по самой высокой цене?
- Какая книга является лидером продаж?

- Какие книги не покупались в этом году?
- Какие книги приобретены тем или иным покупателем?
- Какие книги были приобретены вместе с какими-нибудь другими книгами?

Разумеется, к такой базе данных может быть сделано и множество других запросов, но даже эта подборка даст вам представление о том, как следует спланировать структуру таблиц.

Например, книги и номера ISBN должны быть, наверное, скомбинированы в одной таблице, поскольку они тесно взаимосвязаны (некоторые тонкости этого вопроса будут исследованы чуть позже). В отличие от этого, книги и покупатели должны находиться в разных таблицах, поскольку они слабо взаимосвязаны. Покупатель может купить любую книгу и даже несколько экземпляров одной и той же книги, а книга может быть приобретена многими покупателями и может не привлечь внимания еще большего количества потенциальных покупателей.

Когда планируется множество поисковых операций по каким-нибудь столбцам, зачастую их лучше всего поместить в общую таблицу. А когда какие-то элементы слабо связаны друг с другом, их лучше поместить в отдельные таблицы.

Если принять во внимание эти элементарные правила, то можно предположить, что для удовлетворения всех этих запросов нам понадобятся как минимум три таблицы.

- `authors` (авторы). Предполагается большое количество поисков по авторам, многие из которых сотрудничали при написании книг, а значит, будут показаны вместе. Оптимальных результатов поиска можно добиться, если о каждом авторе будет дана вся относящаяся к нему информация, следовательно, нам нужна таблица авторов — `authors`.
- `books` (книги). Многие книги появляются в различных изданиях. Иногда у них разные издатели, а иногда разные книги имеют одно и то же название. Связи между книгами и авторами настолько сложны, что для книг нужна отдельная таблица.
- `customers` (покупатели). Причина, по которой покупатели должны находиться в собственной таблице, еще более прозрачна — покупатели могут приобрести любую книгу любого автора.

Первичные ключи: ключи к реляционным базам данных

Используя возможности реляционных баз данных, мы можем задавать всю информацию для каждых автора, книги и покупателя в одном и том же месте. Очевидно, что нас интересуют связи между ними, например, кто написал каждую книгу и кто ее приобрел, и мы можем сохранить эту информацию лишь за счет создания связей между тремя таблицами. Я покажу вам основные принципы, которые нетрудно будет усвоить на практике.

Секрет заключается в присваивании каждому автору уникального идентификатора. То же самое делается для каждой книги и каждого покупателя. Смысл всего этого был объяснен в предыдущей главе: нам нужен *первичный ключ*. Для книги имеет смысл использовать в этом качестве номер ISBN, хотя вам, может быть, придется столкнуться с несколькими одинаковыми книгами, имеющими разные номера ISBN. Авторам и покупателям можно просто назначить произвольные ключи, имеющие свойство автоприращения — `AUTO_INCREMENT`, что, судя по предыдущей главе, делается весьма просто.

Проще говоря, каждая таблица будет спроектирована вокруг какого-нибудь объекта, в котором, скорее всего, будет вестись интенсивный поиск, — в данном случае вокруг автора, книги или покупателя, и этот объект должен иметь первичный ключ. В качестве ключа не следует выбирать ничего, что могло бы иметь одинаковое значение для разных объектов. Ситуация с номером ISBN является тем самым редким случаем, когда сама издательская индустрия предоставила нам первичный ключ, который можно считать уникальным для каждого продукта. В большинстве случаев для этих целей следует создавать произвольный ключ, использующий свойство `AUTO_INCREMENT`.

Нормализация

Процесс распределения данных по таблицам и создания первичных ключей называется *нормализацией*. Основная цель нормализации — обеспечить, чтобы каждая порция информации появлялась в базе данных только один раз. Дублирование данных приводит к крайне неэффективной работе, поскольку неоправданно увеличивает объем базы данных и замедляет тем самым доступ к информации. Еще важнее то, что дубликаты создают большой риск обновления только одной строки продублированных данных, приводят к несогласованности в базе данных, являющейся потенциальным источником серьезных ошибок.

Если, к примеру, названия книг перечисляются и в таблице авторов, и в таблице книг и возникает необходимость исправить опечатку в названии, нужно будет вести поиск в обеих таблицах и вносить одинаковые изменения везде, где встречаются названия книг. Лучше хранить названия в одном месте, а в других местах использовать номер ISBN.

В процессе разбиения базы данных на несколько таблиц важно не зайти слишком далеко и не создать больше таблиц, чем требуется, что может также привести к неэффективности конструкции и замедлению доступа к данным.

К счастью, изобретатель реляционной модели Эдгар Кодд проанализировал понятие нормализации и разбил его на три отдельные схемы, названные *первой*, *второй* и *третьей нормальными формами*. Если вносить в базу данных изменения,

последовательно удовлетворяющие каждой из этих форм, то будет обеспечена оптимальная сбалансированность базы данных, способствующая достижению быстрого доступа и использованию минимального объема оперативной и дисковой памяти.

Чтобы понять, как выполняется нормализация, начнем с весьма несуразной базы данных, представленной в табл. 9.1, в которой имеется одна таблица, содержащая все сведения об авторах, книгах и вымышленных покупателях. Ее можно рассматривать в качестве первой попытки создать таблицу, отслеживающую, кто из покупателей какие книги заказал. Неэффективность такой конструкции не вызывает сомнений, поскольку данные повсеместно дублируются (дубликаты в таблице выделены полужирным и курсивным шрифтом), но это всего лишь наша отправная точка.

Таблица 9.1. Крайне неэффективная конструкция таблицы базы данных

Author 1 (Автор 1)	Author 2 (Автор 2)	Title (Название)	ISBN	Price (Цена)	Customer name (Имя покупателя)	Customer address (Адрес покупателя)	Purchase date (Дата покупки)
David Sklar	*Adam Trachtenberg*	*PHP Cookbook*	*0596101015*	*44,99*	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Danny Goodman		Dynamic HTML	0596527403	59,99	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Dec 19 2008**
Hugh E. Williams	David Lane	PHP and MySQL	0596005436	44,95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
David Sklar	*Adam Trachtenberg*	*PHP Cookbook*	*0596101015*	*44,99*	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Dec 19 2008**
Rasmus Lerdorf	Kevin Tatroe & Peter MacIntyre	Programming PHP	0596006815	39,99	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

В следующих трех разделах мы проанализируем эту конструкцию базы данных, и вы увидите, как она может быть улучшена за счет удаления продублированных записей и разбиения одной таблицы на несколько более практичных таблиц, в каждой из которых будет храниться один тип данных.

Первая нормальная форма

Чтобы база данных соответствовала *первой нормальной форме*, она должна выполнять три требования.

- В ней не должно быть повторяющихся столбцов, содержащих одни и те же типы данных.
- Каждая позиция пересечения столбца и строки должна содержать только одно значение.
- Для уникальной идентификации каждой строки должен быть первичный ключ.

Рассматривая по порядку эти требования, вы заметите, что в столбцы `Author 1` и `Author 2` заложены повторяющиеся типы данных. Итак, у нас уже появился тот самый столбец, который следует поместить в отдельную таблицу, поскольку повторяющийся столбец `Author` противоречит правилу 1.

Второе несоответствие связано с тем, что для последней книги, Programming PHP, указаны три автора. Я считаю, что использование одного и того же столбца `Author 2` для имен двух авторов — Kevin Tatroe и Peter MacIntyre — нарушает правило 2. Это еще одна причина перемещения всех сведений об авторах в отдельную таблицу.

А вот правило 3 здесь соблюдается, потому что первичный ключ в столбце ISBN уже создан.

В табл. 9.2 показаны результаты перемещения столбцов авторов из табл. 9.1. Теперь здесь уже меньше беспорядка, хотя все еще остаются дубликаты, выделенные полужирным и курсивным шрифтом.

Таблица 9.2. Результаты удаления столбцов из табл. 9.1

Title (Название)	ISBN	Price (Цена)	Customer name (Имя покупателя)	Customer address (Адрес покупателя)	Purchase date (Дата покупки)
PHP Cookbook	*0596101015*	*44,99*	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Dynamic HTML	0596527403	59,99	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Dec 19 2008**
PHP and MySQL	0596005436	44,95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
PHP Cookbook	*0596101015*	*44,99*	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Dec 19 2008**
Programming PHP	0596006815	39,99	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

Новая таблица `Authors`, приведенная в табл. 9.3, проста по структуре и имеет довольно небольшой размер. В ней просто перечисляются номера ISBN, принадлежащие книге с тем или иным названием, рядом с которыми размещается автор. Если у книги более одного автора, соавторы получают собственную строку. Поначалу эта таблица может показаться несуразной, потому что по ней нельзя понять сразу, кто из авторов какую книгу написал. Но не стоит переживать: MySQL может быстро проинформировать вас об этом. Для этого нужно лишь сообщить, для какой именно книги нужна такая информация, и MySQL воспользуется ее ISBN для поиска в таблице авторов, что займет какие-то миллисекунды.

Таблица 9.3. Новая таблица Authors

ISBN	**Author (Автор)**
0596101015	David Sklar
0596101015	Adam Trachtenberg
0596527403	Danny Goodman
0596005436	Hugh E. Williams
0596005436	David Lane
0596006815	Rasmus Lerdorf
0596006815	Kevin Tatroe
0596006815	Peter MacIntyre

Как было отмечено ранее, ISBN будет служить в качестве первичного ключа для таблицы книг — `Books`, когда дело дойдет до ее создания. Я упомянул об этом, чтобы подчеркнуть, что ISBN тем не менее не является первичным ключом для таблицы `Authors`. При практической разработке для таблицы `Authors` также нужно создать первичный ключ, обеспечивающий уникальную идентификацию авторов.

Поэтому для таблицы `Authors` столбец `ISBN` является простым столбцом, для которого в целях ускорения поиска может быть, наверное, создан ключ, но этот ключ будет уже не *первичным*. Фактически в этой таблице он и *не может* быть первичным, так как не обладает уникальностью: один и тот же номер ISBN появляется по нескольку раз в тех случаях, когда над одной книгой работали несколько авторов.

Поскольку мы будем использовать такой ключ для связи авторов с книгами в другой таблице, этот столбец называется *внешним* ключом.

Ключи (которые также называются индексами) имеют в MySQL несколько предназначений. Основной целью создания ключа является ускорение поиска. В главе 8 были показаны примеры, в которых ключи использовались в условиях WHERE для осуществления поиска. Но ключ можно применять и для уникальной идентификации элемента. Таким образом, уникальный ключ часто задействуется в качестве первичного ключа в одной таблице и в качестве внешнего ключа для связи строк этой таблицы со строками другой.

Вторая нормальная форма

Первая нормальная форма позволяет разобраться с продублированными данными (или избыточностью) в нескольких столбцах. *Вторая нормальная форма* имеет отношение только к решению проблемы избыточности в нескольких строках. Чтобы можно было привести базу данных ко второй нормальной форме, ваши таблицы должны уже иметь первую нормальную форму. Как только это будет сделано, для определения столбцов, данные в которых повторяются в разных местах, и последующего их перемещения в собственные таблицы применяется вторая нормальная форма.

Еще раз посмотрим на табл. 9.2. Видите, Darren Ryder приобрел две книги, и поэтому его данные продублированы. Это говорит о том, что столбцы, имеющие отношение к покупателю (`Customer name` и `Customer address`), следует переместить в их собственные таблицы. В табл. 9.4 показан результат удаления двух столбцов, касающихся покупателя, из табл. 9.2.

Таблица 9.4. Новая таблица Titles

ISBN	Title (Название)	Price (Цена)
0596101015	PHP Cookbook	44,99
0596527403	Dynamic HTML	59,99
0596005436	PHP and MySQL	44,95
0596006815	Programming PHP	39,99

Таким образом, в табл. 9.4 остались только столбцы номера `ISBN`, названия (`Title`) и цены (`Price`) для четырех уникальных книг, поэтому теперь это эффективная в использовании и независимая таблица, удовлетворяющая требованиям как первой, так и второй нормальной формы. Попутно мы справились с сокращением информации до уровня тех данных, которые имеют непосредственное отношение к книгам с определенными названиями. Эта таблица может также включать год издания, количество страниц, количество переизданий и т. д., поскольку все эти данные имеют тесную связь друг с другом. Единственное правило гласит: сюда нельзя помещать столбцы, которые могут содержать несколько значений для одной книги, поскольку тогда нам придется указывать одну и ту же книгу в нескольких строках, нарушая таким образом правила второй нормальной формы. К примеру, к нарушениям на этом этапе нормализации может привести восстановление столбца авторов.

Но изучая извлеченные столбцы, относящиеся к покупателям, которые теперь показаны в табл. 9.5, можно заметить, что эта таблица все же требует дополнительной нормализации, поскольку сведения о покупателе Darren Ryder по-прежнему продублированы. Следует также признать, что правило 2 первой нормальной формы (все столбцы должны содержать только одно значение) здесь не соблюдается,

поскольку адресные данные нужно разбить на отдельные столбцы для адреса — `Address`, города — `City`, штата — `State` и почтового индекса — `Zip`.

Таблица 9.5. Сведения о покупателях из табл. 9.2

ISBN	Customer name (Имя покупателя)	Customer address (Адрес покупателя)	Purchase date (Дата покупки)
0596101015	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
0596527403	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
0596005436	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
0596101015	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
0596006815	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

Нужно продолжить разбиение этой таблицы, чтобы обеспечить однократный ввод сведений, касающихся покупателя. Поскольку ISBN не относится к таким сведениям и не может использоваться в качестве первичного ключа для идентификации покупателей (или авторов), должен быть создан новый ключ.

В табл. 9.6 показан результат нормализации таблицы `Customers` в соответствии с правилами первой и второй нормальных форм. Теперь у каждого покупателя есть уникальный номер покупателя, который называется `CustNo`, используется в качестве первичного ключа и который, скорее всего, был создан с использованием свойства автоприращения — `AUTO_INCREMENT`. Все составляющие адресов были также распределены по разным столбцам, для того чтобы упростить их поиск и обновление.

Таблица 9.6. Новая таблица Customers

CustNo (Номер покупателя)	Name (Имя)	Address (Адрес)	City (Город)	State (Штат)	Zip (Почтовый индекс)
1	Emma Brown	1565 Rainbow Road	Los Angeles	CA	90014
2	Darren Ryder	4758 Emily Drive	Richmond	VA	23219
3	Earl B. Thurston	862 Gregory Lane	Frankfort	KY	40601
4	David Miller	3647 Cedar Lane	Waltham	MA	02154

В то же время для нормализации табл. 9.6 необходимо было удалить информацию о покупках, поскольку в противном случае в ней встречались бы одни и те же

сведения о покупателе для каждой купленной им книги. Вместо этого данные о покупках теперь помещены в новую таблицу `Purchases` (табл. 9.7).

Таблица 9.7. Новая таблица Purchases

CustNo (Номер покупателя)	ISBN	Date (Дата)
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

Здесь в качестве ключа, связывающего вместе таблицы `Customers` и `Purchases`, опять используется столбец `CustNo` из табл. 9.6. Поскольку здесь повторно появляется столбец `ISBN`, эта таблица может быть связана также с таблицами `Authors` и `Titles`.

Столбец `CustNo` может быть полезен в качестве ключа (но только не первичного) в таблице `Purchases`: один и тот же покупатель может приобрести несколько книг (и даже несколько экземпляров одной и той же книги), поэтому столбец `CustNo` не может служить первичным ключом. Фактически у таблицы `Purchases` вообще нет первичного ключа. И это вполне нормально, поскольку потребностей в отслеживании уникальных покупок не предвидится. Если один покупатель приобретет два экземпляра одной и той же книги, то придется смириться с двумя строками, содержащими одну и ту же информацию. Для упрощения поиска можно определить в качестве ключей, только не первичных, оба столбца: `CustNo` и `ISBN`.

Теперь у нас четыре таблицы, на одну больше, чем те три, которые потребовались бы согласно нашим первоначальным прикидкам. Мы пришли к этому решению в процессе нормализации, методически следуя правилам первой и второй нормальных форм, которые однозначно позволили выявить необходимость существования дополнительной четвертой таблицы под названием Purchases (Покупки).

У нас есть следующие таблицы: `Authors` (см. табл. 9.3), `Titles` (см. табл. 9.4), `Customers` (см. табл. 9.6) и `Purchases` (см. табл. 9.7). Каждая из них может быть связана с любой другой с помощью либо ключа `CustNo`, либо ключа `ISBN`.

Например, чтобы посмотреть, какие книги приобрел Darren Ryder, их можно поискать через табл. 9.6, `Customers`, где мы увидим, что `CustNo` этого покупателя 2. Теперь, имея этот номер, можно перейти к табл. 9.7, `Purchases`, найти там столбец `ISBN` и увидеть, что клиент приобрел книги с номерами 0596527403 и 0596101015 19 декабря 2008 года. Подобные поиски кажутся утомительными для человека, но не составляют ни малейшего труда для MySQL.

Определить названия этих книг можно, обратившись затем к табл. 9.4, `Titles`, и увидев, что это книги Dynamic HTML и PHP Cookbook. Если нужно узнать авторов этих книг, следует воспользоваться номерами ISBN, которые теперь нужно найти в табл. 9.3, `Authors`. Станет понятно, что книгу с номером ISBN 0596527403, Dynamic HTML, написал Danny Goodman, а авторы книги с номером ISBN 0596101015, PHP Cookbook — David Sklar и Adam Trachtenberg.

Третья нормальная форма

После приведения в соответствие первой и второй нормальным формам база данных приобрела подходящий вид, и в дальнейшем вам, возможно, уже не придется что-либо в ней изменять. Но если применить к базе данных более суровые требования, то можно довести ее до соответствия правилам *третьей нормальной формы*, которые требуют, чтобы данные, не имеющие непосредственной зависимости от первичного ключа, но имеющие зависимость от другого значения в таблице, были также перемещены в отдельные таблицы в соответствии с тем, к чему они имеют отношение.

Например, касательно табл. 9.6, `Customers`, можно утверждать, что ключи `State`, `City` и `Zip` не имеют прямого отношения к каждому покупателю, поскольку эти же составляющие будут присутствовать в адресах многих других людей. Но они напрямую связаны друг с другом тем, что улица в адресе — `Address` относится к городу — `City`, а город относится к штату — `State`.

Поэтому, чтобы соблюсти правила третьей нормальной формы для табл. 9.6, ее нужно разбить на табл. 9.8–9.11.

Таблица 9.8. Таблица Customers, соответствующая правилам третьей нормальной формы

CustNo (Номер покупателя)	Name (Имя)	Address (Адрес)	Zip (Почтовый индекс)
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	02154

Таблица 9.9. Таблица Zip codes, соответствующая правилам третьей нормальной формы

Zip (Почтовый индекс)	CityID (Идентификатор города)
90014	1234
23219	5678
40601	4321
02154	8765

Таблица 9.10. Таблица Cities, соответствующая правилам третьей нормальной формы

CityID (Идентификатор города)	Name (Название)	StateID (Идентификатор штата)
1234	Los Angeles	5
5678	Richmond	46
4321	Frankfort	17
8765	Waltham	21

Таблица 9.11. Таблица States, соответствующая правилам третьей нормальной формы

StateID (Идентификатор штата)	Name (Название)	Abbreviation (Аббревиатура)
5	California	CA
46	Virginia	VA
17	Kentucky	KY
21	Massachusetts	MA

Ну и как пользоваться этим набором из четырех таблиц вместо одной табл. 9.6? В табл. 9.8 нужно найти zip-код, затем в табл. 9.9 — соответствующий ему город. Располагая этой информацией, в табл. 9.10 можно найти название города, а затем — идентификатор штата — `StateID`, который можно использовать в табл. 9.11 для поиска его названия.

Хотя подобное подстраивание под третью нормальную форму может показаться излишним, у него могут быть и свои преимущества. Например, взгляните на табл. 9.11, в которую удалось включить как название, так и двухбуквенную аббревиатуру штата. Сюда же при желании можно включить данные о количестве жителей и другие демографические сведения.

Таблица 9.10 может также содержать более локализованную демографическую информацию, которая окажется полезной вам и (или) вашим покупателям. Разбивая эти данные на части, вы можете упростить обслуживание своей базы данных в будущем, когда потребуется добавить к таблицам дополнительные столбцы.

Решение о том, к чему именно следует применить правило третьей нормальной формы, может оказаться непростым. Оценка должна основываться на том, какие дополнительные данные могут понадобиться со временем. Если вы абсолютно уверены в том, что ничего, кроме имени и адреса покупателя, не понадобится, то, наверное, без этой заключительной стадии нормализации можно будет обойтись.

С другой стороны, представьте, что вы создаете базу данных для такой крупной организации, как Почтовая служба США. Что вы будете делать, если город будет переименован? С такой таблицей, как табл. 9.6, вам придется проводить глобаль-

ный поиск и менять название города везде, где оно упоминается. Но если ваша база данных нормализована по правилам третьей нормальной формы, нужно будет изменить всего лишь одну запись в табл. 9.10 для того, чтобы это изменение отразилось на всей базе данных.

Поэтому я советую ответить себе на два вопроса, которые помогут принять решение, нужно ли применять нормализацию по правилам третьей нормальной формы к той или иной таблице.

- Существует ли вероятность того, что к таблице нужно будет добавить много новых столбцов?
- Может ли когда-нибудь для любого из полей этих таблиц потребоваться глобальное обновление?

Если оба ответа на эти вопросы положительные, значит вам все же следует провести заключительную стадию нормализации.

Когда не следует проводить нормализацию

Теперь, когда вы ознакомились со всеми тонкостями нормализации, я хочу рассказать о том, почему нужно отбросить все эти правила при работе с сайтами, имеющими высокий уровень обращений. Вам действительно *не следует* проводить полную нормализацию таблиц, используемых сайтом, если это приведет к излишней загруженности MySQL.

Нормализация требует распространения данных по нескольким таблицам, а это означает, что при каждом запросе будет осуществляться несколько вызовов MySQL. Если на сайте, пользующемся высокой популярностью, будут нормализованные таблицы и счет одновременно обслуживаемых пользователей пойдет на десятки, то скорость доступа к базе данных существенно снизится, потому что для их обслуживания потребуются сотни обращений к этой базе. Если серьезно, то я хочу пойти еще дальше и сказать, что вы должны провести максимально возможную *денормализацию* любых часто востребуемых данных.

Причина в том, что дублирование данных в таблицах позволяет существенно сократить количество необходимых дополнительных запросов, потому что основная масса востребованных данных доступна в каждой таблице. Это означает, что можно будет просто добавить к запросу еще один столбец, и это поле станет доступно для всех соответствующих результатов, хотя (разумеется) вам придется смириться со всеми упомянутыми ранее издержками, включая использование большого объема дискового пространства и обеспечение обновления каждой отдельной копии продублированных данных, когда одна из них требует модификации.

Конечно, многократные обновления можно компьютеризировать. Система MySQL предоставляет возможность использовать *триггеры*, которые осуществляют

автоматические изменения базы данных в соответствии с произведенными вами изменениями. (Триггеры в данной книге не рассматриваются.) Другой способ копирования в среде избыточных данных состоит в настройке PHP-программы на регулярный запуск и поддержание всех копий в синхронизированном состоянии. Программа считывает изменения с «ведущей» таблицы и обновляет все остальные. (Способы доступа к MySQL из PHP будут описаны в следующей главе.)

Но пока вы не приобретете опыт работы с MySQL, я рекомендую проводить полную нормализацию всех ваших таблиц (по крайней мере приводить к первой и второй нормальным формам), чтобы это вошло в привычку и принесло пользу в дальнейшем. Только после этого можно приступать к выявлению «заторов» в работе MySQL и присматриваться к денормализации.

Отношения

MySQL называют системой управления *реляционными* базами данных, потому что в ее таблицах содержатся не только данные, но и *отношения* между ними. Существует три категории отношений.

«Один к одному»

Отношение «один к одному» между двумя типами данных похоже на традиционные брачные отношения: каждый элемент данных соотносится только с одним элементом другого типа. Это на удивление редкий тип отношений. Например, автор может написать несколько книг, у книги может быть несколько авторов, и даже адрес может быть связан с несколькими покупателями. Возможно, наилучшим примером, встречавшимся в этой главе, может послужить отношение «один к одному» между названием штата и его двухбуквенной аббревиатурой.

Чтобы легче было объяснить, что это такое, предположим, что по какому-нибудь конкретному адресу может проживать только один покупатель. В таком случае отношение `Customers` — `Addresses` на рис. 9.1 будет отношением «один к одному»: только один покупатель живет по каждому адресу, и по каждому адресу может жить только один покупатель.

Обычно, когда у двух элементов имеется отношение «один к одному», их включают в качестве столбцов в одну и ту же таблицу. Для отнесения их к двум отдельным таблицам могут быть две причины:

- вы хотите быть готовыми к тому, что позже это отношение изменится и больше уже не будет иметь характер «один к одному»;
- в таблице слишком много столбцов, и вы полагаете, что производительность работы системы или возможности ее обслуживания улучшатся за счет ее разбиения.

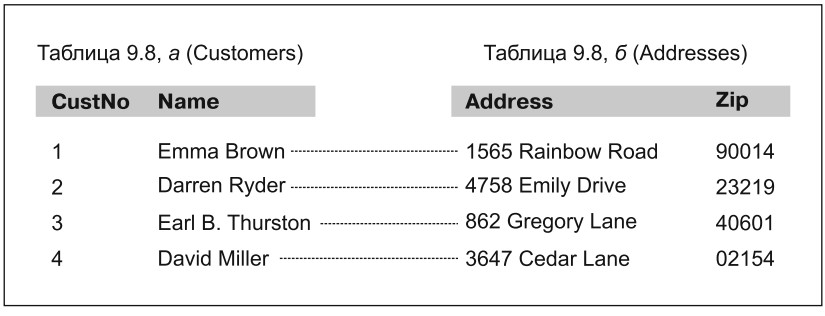

Таблица 9.8, *а* (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
3	Earl B. Thurston
4	David Miller

Таблица 9.8, *б* (Addresses)

Address	Zip
1565 Rainbow Road	90014
4758 Emily Drive	23219
862 Gregory Lane	40601
3647 Cedar Lane	02154

Рис. 9.1. Таблица покупателей, Customers (табл. 9.8), разбитая на две таблицы

Разумеется, когда дело дойдет до создания вашей собственной, настоящей базы данных, между покупателями и адресами нужно будет создать отношения «один ко многим» (*один* адрес, *много* покупателей).

«Один ко многим»

Отношения «один ко многим» (или «многие к одному») возникают в том случае, когда одна строка в одной таблице связана со многими строками в другой таблице. Вы уже поняли, что в табл. 9.8 возникли бы отношения «один ко многим», если бы несколько покупателей проживали по одному и тому же адресу. В таком случае ее нужно разбить.

Если посмотреть на табл. 9.8, *а*, показанную на рис. 9.1, можно увидеть, что у нее имеется отношение «один ко многим» с таблицей покупок, табл. 9.7, поскольку каждый покупатель представлен только одним конкретным человеком из табл. 9.8, *а*.

Но табл. 9.7, `Purchases`, может содержать (и содержит) более одной покупки, сделанной одним и тем же покупателем. Поэтому *один* покупатель имеет отношение *ко многим* покупкам.

На рис. 9.2 эти две таблицы показаны рядом друг с другом, а линии соединяют строки в каждой таблице и, начинаясь в одной строке левой таблицы, могут соединять с ней более одной строки правой таблицы. Схема отношения «один ко многим» также хорошо подходит и для описания отношения «многие к одному», в этом случае нужно левую и правую таблицы поменять местами и рассматривать их как отношение «один ко многим».

Чтобы отобразить в реляционной базе данных отношение «один ко многим», создайте таблицу для «многих» и таблицу для «одного». Таблица для «многих» должна содержать столбец с перечислением первичного ключа из таблицы для «одного». Таким образом, таблица `Purchases` будет содержать столбец с перечислением первичного ключа покупателя.

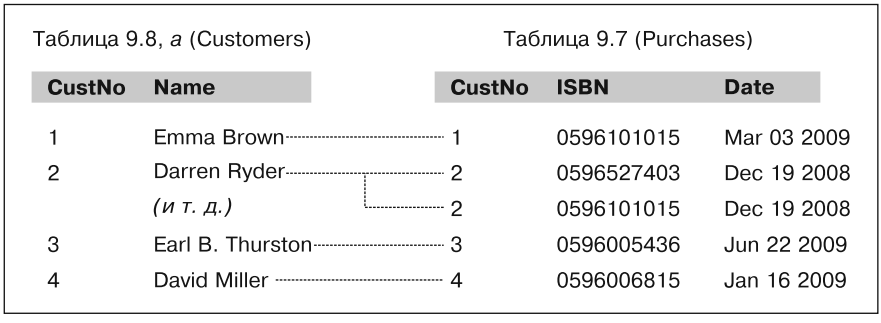

Рис. 9.2. Иллюстрация отношения между двумя таблицами

«Многие ко многим»

В отношении «многие ко многим» многие строки в одной таблице связаны с многими строками в другой таблице. Чтобы создать это отношение, нужно добавить третью таблицу, содержащую по столбцу из каждой из этих двух таблиц. В третьей таблице больше ничего не содержится, она предназначена только для связи других таблиц.

Именно такой промежуточной таблицей и является табл. 9.12. Она была извлечена из табл. 9.7, `Purchases` (Покупки), но в ней отсутствует информация о дате покупки. Теперь она содержит копию номера ISBN каждой проданной книги, а также номер покупателя.

Таблица 9.12. Промежуточная таблица

Customer (Покупатель)	ISBN
1	0596101015
2	0596527403
2	0596101015
3	0596005436
4	0596006815

С помощью этой промежуточной таблицы можно пройти по всем хранящимся в базе данным, пользуясь схемой их отношений. За отправную точку можно взять адрес и найти авторов любых книг, приобретенных покупателем, проживающим по этому адресу.

Предположим, к примеру, что нужно найти покупки, связанные с почтовым индексом 23219. Если поискать этот почтовый индекс в табл. 9.8, *б*, то можно обнаружить, что покупатель с номером 2 приобрел как минимум одну книгу, имеющуюся в базе данных. Теперь можно воспользоваться табл. 9.8, *а* и найти имя этого покупателя

или воспользоваться новой промежуточной табл. 9.12, для того чтобы найти приобретенную им книгу или книги.

По этой таблице можно определить, что были приобретены две книги, и, отследив их номера в табл. 9.4, найти названия и цены этих книг или обратиться к табл. 9.3 и увидеть в ней их авторов.

Если вам показалось, что все это, по сути, не что иное, как сочетание нескольких отношений «один ко многим», то так оно и есть. Чтобы проиллюстрировать это, на рис. 9.3 все три таблицы представлены вместе.

Проследите по любому почтовому индексу (zip-коду) в левой таблице связанные с ним идентификаторы покупателей. Далее можно проследить их связь с промежуточной таблицей, которая объединяет левую и правую таблицы путем связывания покупательских идентификаторов и номеров ISBN. Теперь остается только проследовать по ISBN к правой таблице, чтобы увидеть, к какой книге он относится.

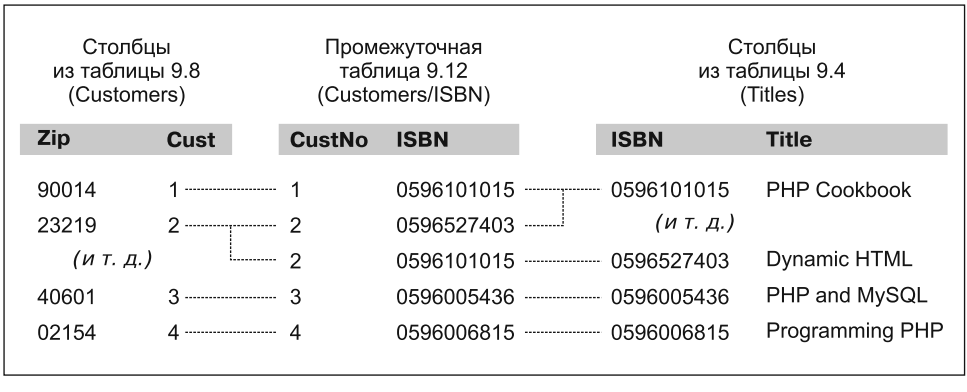

Рис. 9.3. Создание отношения «многие ко многим» с помощью третьей таблицы

Промежуточную таблицу можно использовать также для следования в обратном направлении — от названий книг до zip-кода. Из таблицы `Titles` можно взять ISBN, которым воспользоваться для поиска в промежуточной таблице идентификационных номеров покупателей этих книг, и, наконец, в таблице `Customers` идентификационные номера будут сопоставлены с zip-кодами мест проживания покупателей.

Базы данных и анонимность

Интересный аспект использования отношений заключается в том, что о каком-нибудь элементе, например покупателе, можно собрать массу сведений, не зная ничего о его личности. Обратите внимание на то, что в предыдущем примере мы прошли от покупательских zip-кодов к покупкам и вернулись назад, не определяя

имен покупателей. Базы данных могут использоваться для отслеживания сведений о людях, но они также могут использоваться и для защиты относящихся к ним конфиденциальных данных, при этом сохраняется возможность поиска полезной информации.

Транзакции

В некоторых приложениях жизненно необходимо, чтобы последовательность запросов шла в нужном порядке и при этом каждый отдельный запрос успешно завершался. Представим, например, что создается последовательность запросов для перевода средств с одного банковского счета на другой. Вам бы не хотелось, чтобы при этом происходило что-либо подобное:

- вы зачислили средства на второй счет, а когда попытались снять их с первого счета, при обновлении данных произошел сбой, и теперь эти средства числятся на обоих счетах;
- вы сняли средства с первого банковского счета, но при запросе на обновление с целью их зачисления на второй счет произошел сбой, и теперь эти средства бесследно исчезли.

Как видите, для этого типа транзакций важен не только порядок выполнения запросов, необходимо также, чтобы все части транзакции завершились успешно. Но как все это обеспечить? Ведь после осуществления запроса аннулировать его уже невозможно. Необходимо ли отслеживать все части транзакции, а затем проводить полный откат, если одна из ее частей даст сбой? Ничего этого делать не нужно, поскольку MySQL поставляется с мощным средством обработки транзакций, которое защищает именно от таких непредвиденных обстоятельств.

Кроме того, транзакции предоставляют одновременный доступ к базе данных множеству пользователей или программ за счет обеспечения очередности проведения всех транзакций, и каждый пользователь или программа соблюдают очередность, не наступая друг другу на пятки, — MySQL со всем этим прекрасно справляется.

Ядра (механизмы хранения) транзакций

Чтобы использовать имеющееся в MySQL средство обработки транзакций, нужно задействовать MySQL-ядро InnoDB (используемое по умолчанию, начиная с версии 5.5). Если вы не уверены в номере используемой версии MySQL, на которой будет запускаться ваш код, нужно не выстраивать предположения, что InnoDB является механизмом по умолчанию, а использовать его принудительно при создании таблицы, как показано далее.

Создадим таблицу банковских счетов, введя команды, приведенные в примере 9.1. (Напомню, что для этого нужно получить доступ к командной строке MySQL и воспользоваться подходящей для этой таблицы базой данных.)

Пример 9.1. Создание таблицы, готовой к обработке транзакций

```
CREATE TABLE accounts (
  number INT, balance FLOAT, PRIMARY KEY(number)
  ) ENGINE InnoDB;
DESCRIBE accounts;
```

Команда, которая находится в последней строке этого примера, отобразит содержимое новой таблицы, позволяя убедиться в ее успешном создании. Будет выведена следующая информация:

```
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| number  | int(11) | NO   | PRI | 0       |       |
| balance | float   | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Теперь создадим в этой таблице две строки, которые можно будет задействовать в транзакциях. Для этого введем команды, показанные в примере 9.2.

Пример 9.2. Заполнение таблицы accounts

```
INSERT INTO accounts(number, balance) VALUES(12345, 1025.50);
INSERT INTO accounts(number, balance) VALUES(67890, 140.00);
SELECT * FROM accounts;
```

Команда в третьей строке отобразит содержимое таблицы, подтверждая успешное создание строк. Будет выведена следующая информация:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 |  1025.5 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

После создания и предварительного заполнения этой таблицы можно приступить к использованию транзакций.

Команда BEGIN

Транзакции в MySQL начинаются либо с команды `BEGIN`, либо с команды `START TRANSACTION`. Чтобы отправить транзакцию системе MySQL, введите команды, показанные в примере 9.3.

Пример 9.3. Транзакция MySQL

```
BEGIN;
UPDATE accounts SET balance=balance+25.11 WHERE number=12345;
COMMIT;
SELECT * FROM accounts;
```

Результаты этой транзакции выводятся командой, содержащейся в последней строке, и должны иметь следующий вид:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 | 1050.61 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

Как видите, баланс счета 12345 увеличился на 25,11 и теперь составляет 1050,61. В примере 9.3 можно было также заметить команду `COMMIT`, которая рассматривается в следующем разделе.

Команда COMMIT

Когда вы убедитесь в том, что ряд запросов, входящих в транзакцию, успешно выполнен, введите команду `COMMIT`, чтобы передать все изменения базе данных. До тех пор пока не будет получена команда `COMMIT`, все внесенные изменения рассматриваются MySQL как временные. Эта особенность позволяет отменить транзакцию, отправляя вместо команды передачи `COMMIT` команду отката `ROLLBACK`.

Команда ROLLBACK

Используя команду `ROLLBACK`, можно заставить MySQL забыть обо всех запросах, выданных с начала транзакции, и отменить транзакцию. Можете проверить эту команду в действии путем ввода транзакции по переводу средств, показанной в примере 9.4.

Пример 9.4. Транзакция по переводу средств

```
BEGIN;
UPDATE accounts SET balance=balance-250 WHERE number=12345;
UPDATE accounts SET balance=balance+250 WHERE number=67890;
SELECT * FROM accounts;
```

Как только будут введены эти строки, вы увидите следующий результат:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 |  800.61 |
|  67890 |     390 |
+--------+---------+
2 rows in set (0.00 sec)
```

Теперь у первого банковского счета значение на 250 единиц меньше, чем раньше, а значение второго увеличилось на 250 единиц — вы осуществили между ними перевод на 250 единиц. А теперь предположим, что что-то пошло не так и эту транзакцию нужно отменить. Для этого необходимо лишь ввести команду, показанную в примере 9.5.

Пример 9.5. Отмена транзакции с помощью команды ROLLBACK

```
ROLLBACK;
SELECT * FROM accounts;
```

Теперь вы должны увидеть следующую выходную информацию, показывающую восстановление прежнего баланса на обоих счетах, благодаря тому, что транзакция была отменена командой `ROLLBACK`:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 | 1050.61 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

Команда EXPLAIN

Система MySQL поставляется с мощным инструментарием, который позволяет исследовать, как она интерпретировала выданные ей запросы. Используя команду `EXPLAIN`, можно получить отображение состояния любого запроса, чтобы понять, можно ли его выдать более удобным или эффективным способом. Применение этой команды с созданной ранее таблицей `accounts` показано в примере 9.6.

Пример 9.6. Использование команды EXPLAIN

```
EXPLAIN SELECT * FROM accounts WHERE number='12345';
```

Результаты выполнения команды `EXPLAIN` будут выглядеть следующим образом:

```
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
|id|select_type|table   |type |possible_keys|key    |key_len|ref  |rows|Extra|
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
| 1|SIMPLE     |accounts|const|PRIMARY      |PRIMARY|4      |const|   1|     |
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
1 row in set (0.00 sec)
```

Здесь MySQL предоставляет вам следующую информацию.

- `select_type`. Тип выборки простой — `SIMPLE`. При объединении таблиц будет показан объединенный (join) тип.
- `table`. Текущей запрашиваемой таблицей была `accounts`.
- `type`. Тип запроса — `const`. Если идти от наименее эффективного типа к наиболее эффективному, то возможные значения выстраиваются в следующий ряд: `ALL`, `index`, `range`, `ref`, `eq_ref`, `const`, `system` и `NULL`.

- `possible_keys`. Возможно, это первичный ключ, `PRIMARY`, а это значит, что доступ должен быть быстрым.
- `key`. В данном случае используется ключ `PRIMARY`, что является хорошим показателем.
- `key_len`. Длина ключа равна `4`. Это количество байтов индекса, которое будет использовано MySQL.
- `ref`. Столбец `ref` отображает, какие столбцы или константы используются с ключом. В данном случае применяется константный ключ.
- `rows`. Количество строк, которые должны быть просмотрены этим запросом, равно `1`, что также является хорошим показателем.

Когда появится запрос, который подозревается в лишней трате времени на свое выполнение, попробуйте воспользоваться командой `EXPLAIN`, чтобы посмотреть, как его можно оптимизировать. Вы сможете обнаружить, какие ключи (если таковые имеются) были задействованы, какова их длина и т. д., и тогда можно будет соответствующим образом подкорректировать запрос или конструкцию таблицы (или таблиц).

После того как эксперименты с временной таблицей accounts будут завершены, может появиться желание удалить эту таблицу с помощью следующей команды:

```
DROP TABLE accounts;
```

Резервное копирование и восстановление данных

Независимо от того, какого рода данные хранятся в вашей базе, они все равно должны представлять для вас определенную ценность, даже если она измеряется временем, необходимым для их повторного ввода в случае повреждения жесткого диска. Поэтому для защиты вложенного вами труда важно сохранять резервные копии. Может также возникнуть потребность в перемещении вашей базы данных на новый сервер, и наилучшим способом является предварительное снятие с нее резервной копии. Важно также время от времени проверять резервные копии, для того чтобы убедиться в их целостности и работоспособности.

Создание резервных копий и восстановление данных MySQL существенно облегчаются при использовании команды `mysqldump`.

Команда mysqldump

Команда `mysqldump` позволяет выгрузить базу данных или коллекцию баз данных в один или несколько файлов, содержащих все инструкции, необходимые для воссоздания всех ваших таблиц и повторного заполнения их данными. Эта команда

также может создавать файлы в формате с разделением значений запятыми — CSV (Comma-Separated Values) и в других текстовых форматах, использующих разделители, или даже в XML-формате. Главный недостаток команды заключается в том, что в процессе резервного копирования таблицы нужно обеспечить, чтобы никто не ввел в нее запись. Эта задача решается разными способами, но самый простой состоит в остановке MySQL-сервера перед запуском `mysqldump` и его повторном запуске после окончания ее работы.

Можно также перед запуском команды `mysqldump` заблокировать все копируемые таблицы. Для блокировки чтения таблиц (поскольку нам нужно считать данные) в командную строку MySQL нужно ввести следующую команду:

```
LOCK TABLES имя_таблицы1 READ, имя_таблицы2 READ ...
```

А для снятия блокировки нужно ввести такую команду:

```
UNLOCK TABLES;
```

По умолчанию вся выходная информация выводится командой `mysqldump` на стандартное устройство, но ее можно перенаправить в файл, воспользовавшись символом >.

Стандартный формат `mysqldump` имеет следующий вид:

```
mysqldump -u пользователь -pпароль база_данных
```

Но перед тем, как выгружать содержимое базы данных, важно убедиться в том, что путь к программе `mysqldump` может быть найден по умолчанию, или же указать ее размещение в самой команде. В табл. 9.13 показаны наиболее вероятные места нахождения этой программы для различных установок и операционных систем, рассмотренных в главе 2. Если у вас какой-нибудь другой вариант установки, местонахождение программы может быть несколько иным.

Таблица 9.13. Наиболее вероятные места нахождения программы mysqldump для различных установок

Операционная система и программа	Наиболее вероятная папка местонахождения
Windows AMPPS	C:\Program Files (x86)\Ampps\mysql\bin
macOS AMPPS	/Applications/ampps/mysql/bin
Linux AMPPS	/Applications/ampps/mysql/bin

Для вывода на экран содержимого базы данных `publications`, созданной в главе 8, сначала выйдите из MySQL, а затем введите команду, показанную в примере 9.7 (указав при необходимости полный путь к `mysqldump`).

Пример 9.7. Вывод базы данных publications на экран

```
mysqldump -u пользователь -pпароль publications
```

Вместо слов *пользователь* и *пароль* подставьте имя пользователя и пароль, которые используются в вашей установке MySQL. Если пароль для пользователя не установлен, эту часть команды можно опустить, но часть команды `-u` *пользователь* является обязательной, если только у вас не установлен привилегированный доступ (root) без пароля и вы не работаете в этом режиме (что делать не рекомендуется). Результат ввода этой команды будет похож на тот, что изображен на рис. 9.4.

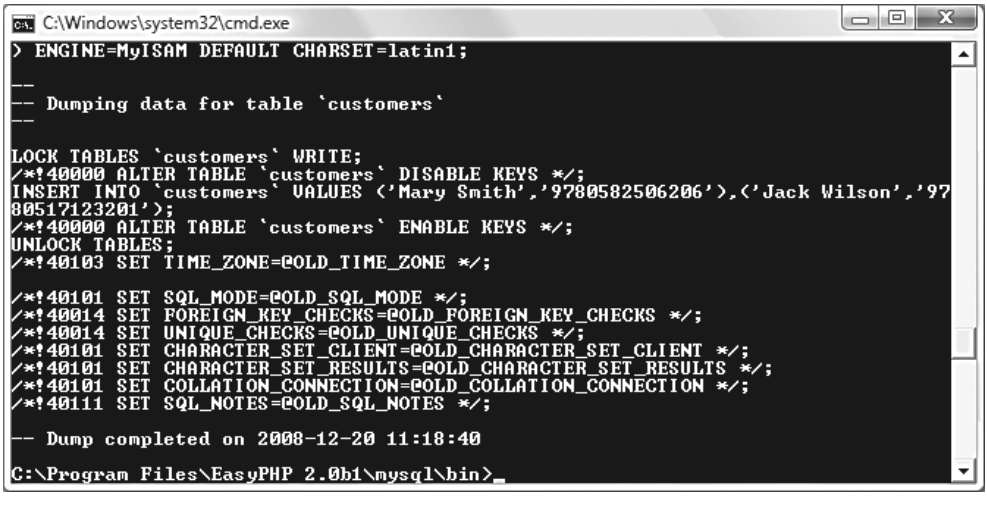

Рис. 9.4. Выгрузка базы данных publications на экран

Создание файла резервной копии

Запустив команду `mysqldump` и убедившись в том, что она выводит на экран нужные данные, можно перенаправить данные резервной копии непосредственно в файл, используя символ `>`.

Если предположить, что вам захотелось назвать файл резервной копии `publications.sql`, нужно ввести команду, показанную в примере 9.8 (не забудьте подставить вместо слов *пользователь* и *пароль* настоящее имя пользователя и пароль).

Пример 9.8. Выгрузка базы данных publications в файл

```
mysqldump -u пользователь -pпароль publications > publications.sql
```

Команда в примере 9.8 сохраняет файл резервной копии в текущем каталоге. Если нужно сохранить его в каком-нибудь другом месте, то перед именем файла следует указать соответствующий путь. Кроме того, необходимо убедиться в том, что каталог, куда будет сохраняться файл резервной копии, имеет соответствующие установки доступности, позволяющие записывать в него этот файл.

При выводе файла резервной копии на экран или загрузке его в текстовый редактор вы увидите, что он состоит из последовательности SQL-команд:

```
DROP TABLE IF EXISTS 'classics';
CREATE TABLE 'classics' (
  'author' varchar(128) default NULL,
  'title' varchar(128) default NULL,
  'category' varchar(16) default NULL,
  'year' smallint(6) default NULL,
  'isbn' char(13) NOT NULL default '',
  PRIMARY KEY ('isbn'),
  KEY 'author' ('author' (20)),
  KEY 'title' ('title' (20)),
  KEY 'category' ('category' (4)),
  KEY 'year' ('year'),
  FULLTEXT KEY 'author_2' ('author','title')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Это весьма продуманный код, который может быть использован для восстановления базы данных из резервной копии, даже если она уже существует, поскольку сначала он удалит все таблицы, которые должны быть воссозданы, избавляясь таким образом от потенциальных ошибок MySQL.

Создание резервной копии отдельной таблицы

Чтобы создать резервную копию отдельной таблицы базы данных (такой как таблица `classics` базы данных `publications`), сначала нужно из командной строки MySQL заблокировать таблицу, набрав следующую команду:

```
LOCK TABLES publications.classics READ;
```

Это обеспечит работу MySQL в режиме чтения, но сделает невозможной запись. Затем, не закрывая командную строку MySQL, используйте другое окно терминала, чтобы ввести из командной строки операционной системы следующую команду:

```
mysqldump -u пользователь -pпароль publications classics > classics.sql
```

Теперь можно снять блокировку таблицы, для чего в первом окне терминала в командной строке MySQL нужно ввести следующую команду, которая разблокирует все таблицы, заблокированные в текущем сеансе:

```
UNLOCK TABLES;
```

Создание резервной копии всех таблиц

Если понадобится создать резервную копию сразу всех ваших баз данных MySQL (включая и такие системные базы данных, как `mysql`), можно воспользоваться командой, показанной в примере 9.9, которая позволит восстановить всю установку базы данных MySQL, но при этом следует не забыть про блокировку там, где она потребуется.

Пример 9.9. Выгрузка всех баз данных MySQL в файл

```
mysqldump -u пользователь -pпароль --all-databases > all_databases.sql
```

Разумеется, в файлах резервных копий баз данных MySQL содержится очень много строк SQL-кода. Я советую потратить несколько минут на изучение ряда этих строк для ознакомления с типами команд, которые встречаются в файлах резервных копий, и с порядком их работы.

Восстановление данных из файла резервной копии

Чтобы восстановить данные из файла, нужно вызвать исполняемую программу `mysql` и передать ей файл, из которого восстанавливаются данные, для чего следует воспользоваться символом <. Для восстановления всей базы данных, выгруженной с помощью ключа `--all-databases`, используется команда, показанная в примере 9.10.

Пример 9.10. Восстановление полного набора баз данных

```
mysql -u пользователь -pпароль < all_databases.sql
```

Для восстановления одной базы данных применяется ключ `-D`, за которым следует имя базы данных. В примере 9.11 показано, как восстановить базу данных `publications` из резервной копии, созданной кодом, который показан в примере 9.8.

Пример 9.11. Восстановление базы данных publications

```
mysql -u пользователь -pпароль -D publications < publications.sql
```

Для восстановления отдельной таблицы базы данных используется команда, показанная в примере 9.12, где в базе данных `publications` восстанавливается только таблица `classics`.

Пример 9.12. Восстановление таблицы classics в базе данных publications

```
mysql -u пользователь -pпароль -D publications < classics.sql
```

Выгрузка данных в файлы формата CSV

Как уже отмечалось, программа `mysqldump` обладает завидной гибкостью и поддерживает различные типы выходных данных, в том числе формат CSV, которым, кроме всего прочего, можно воспользоваться для импортирования данных в электронную таблицу. В примере 9.13 показано, как можно выгрузить данные из таблиц `classics` и `customers` базы данных `publications` в файлы `classics.txt` и `customers.txt`, находящиеся в папке `c:/temp`. Если работа идет в операционной системе macOS или Linux, следует изменить путь назначения на существующую папку.

Пример 9.13. Выгрузка данных в файлы формата CSV

```
mysqldump -u пользователь -pпароль --no-create-info --tab=c:/temp
  --fields-terminated-by=',' publications
```

Команда слишком длинная, и в этом примере она занимает несколько строк, но вводить ее нужно в одной строке. В результате работы команды будет выведен следующий текст:

```
Mark Twain (Samuel Langhorne Clemens)','The Adventures of Tom Sawyer',
  'Classic Fiction','1876','9781598184891
Jane Austen','Pride and Prejudice','Classic Fiction','1811','9780582506206
Charles Darwin','The Origin of Species','Non-Fiction','1856','9780517123201
Charles Dickens','The Old Curiosity Shop','Classic Fiction','1841','9780099533474
William Shakespeare','Romeo and Juliet','Play','1594','9780192814968

Mary Smith','9780582506206
Jack Wilson','9780517123201
```

Планирование резервного копирования

Золотое правило резервного копирования гласит, что его следует проводить с той периодичностью, которая имеет практический смысл. Чем ценнее данные, тем чаще следует создавать их резервные копии и тем больше резервных копий нужно делать. Если ваша база данных обновляется хотя бы раз в сутки, то резервное копирование необходимо проводить ежедневно. Если же она не подвергается частым обновлениям, то резервные копии можно создавать значительно реже.

Нужно также подумать о создании нескольких резервных копий и о хранении их в разных местах. Если у вас используются несколько серверов, то можно просто растиражировать резервные копии по этим серверам. Можно также прислушаться к хорошему совету и создавать физические резервные копии съемных жестких дисков, миниатюрных носителей, компакт-дисков или DVD и т. д. и хранить их в разных местах, предпочтительно в чем-то вроде сейфов.

Важно также периодически тестировать восстановление базы данных, чтобы убедиться, что резервные копии выполнены правильно. Освоение процесса восстановления базы данных понадобится и по той причине, что вам, возможно, придется им заниматься в стрессовой ситуации и в спешке, например после сбоя питания, закрывшего веб-сайт. Базу данных можно восстановить на частном сервере и запустить несколько SQL-команд, чтобы убедиться в том, что данные находятся в должном состоянии.

После изучения материалов этой главы вы должны стать специалистом по работе как с PHP, так и с MySQL. В следующей главе будет показано, как можно объединить эти две технологии.

Вопросы

1. Что означает слово *отношение* (relationship) применительно к реляционным базам данных?
2. Какое понятие применяется к процессу удаления повторяющихся данных и оптимизации таблиц?
3. Как формулируются три правила первой нормальной формы?
4. Как привести таблицу в соответствие с правилом второй нормальной формы?
5. Что нужно поместить в столбец, чтобы связать две таблицы, содержащие элементы, имеющие отношение «один ко многим»?
6. Как создать базу данных с отношением «многие ко многим»?
7. Какие команды инициируют и завершают транзакцию MySQL?
8. Какие возможности предоставляет MySQL для изучения подробностей работы запроса?
9. Какую команду нужно использовать для создания резервной копии базы данных `publications` в файле `publications.sql`?

Ответы на эти вопросы можно найти в приложении A, в разделе «Ответы на вопросы главы 9».

10 Доступ к MySQL с использованием PHP

При полноценном изучении предыдущих глав вы должны были приобрести навыки работы как с MySQL, так и с PHP. Из этой главы вы узнаете, как объединить эти два компонента путем использования встроенных в PHP функций доступа к MySQL.

Запросы к базе данных MySQL с помощью PHP

Смысл использования PHP в качестве интерфейса к MySQL заключается в форматировании результатов SQL-запросов и придании им внешнего вида, предназначенного для вывода на веб-страницу. Обладая возможностью входа в установленную систему MySQL с помощью своего имени пользователя и пароля, вы можете сделать то же самое и из PHP. Но вместо использования командной строки MySQL для ввода команд и просмотра выходной информации нужно будет создать строки запроса, а затем передать их MySQL. Ответ MySQL поступит в виде структуры данных, которую PHP сможет распознать, а не в виде того отформатированного экранного вывода, который вы наблюдали в процессе работы с командной строкой. Затем с помощью команд PHP можно будет извлекать данные и приводить их к формату веб-страницы.

Процесс

Процесс использования MySQL с помощью PHP заключается в следующем.

1. Подключение к MySQL и выбор базы данных, которая будет использоваться.
2. Подготовка строки запроса.
3. Выполнение запроса.
4. Извлечение результатов и вывод их на веб-страницу.

5. Повторение шагов с 2-го по 4-й до тех пор, пока не будут извлечены все необходимые данные.
6. Отключение от MySQL.

Далее процесс будет рассмотрен поэтапно, но сначала важно настроить все элементы входа в систему на безопасную работу, для того чтобы шпионы, заинтересовавшиеся вашей системой, натыкались на заслон при попытке получения доступа к вашей базе данных.

Создание файла регистрации

Большинство сайтов, разработанных на PHP, содержат множество программных файлов, которым понадобится доступ к MySQL, и им нужны будут сведения, касающиеся входа в систему и пароля. Поэтому имеет смысл создать отдельный файл для их хранения, а затем включать его туда, где он необходим. Такой файл, который я назвал `login.php`, показан в примере 10.1.

Пример 10.1. Файл login.php

```
<?php // login.php
  $hm = 'localhost';
  $db = 'publications';
  $un = 'имя_пользователя';
  $pw = 'пароль';
?>
```

Наберите текст этого примера, заменяя значения *имя_пользователя* и *пароль* теми, которыми вы пользуетесь для входа в свою базу данных MySQL, и сохраните текст в файле, поместив его в разработочный каталог, созданный согласно рекомендациям, которые были даны в главе 2. Вскоре этот файл нам пригодится.

Имя хоста `localhost` будет работать до тех пор, пока вы используете базу данных MySQL в своей локальной системе, точно так же будет работать и база данных `publications`, если вы вводили в компьютер код всех встречавшихся ранее примеров.

Для файла `login.php`, показанного в примере 10.1, особую роль играют охватывающие теги `<?php` и `?>`, поскольку они дают понять, что все строки, находящиеся между ними, должны интерпретироваться *только* как код PHP. Если их не поставить, то при вызове файла непосредственно с вашего сайта он будет отображен в виде текста, раскрывая все ваши секреты. А когда теги на месте, на сайте будет видна пустая страница. Этот файл будет без каких-либо проблем включаться в другие ваши PHP-файлы.

Переменная `$hn` сообщит PHP, какой компьютер следует использовать при подключении к базе данных. Ее присутствие обусловлено тем, что вы можете получить доступ к любой базе данных MySQL на любом компьютере, подключенном к той

машине, на которой вы установили PHP, и она может потенциально включать в себя любой хост на просторах Всемирной паутины. Но примеры, приводимые в данной главе, будут работать только на локальном сервере. Поэтому здесь не будет указываться домен вроде `mysql.myserver.com`, а может просто использоваться слово `localhost` (или IP-адрес `127.0.0.1`).

В роли рабочей базы данных, `$db`, будет выступать база данных `publications`, которую мы уже создали, изучая главу 8 (или одна из тех баз данных, которую вам предоставил администратор вашего сервера, в таком случае нужно будет также внести соответствующие изменения в файл `login.php`).

Другим преимуществом хранения всех сведений, необходимых для входа в систему, в одном месте станет возможность изменения пароля с нужной вам периодичностью, для чего придется обновлять только один файл, независимо от количества PHP-файлов, получающих доступ к MySQL.

Подключение к базе данных MySQL

После сохранения файла `login.php` можно будет с помощью инструкции `require_once` включать его в любые PHP-файлы, которым нужен доступ к базе данных. Выбор пал именно на эту инструкцию, а не на инструкцию `include`, поскольку, если файл не будет найден, он сгенерирует фатальную ошибку. И уж поверьте мне, если не будет найден файл, содержащий сведения для подключения к вашей базе данных, это действительно будет фатальной ошибкой.

Использование `require_once`, а не `require` означает, что файл будет считан только в том случае, если он не был включен до этого в какой-нибудь другой файл, что исключит совершенно бесполезные повторные обращения к диску. Код, используемый для подключения, показан в примере 10.2.

Пример 10.2. Подключение к серверу MySQL с помощью mysqli

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");
?>
```

В данном примере при вызове нового экземпляра метода `mysqli` с передачей ему всех значений, извлеченных из файла `login.php`, создается новый объект по имени `$conn`. Проверка на возникновение ошибки осуществляется путем ссылки на свойство `$conn->connect_error`.

Оператор `->` показывает, что элемент справа от него является свойством или методом объекта, обозначенного слева от него. В данном случае, если у `connect_error`

имеется значение, значит произошла ошибка, из-за чего вызывается функция `die` для прекращения выполнения программы.

Объект `$conn` используется в следующих примерах доступа к базе данных MySQL.

Функция die хорошо подходит для разработки кода PHP, но на рабочем сервере, вам, конечно же, захочется использовать более вразумительные сообщения об ошибках. В этом случае нужно будет не выходить из программы PHP в аварийном режиме, а составить сообщение, которое будет отображено при нормальном выходе из программы, например:

```
function mysql_fatal_error()
{
  echo <<< _END

К сожалению, завершить запрашиваемую задачу не представилось возможным.
Было получено следующее сообщение об ошибке:
  <p>Fatal Error</p>
Пожалуйста, нажмите кнопку возврата вашего браузера и повторите
попытку. Если проблемы не прекратятся, пожалуйста, <a
href="mailto:admin@server.com">сообщите о них нашему администратору
</a>. Спасибо.
_END;
}
```

Также никогда не нужно пытаться вывести содержимое любого сообщения об ошибке, полученного от MySQL. Вместо того чтобы помочь своим пользователям, вы можете предоставить хакерам такую конфиденциальную информацию, как данные о входе в систему. Лучше просто снабдите пользователя информацией о способе преодоления трудностей на основе того отчета, который содержится в сообщении об ошибке.

Создание и выполнение запроса

Оправка запроса к MySQL из PHP сводится к простому включению соответствующего кода SQL в вызов метода `query`, принадлежащего объекту подключения. Как это делается, показано в примере 10.3.

Пример 10.3. Отправка запроса к базе данных с помощью mysqli

```
<?php
  $query = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die ("Fatal Error");
?>
```

Как видите, MySQL-запрос выглядит практически так же, как и непосредственно набираемая вручную инструкция в командной строке, за исключением того, что здесь отсутствует замыкающая точка с запятой, поскольку при обращении к MySQL из PHP она не нужна.

Сначала переменной `$query` присваивается значение, содержащее код предстоящего запроса, а затем она передается методу `query` объекта `$conn`, который возвращает результат, помещаемый в объект `$result`. Если в объекте `$result` содержится значение `FALSE`, то возникла проблема, подробности которой будут содержаться в свойстве `error` объекта подключения, а вызываемая функция `die` покажет их на экране.

Теперь все данные, возвращаемые MySQL, хранятся в легко поддающемся опросу формате в объекте `$result`.

Извлечение результата

После возвращения объекта `$result` его можно использовать для поэлементного извлечения нужных вам данных с помощью имеющегося у этого объекта метода `fetch_assoc`. В примере 10.4 предыдущие примеры объединены и расширены в программу, которую для получения этих результатов можно запустить самостоятельно (рис. 10.1). Наберите этот сценарий и сохраните его под именем `query-mysqli.php` или же загрузите его с сопровождающего книгу сайта http://lpmj.net/.

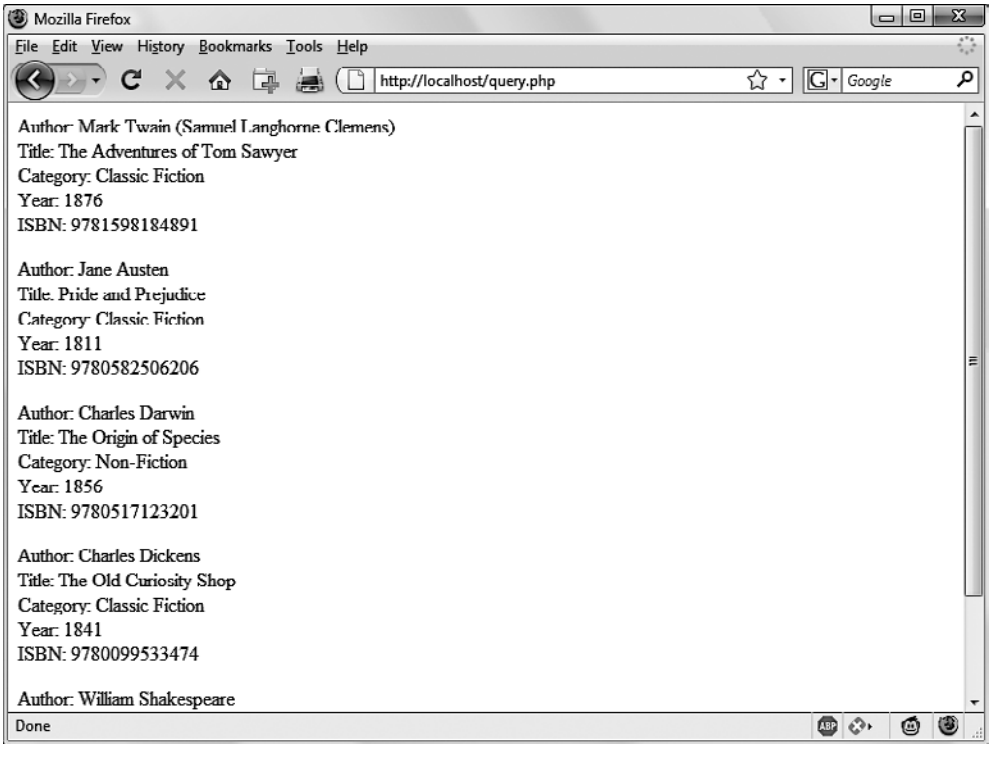

Рис. 10.1. Данные, выводимые программой query.php, представленной в примере 10.4

Пример 10.4. Поэлементное извлечение результатов

```
<?php // query-mysqli.php
  require_once 'login.php';
  $connection = new mysqli ($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  $query = "SELECT * FROM classics";
  $result = $connection->query($query);

  if (!$result) die ("Fatal Error");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $result->data_seek($j);
    echo 'Author: '   .htmlspecialchars($result->fetch_assoc()['author']) .'<br>';
    $result->data_seek($j);
    echo 'Title: '    .htmlspecialchars($result->fetch_assoc()['title']) .'<br>';
    $result->data_seek($j);
    echo 'Category: ' .htmlspecialchars($result->fetch_assoc()['category']).'<br>';
    $result->data_seek($j);
    echo 'Year: '     .htmlspecialchars($result->fetch_assoc()['year']) .'<br>';
    $result->data_seek($j);
    echo 'ISBN: '     .htmlspecialchars($result->fetch_assoc()['isbn']) .'<br><br>';
  }

  $result->close();
  $connection->close();
?>
```

Для извлечения значения, хранящегося в каждом поле, при каждом прохождении цикла вызывается метод `fetch_assoc`, а для вывода результата на экран используются инструкции `echo`.

При выводе в браузер данных, источником которых был (или мог быть) пользовательский ввод, всегда есть риск наличия в них вставок из вредоносного кода HTML, даже если вы уверены, что эти данные прошли процесс обезвреживания. Такой код может потенциально использоваться для осуществления атак по принципу межсайтового выполнения сценариев (XSS-атак). Простым способом устранения такой возможности является вставка всех выводимых на экран данных такого характера в вызов функции `htmlspecialchars`, заменяющей все опасные символы безопасными HTML-элементами. Именно этот прием был применен в предыдущем примере и будет использоваться во многих последующих примерах книги.

Трудно не согласиться с тем, что весь этот многократный поиск и т. д. выглядит слишком громоздко и что должен быть более эффективный способ достижения того же результата. И такой способ построчного извлечения данных действительно есть.

В главе 9 шла речь о первой, второй и третьей нормальных формах, а теперь можно заметить, что таблица classics не удовлетворяет правилам этих форм, потому что сведения как об авторах, так и о книгах включены в одну и ту же таблицу. Причина состоит в том, что эта таблица была создана еще до того, как мы приступили к изучению нормализации. Но ее повторное использование для иллюстрации доступа к MySQL из PHP избавляет нас от необходимости ввода нового набора тестовых данных, поэтому в данном случае мы продолжим работу с этой таблицей.

Извлечение строки

Для построчного извлечения данных цикл `for` из примера 10.4 следует заменить циклом, выделенным в примере 10.5 полужирным шрифтом, и вы сможете убедиться в том, что будет получен точно такой же результат, как и на рис. 10.1. Этот исправленный файл можно сохранить под именем `fetchrow.php`.

Пример 10.5. Построчное извлечение результатов

```
<?php // fetchrow.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die("Fatal Error");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_array(MYSQLI_ASSOC);

    echo 'Author: '   .htmlspecialchars($row['author'])     . '<br>';
    echo 'Title: '    .htmlspecialchars($row['title'])      . '<br>';
    echo 'Category: ' .htmlspecialchars($row['category'])   . '<br>';
    echo 'Year: '     .htmlspecialchars($row['year'])       . '<br>';
    echo 'ISBN: '     .htmlspecialchars($row['isbn'])       . '<br><br>';
  }

  $result->close();
  $conn->close();
?>
```

В этом измененном коде к объекту `$result` выполняется всего лишь одна пятая обращений (по сравнению с предыдущим примером) и при каждом проходе цикла производится только один поиск внутри объекта, поскольку посредством использования метода `fetch_array` каждая строка извлекается целиком. То есть отдельная строка данных возвращается в виде массива, значения которого затем присваиваются массиву `$row`.

В соответствии с переданным ему значением метод `fetch_array` может возвращать три типа массивов.

- `MYSQLI_NUM` — числовой массив. Каждый столбец появляется в массиве в том порядке, который был определен при созданиии (или изменении) таблицы. В нашем случае нулевой элемент массива содержит столбец `Author`, элемент 1 содержит `Title` и т. д.
- `MYSQLI_ASSOC` — ассоциативный массив. Каждый ключ является именем столбца. Поскольку на элементы данных дается ссылка по имени столбца (а не по номеру индекса), этот вариант нужно использовать в своем коде везде, где только можно, чтобы облегчить отладку и помочь другим программистам в сопровождении вашего кода.
- `MYSQLI_BOTH` — ассоциативный и числовой массив.

Ассоциативные массивы обычно полезнее числовых, поскольку к каждому столбцу можно обращаться по имени, например `$row['author']`, не утруждая себя воспоминаниями, каким идет нужный столбец по счету. Поэтому в данном сценарии используется ассоциативный массив, что заставило нас передать методу аргумент `MYSQLI_ASSOC`.

Отключение

Со временем, после того как выполнение сценария завершится, PHP должен возвратить память, выделенную объектам, следовательно, при использовании небольших сценариев о самостоятельном высвобождении памяти вам обычно беспокоиться не следует. Но если результирующим объектам выделен слишком большой объем памяти или были извлечены большие объемы данных, было бы неплохо высвободить используемую память во избежание возможных проблем при работе вашего сценария.

Особую важность это приобретает на страницах с более высоким уровнем трафика, поскольку объем памяти, потребляемый в ходе сессии, может быстро возрасти.

Поэтому обратите внимание на то, что в предыдущих сценариях, как только миновала надобность в объекте, вызывались методы `close` объектов `$result` и `$conn`:

```
$result->close();
$conn->close();
```

В идеале каждый результирующий объект нужно закрывать сразу же, как только будет завершено его использование, а затем, когда ваш сценарий больше не будет обращаться к MySQL, нужно закрыть объект подключения. Чтобы оптимизировать работу MySQL и развеять сомнения в способности PHP к возвращению неиспользуемой памяти на тот момент, когда в ней возникнет острая необходимость, нужно взять за правило возвращать ресурсы как можно быстрее.

Практический пример

Теперь настало время создать наш первый пример использования PHP для вставки данных в таблицу MySQL и удаления их оттуда. Я рекомендую набрать пример 10.6 и сохранить его в вашем разработочном каталоге в файле под именем `sqltest.php`. В результате работы кода из этого примера экран приобретает вид, показанный на рис. 10.2.

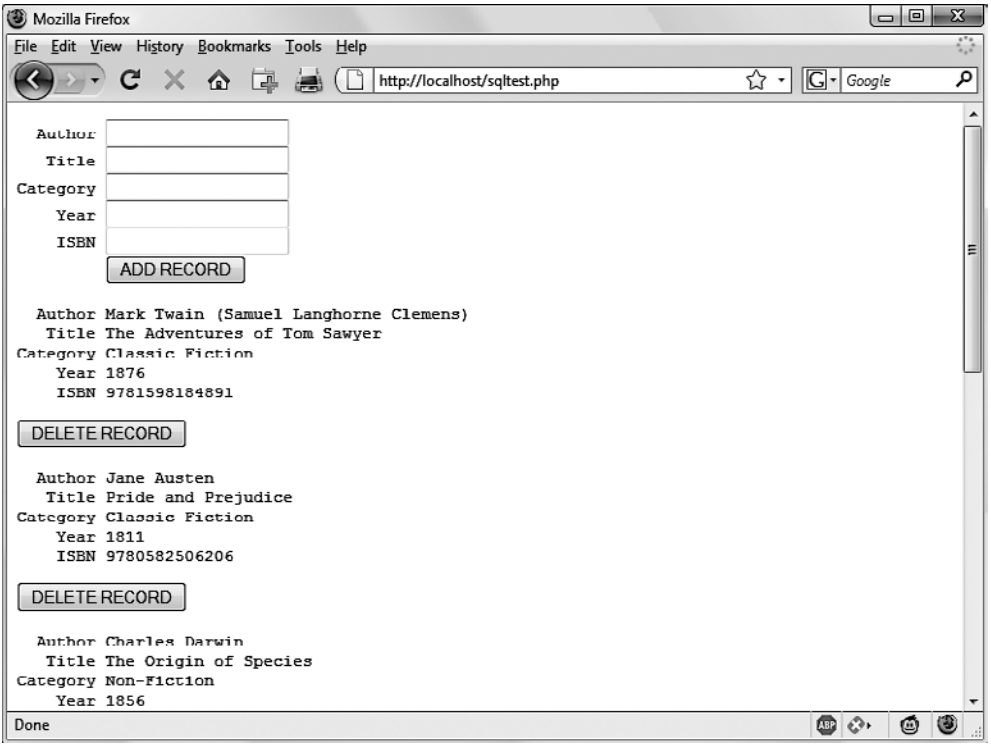

Рис. 10.2. Вид экрана, получаемый в результате работы кода из примера 10.6, сохраненного в файле sqltest.php

В примере 10.6 создается стандартная HTML-форма. Более подробно такие формы рассматриваются в главе 11, а здесь обработка формы используется только для демонстрации взаимодействия с базой данных.

Пример 10.6. Вставка и удаление данных с помощью программы sqltest.php

```
<?php // sqltest.php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
```

```
  if ($conn->connect_error) die("Fatal Error");

  if (isset($_POST['delete']) && isset($_POST['isbn']))
  {
    $isbn   = get_post($conn, 'isbn');
    $query = "DELETE FROM classics WHERE isbn='$isbn'";
    $result = $conn->query($query);
    if (!$result) echo "Сбой при удалении данных<br><br>";
  }

  if (isset($_POST['author'])   &&
      isset($_POST['title'])    &&
      isset($_POST['category']) &&
      isset($_POST['year'])     &&
      isset($_POST['isbn']))
  {
    $author   = get_post($conn, 'author');
    $title    = get_post($conn, 'title');
    $category = get_post($conn, 'category');
    $year     = get_post($conn, 'year');
    $isbn     = get_post('isbn');
    $query    = "INSERT INTO classics VALUES" .
      "('$author', '$title', '$category', '$year', '$isbn')";
    $result = $conn->query($query);
    if (!$result) echo "Сбой при вставке данных<br><br>";
  }

  echo <<<_END
  <form action="sqltest.php" method="post"><pre>
    Author <input type="text" name="author">
     Title <input type="text" name="title">
  Category <input type="text" name="category">
      Year <input type="text" name="year">
      ISBN <input type="text" name="isbn">
           <input type="submit" value="ADD RECORD"> // Кнопка ДОБАВИТЬ ЗАПИСЬ
  </pre></form>
_END;

  $query = "SELECT * FROM classics";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result-> fetch_arrow(MYSQLI_NUM);

    $r0 = htmlspecialchars($row[0]);
    $r1 = htmlspecialchars($row[1]);
    $r2 = htmlspecialchars($row[2]);
    $r3 = htmlspecialchars($row[3]);
    $r4 = htmlspecialchars($row[4]);

    echo <<<_END
```

```
  <pre>
    Author $r0
     Title $r1
  Category $r2
      Year $r3
      ISBN $r4
  </pre>
  <form action= 'sqltest.php' method= 'post'>
  <input type= 'hidden' name= 'delete' value= 'yes'>
  <input type= 'hidden' name= 'isbn' value= '$r4'>
  <input type= 'submit' value= 'DELETE RECORD'></form> // Кнопка УДАЛИТЬ ЗАПИСЬ
_END;
  }

  $result->close();
  $conn->close();

  function get_post($conn, $var)
  {
    return $conn->real_escape_string($_POST[$var]);
  }
?>
```

Эта программа со своими более чем 80 строками кода может показаться пугающей, но не стоит переживать — многие из этих строк уже были изучены в примере 10.5, а в работе этого кода нет ничего сложного.

Сначала выполняется проверка всех введенных данных, а затем в соответствии с предоставленным вводом осуществляется либо вставка новых данных в таблицу `classics` базы данных `publications`, либо удаление строки из этой таблицы. Независимо от того, что именно было введено, программа вслед за этим выводит все строки таблицы в браузер. Посмотрим, как все это работает.

Первая часть нового кода начинается с использования функции `isset`, чтобы проверить, были ли отправлены программе значения для всех полей. На основании такого подтверждения каждая из строк, находящихся внутри инструкции `if`, вызывает функцию `get_post`, которая появляется в самом конце программы. Эта функция делает небольшую, но очень важную работу: извлекает введенные данные из браузера.

Из соображений ясности и краткости, а также в целях максимально возможного упрощения объяснений из многих последующих примеров убраны весьма нужные фрагменты кода, отвечающие за соблюдение мер безопасности, поскольку они бы сделали примеры длиннее и отвлекали бы от четкого объяснения функций, заложенных в этих примерах. Поэтому важно не пропустить изучение следующего далее в этой главе раздела «Предотвращение попыток взлома», где рассматриваются дополнительные меры по защите ваших баз данных, которые можно предпринять в отношении кода, чтобы сделать его безопасным.

Массив $_POST

В одной из предыдущих глав я уже упоминал о том, что браузер отправляет пользовательский ввод в виде GET- или POST-запроса. Обычно предпочтение отдается POST-запросу, поэтому именно он здесь и используется. Веб-сервер объединяет все введенное пользователем (даже если в форме заполнено под сотню полей) и помещает его в массив `$_POST`.

`$_POST` относится к ассоциативным массивам, рассмотренным в главе 6. Данные формы будут помещаться в ассоциативный массив по имени `$_POST` или `$_GET` в зависимости от того, какой метод используется для отправки формы — POST или GET. Оба этих массива могут быть прочитаны абсолютно одинаковым способом.

У каждого поля есть элемент в массиве, имеющий точно такое же имя. Поэтому, если в форме есть поле `isbn`, в массиве `$_POST` будет элемент с ключом `isbn`. PHP-программа может прочитать это поле, ссылаясь на него либо в виде `$_POST['isbn']`, либо в виде `$_POST["isbn"]` (в данном случае одинарные и двойные кавычки обладают одинаковым действием).

Если синтаксис работы с массивом `$_POST` кажется вам слишком сложным, можно спокойно воспользоваться приемом, показанным в примере 10.6: скопировать введенные пользователем данные в другие переменные, после чего о массиве `$_POST` можно будет забыть. Для PHP-программ это вполне нормальный подход: они получают все поля из массива `$_POST` в самом начале программы и больше к нему не обращаются.

Записывать элемент в массив $_POST нет никакого смысла. Он предназначен только для передачи информации из браузера в программу, и перед его изменением лучше скопировать данные в свои собственные переменные.

Возвращаясь к функции `get_post`, следует отметить, что она пропускает каждый получаемый ею элемент через метод `real_escape_string` объекта подключения, чтобы удалить любые цитаты, которые злоумышленник может вставить, пытаясь взломать или изменить вашу базу данных:

```
function get_post($conn, $var)
{
  return $conn->real_escape_string($_POST[$var]);
}
```

Удаление записи

Перед проверкой отправки новых данных программа проверяет, есть ли значение у переменной `$_POST['delete']`. Если у нее есть значение, то пользователь нажал кнопку `DELETE RECORD` (Удалить запись). В этом случае должно быть отправлено и значение `$isbn`.

Как уже говорилось, номер ISBN является уникальным идентификатором каждой записи. HTML-форма добавляет ISBN к строке запроса `DELETE FROM`, создаваемой

в переменной $query, которая затем передается методу query объекта $conn, чтобы этот запрос попал к MySQL.

Если значение для $_POST['delete']) не установлено (и поэтому нет записи для удаления), проверяются $_POST['author']) и другие отправляемые значения. Если всем им присвоены значения, переменной $query присваивается текст команды INSERT INTO, за которым перечисляются пять вставляемых значений. Затем эта строка передается методу query, который по завершении своей работы возвращает либо TRUE, либо FALSE. Если возвращено значение FALSE, выводится сообщение об ошибке:

```
if (!$result) echo "INSERT failed"<br><br>";
```

Отображение формы

Перед выводом на экран небольшой формы (показанной на рис. 10.2) программа обезвреживает копии элементов, которые будут выводиться из массива $row, передавая эти копии функции htmlspecialchars с целью замены всех потенциально опасных HTML-символов безопасными HTML-элементами, а затем помещает получившиеся данные в переменные от $r0 и до $r4.

После этого следует та часть кода, которая отображает данные, используя структуру echo <<<_END..._END, выводит на экран все, что находится между тегами _END, и должна быть вам знакома по предыдущим главам.

Вместо команды echo программа может завершить работу с интерпретатором PHP, используя тег ?>, выдать код HTML, а затем опять вернуться к работе с интерпретатором PHP, используя тег <?php. Какой из стилей применять — дело вкуса программиста, но я всегда рекомендую оставаться в рамках PHP-кода в силу следующих причин:

- при отладке (а также при разборе кода другими пользователями) это дает абсолютную уверенность в том, что все содержимое файла .php является кодом PHP. Поэтому не возникает нужды отлавливать временные выходы в код HTML;
- когда значение переменной PHP нужно вставить непосредственно в код HTML, можно просто набрать ее имя внутри этого кода. А при выходе в HTML нужно будет временно вернуться к обработке PHP, вывести переменную, а затем снова вернуться в HTML.

Раздел HTML-формы направляет все, что сделано в форме, в адрес sqltest.php. Это означает, что при отправке формы содержимое ее полей будет передано файлу sqltest.php, в котором и хранится сама программа. Форма настроена также на отправку полей в POST-, а не в GET-запросе. Причина в том, что GET-запросы являются дополнением к отправляемому URL-адресу и могут иметь в вашем браузере неприглядный вид. Эти запросы также позволяют пользователям без особого труда вносить изменения в отправляемую информацию и предпринимать попытки взлома вашего сервера (хотя того же самого можно добиться с помощью встроенных в браузер инструментов разработчика). Кроме этого, отказ от использования GET-запросов уберегает от появления в регистрационных файлах сервера слишком

большого объема информации. Поэтому при малейшей возможности нужно использовать для отправки данных POST-запросы, которые к тому же имеют преимущество, позволяющее скрыть отправляемые данные от просмотра.

При выводе полей формы HTML отображает кнопку отправки с именем ADD RECORD (Добавить запись) и форма закрывается. Обратите внимание на теги `<pre>` и `</pre>`, позволяющие воспользоваться моноширинным шрифтом и выровнять по линейке все элементы ввода данных. Внутри тегов `<pre>` в выводимые данные попадают также символы возврата каретки, стоящие в конце каждой строки.

Запросы к базе данных

Далее код программы возвращается в привычное для нас русло примера 10.5, где в адрес MySQL отправляется запрос на выдачу всех записей в таблице `classics`:

```
$query = "SELECT * FROM classics";
$result = $conn->query($query);
```

Затем переменной `$rows` присваивается значение, равное количеству строк в таблице:

```
$rows = $result->num_rows;
```

После этого с использованием значения в `$rows` запускается цикл `for`, предназначенный для вывода на экран содержимого каждой строки.

Затем программа заполняет массив `$row` строкой результатов, для чего вызывается метод `fetch_array` объекта `$result`, которому передается значение константы `MYSQLI_NUM`, заставляющее возвратить числовой (а не ассоциативный) массив:

```
$row = $result->fetch_array(MYSQLI_NUM)
```

После того как данные попали в массив `$row`, они могут быть без особых усилий выведены на экран с помощью последующей heredoc-инструкции `echo`, где я решил использовать тег `<pre>`, чтобы выровнять на экране каждую запись, придав всему изображению привлекательный вид.

После отображения каждой записи следует вторая форма, которая также отправляет все свои данные файлу `sqltest.php` (то есть самой программе), но теперь в форме есть два скрытых поля: `delete` и `isbn`. Поле `delete` устанавливается в `yes`, а полю `isbn` присваивается значение, сохраненное в элементе массива `$row[4]`, в котором содержится ISBN для этой записи.

Далее отображается кнопка отправки с надписью DELETE RECORD (Удалить запись) и форма закрывается. Затем фигурная скобка закрывает тело цикла `for`, который продолжает работу до тех пор, пока не будут отображены все записи, и в этот момент ресурсы возвращаются PHP-программе, для чего вызываются методы `close`, принадлежащие объектам `$result` и `$conn`:

```
$result->close();
$conn->close();
```

В самом конце программы дано определение функции `get_post`, которую мы уже рассматривали. Вот так выглядит наша первая PHP-программа, предназначенная для управления базой данных MySQL. А теперь проверим, на что она способна.

После набора программы (и исправления всех опечаток) попробуйте ввести в поля ввода следующие сведения о книге *Moby Dick*, которые предназначены для добавления новой записи к базе данных:

```
Herman Melville
Moby Dick
Fiction
1851
9780199535729
```

Запуск программы

Когда эти данные будут отправлены нажатием кнопки ADD RECORD (Добавить запись), прокрутите веб-страницу до самого конца, чтобы посмотреть только что добавленную информацию. Ее предполагаемый вид показан на рис. 10.3.

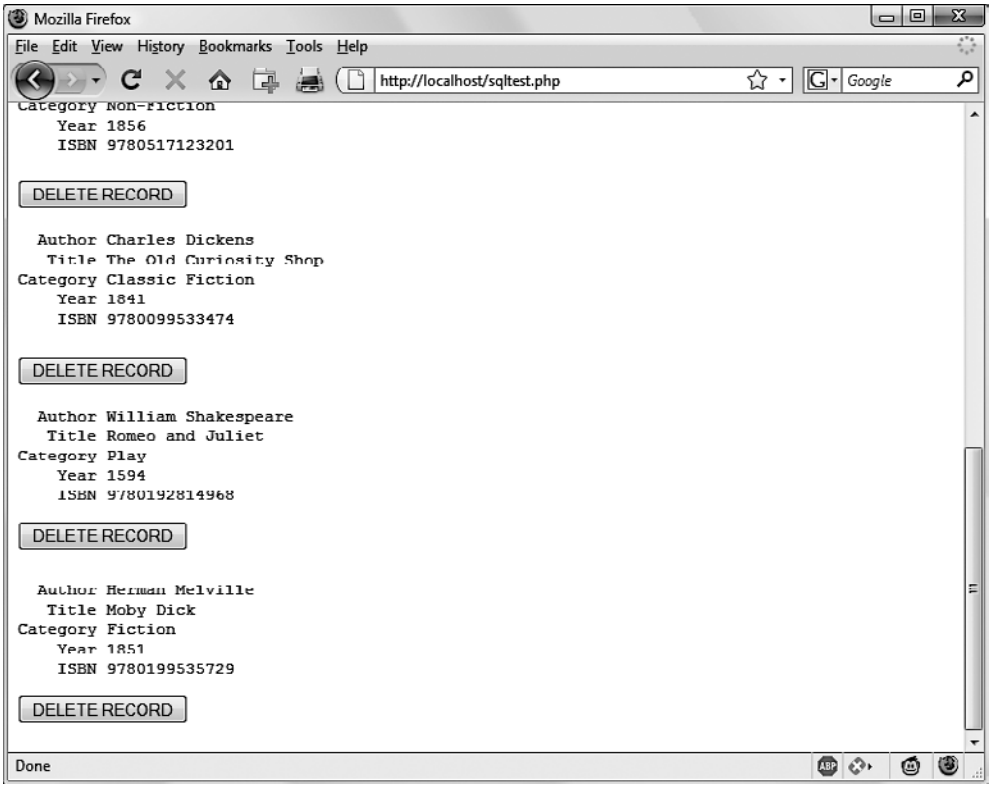

Рис. 10.3. Результат добавления сведений о книге Moby Dick в базу данных

Теперь посмотрим, как работает удаление записи, специально создав для этого ненужную запись. Попробуйте ввести во все пять полей только одну цифру 1 и нажмите кнопку ADD RECORD (Добавить запись). Если теперь прокрутить страницу вниз, станет видна новая запись, состоящая из одних единиц. Конечно, такая запись в таблице не нужна, поэтому нажмите кнопку DELETE RECORD (Удалить запись) и снова прокрутите страницу вниз, чтобы убедиться в том, что запись была удалена.

Теперь, если все получилось, вы можете добавлять и удалять записи по своему усмотрению. Попробуйте сделать все это несколько раз, но основные записи (включая и новую запись о книге Moby Dick) оставьте нетронутыми, поскольку они нам еще пригодятся. Можно также попробовать добавить запись, состоящую из одних единиц, два раза и посмотреть, как при второй попытке будет выведено сообщение об ошибке, в котором говорится о том, что в таблице уже есть запись со значением ISBN, равным единице.

Практическая работа с MySQL

Теперь вы готовы к изучению некоторых практических приемов, которые можно использовать в PHP для доступа к базам данных MySQL, куда включены задачи создания и удаления таблиц, вставки, обновления и удаления данных, а также защиты вашей базы данных и сайта от злоумышленников. Учтите, что в следующих примерах предполагается, что вы создали программу `login.php`, рассмотренную ранее в этой главе.

Создание таблицы

Предположим, что зоопарк поручил вам создать базу данных со сведениями обо всех содержащихся в нем представителях семейства кошачьих. Вам сказали, что в зоопарке девять представителей кошачьих: лев, тигр, ягуар, леопард, пума, гепард, рысь, каракал и домашний кот. Вам необходим отдельный столбец для вида кошачьих. У каждого животного есть кличка, и для нее нужен еще один столбец. Необходимо также отслеживать возраст животных, и для этого требуется еще один столбец.

Разумеется, позже могут понадобиться дополнительные столбцы, например для учета рациона питания, сделанных прививок и других сведений, но пока, чтобы приступить к работе, достаточно и этих столбцов. Каждому животному нужен также уникальный идентификатор, поэтому решено было создать для него столбец `id`.

В примере 10.7 показан код, который можно использовать для создания таблицы MySQL, хранящей все эти сведения. Операция присваивания, относящаяся к главному запросу, выделена полужирным шрифтом.

Пример 10.7. Создание таблицы cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "CREATE TABLE cats (
    id SMALLINT NOT NULL AUTO_INCREMENT,
    family VARCHAR(32) NOT NULL,
    name VARCHAR(32) NOT NULL,
    age TINYINT NOT NULL,
    PRIMARY KEY (id)
  )";

  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");
?>
```

Как видите, MySQL-запрос очень похож на тот запрос, который приходилось набирать непосредственно в командной строке, за исключением того, что в нем нет завершающей точки с запятой.

Описание таблицы

Если вы не вошли в командную строку MySQL, то можно воспользоваться весьма полезным фрагментом кода, позволяющим проверить в браузере факт успешного создания таблицы. Этот код просто выдает запрос `DESCRIBE cats`, а затем выводит HTML-таблицу, имеющую четыре заголовка: Column (Графа), Type (Тип), Null (Нуль) и Key (Ключ), ниже которых отображаются все имеющиеся в таблице столбцы.

Чтобы использовать код примера 10.8 с другими таблицами, нужно просто заменить в запросе имя `cats` именем новой таблицы.

Пример 10.8. Описание таблицы cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "DESCRIBE cats";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");

  $rows = $result->num_rows;

  echo "<table><tr> <th>Column</th> <th>Type</th><th>Null</th> <th>Key</th> </tr>";

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_arrow(MYSQLI_NUM);

    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k)
```

```
      echo "<td>"  . htmlspecialchars($row[$k]) . "</td>";
    echo "</tr>";
  }

  echo "</table>";
?>
```

Информация, выводимая программой, должна иметь следующий вид:

```
Column  Type         Null  Key
id      smallint(6)  NO    PRI
family  varchar(32)  NO
name    varchar(32)  NO
age     tinyint(4)   NO
```

Удаление таблицы

Удалить таблицу очень легко, и это весьма опасное действие нужно выполнять с большой осторожностью. В примере 10.9 показан необходимый для этого код. Но я не советую его применять до тех пор, пока не будут проработаны все остальные примеры (вплоть до подраздела «Выполнение дополнительных запросов»), поскольку в результате его выполнения будет удалена таблица `cats` и вам придется создавать ее заново с помощью кода, показанного в примере 10.7.

Пример 10.9. Удаление таблицы cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "DROP TABLE cats";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");
?>
```

Добавление данных

Добавим к таблице некоторые данные, воспользовавшись кодом, показанным в примере 10.10.

Пример 10.10. Добавление данных к таблице cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4)";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");
?>
```

Если изменить значение переменной `$query`, как показано далее, можно добавить еще два элемента данных и еще раз вызвать программу из вашего браузера:

```
$query = "INSERT INTO cats VALUES(NULL, 'Cougar', 'Growler', 2)";
$query = "INSERT INTO cats VALUES(NULL, 'Cheetah', 'Charly', 3)";
```

Кстати, вы заметили, что в качестве первого параметра было передано значение `NULL`? Это сделано потому, что столбец `id` имеет тип `AUTO_INCREMENT` и MySQL решит, что нужно присвоить следующее доступное в используемой последовательности значение, поэтому мы просто передаем значение `NULL`, которое будет проигнорировано.

Разумеется, наиболее эффективный способ заполнения MySQL данными заключается в создании массива и вставке данных с использованием только одного запроса.

На данный момент главное внимание уделяется демонстрации способов непосредственной вставки данных в MySQL (и предоставлению соответствующих мер безопасности). Но далее в книге мы перейдем к изучению более подходящего практического метода с использованием указателей мест заполнения (см. далее подраздел «Указатели мест заполнения»), который практически исключает возможность ввода пользователями вредоносного кода в вашу базу данных. Поэтому при чтении данного подраздела имейте в виду, что в нем просто рассматриваются основы работы MySQL-вставки, а в дальнейшем будут изучены способы усовершенствования этих базовых приемов.

Извлечение данных

Теперь, когда в таблицу `cats` введены некоторые данные, в примере 10.11 показано, как можно убедиться в том, что они были благополучно вставлены в эту таблицу.

Пример 10.11. Извлечение строк из таблицы cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "SELECT * FROM cats";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");

  $rows = $result-> num_rows;
  echo "<table><tr> <th>Id</th> <th>Family</th><th>Name</th><th>Age</th></tr>";

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_arrow(MYSQLI_NUM);

    echo "<tr>";
```

```
    for ($k = 0 ; $k < 4 ; ++$k)
      echo "<td>" . htmlspecialchars($row[$k]) . "</td>";
    echo "</tr>";
  }

  echo "</table>";
?>
```

Этот код выдает простой MySQL-запрос `SELECT * FROM cats`, а затем отображает все возвращенные строки. Выходная информация должна иметь следующий вид:

```
Id  Family  Name    Age
1   Lion    Leo     4
2   Cougar  Growler 2
3   Cheetah Charly  3
```

Здесь можно убедиться в том, что столбец `id` получает правильное автоприращение.

Обновление данных

Изменение внесенных в таблицу данных также выполняется очень просто. Вы заметили, что кличка гепарда (cheetah) записана как Charly? Исправим ее на Charlie, как показано в примере 10.12.

Пример 10.12. Переименование гепарда Charly в Charlie

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "UPDATE cats SET name='Charlie' WHERE name='Charly'";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");
?>
```

Если теперь еще раз запустить код из примера 10.11, то будет выведена следующая информация:

```
Id  Family  Name    Age
1   Lion    Leo     4
2   Cougar  Growler 2
3   Cheetah Charlie 3
```

Удаление данных

Пума по имени Growler была перевезена в другой зоопарк, поэтому сведения о ней нужно удалить из базы данных. Удаление данных из таблицы показано в примере 10.13.

Пример 10.13. Удаление сведений о пуме Growler из таблицы cats

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "DELETE FROM cats WHERE name='Growler'";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");
?>
```

Здесь используется стандартный запрос `DELETE FROM`, и, когда будет запущен код из примера 10.11, в отображаемой информации вы сможете увидеть, что строка была удалена:

```
Id  Family  Name    Age
1   Lion    Leo     4
3   Cheetah Charlie 3
```

Свойство AUTO_INCREMENT

При использовании свойства `AUTO_INCREMENT` вы не можете знать, какое значение было дано столбцу до вставки строки. Но если нужно узнать об этом, можно позже обратиться к MySQL с помощью свойства `mysql_insert_id` объекта подключения. Потребность в этом возникает довольно часто, например при обработке покупки можно вставить в таблицу `Customers` нового покупателя, а затем сослаться на только что созданный столбец `CustId` при вставке покупки в таблицу покупок. Пример 10.10 может быть переписан и превращен в пример 10.14, отображающий это значение после каждой вставки.

Вместо выбора наибольшего по значению идентификатора в столбце id и увеличения его значения на единицу рекомендуется использовать свойство AUTO_INCREMENT, поскольку одновременно выполняемые запросы могут после извлечения наибольшего значения изменить значения в данном столбце еще до того, как будет сохранено новое вычисленное значение.

Пример 10.14. Добавление данных к таблице cats и отчет о вставленном ID

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");

  echo "Значение вставленного ID равно " . $conn->insert_id;
?>
```

Теперь содержимое таблицы приобретет следующий вид (обратите внимание на то, что ранее использовавшееся значение ID, равное 2, повторно *не* используется, поскольку в некоторых случаях это может вызвать определенные сложности):

```
Id Family  Name    Age
1  Lion    Leo     4
3  Cheetah Charlie 3
4  Lynx    Stumpy  5
```

Идентификаторы вставленных строк

Зачастую данные вставляются сразу в несколько таблиц: за книгой следует ее автор, за покупателем — его покупка и т. д. При вставке данных в столбцы с автоприращением нужно будет запомнить вставленный ID, возвращенный для его сохранения в связанной таблице.

Предположим, что для привлечения дополнительных средств над представителями кошачьих могут брать шефство какие-нибудь организации, и когда животное сохраняется в таблице кошек, нам нужно создать ключ для привязки его к организации, взявшей над ним шефство. Код для этого похож на код примера 10.14, за исключением того, что возвращенный ID, который был вставлен в таблицу, сохраняется в переменной `$insertID`, а затем используется в качестве составной части такого запроса:

```
$query    = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";
$result   = $conn->query($query);
$insertID = $conn->insert_id;

$query    = "INSERT INTO owners VALUES($insertID, 'Ann', 'Smith')";
$result   = $conn->query($query);
```

Теперь животное связано со своим шефом посредством уникального кошачьего идентификатора, который был автоматически создан благодаря свойству `AUTO_INCREMENT`.

Выполнение дополнительных запросов

Итак, хватит развлекаться с кошками. Чтобы исследовать более сложные запросы, нужно вернуться к использованию таблиц `customers` и `classics`, которые вы должны были создать в процессе работы с главой 8. В таблице `customers` должны быть сведения о двух покупателях, а в таблице `classics` — сведения о нескольких книгах. В них также совместно используется столбец с номером ISBN, названный `isbn`, который можно применять для выполнения дополнительных запросов.

Например, для отображения всех покупателей вместе с названиями и авторами купленных ими книг можно воспользоваться кодом, показанным на рис. 10.15.

Пример 10.15. Выполнение вторичного запроса

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $query = "SELECT * FROM customers";
  $result = $conn->query($query);
  if (!$result) die ("Сбой при доступе к базе данных");

  $rows = $result->num_rows;

  for ($j = 0 ; $j < $rows ; ++$j)
  {
    $row = $result->fetch_arrow(MYSQLI_NUM);
    echo htmlspecialchars($row[0]) . " purchased ISBN " .
         htmlspecialchars($row[1]) . ":<br>";

    $subquery = "SELECT * FROM classics WHERE isbn='$row[1]'";
    $subresult = $conn->query($subquery);
    if (!$subresult) die ("Сбой при доступе к базе данных");

    $subrow = $subresult->fetch_arrow(MYSQLI_NUM);
    echo "  " . htmlspecialchars("'$subrow[1]'") . " by " .
                          htmlspecialchars( $subrow[0]) . "<br><br>";
  }
?>
```

В этой программе используется первичный запрос к таблице `customers`, чтобы найти всех покупателей. Затем на основе номера ISBN книг, приобретенных каждым покупателем, делается новый запрос к таблице `classics`, чтобы найти название и автора каждой из книг. При выполнении этого кода будет выведена следующая информация:

```
Joe Bloggs purchased ISBN 9780099533474:
  'The Old Curiosity Shop' by Charles Dickens

Jack Wilson purchased ISBN 9780517123201:
  'The Origin of Species' by Charles Darwin

Mary Smith purchased ISBN 9780582506206:
  'Pride and Prejudice' by Jane Austen
```

Хотя это и не имеет отношения к иллюстрации использования дополнительных запросов, в данном случае можно вернуть точно такую же информацию, используя запрос с видом объединения NATURAL JOIN (см. главу 8):

```
SELECT name,isbn,title,author FROM customers
NATURAL JOIN classics;
```

Предотвращение попыток взлома

Наверное, опасность передачи MySQL не прошедшей проверку вводимой пользователем информации все же недооценивается. Представьте, к примеру, что для идентификации пользователя применяется следующий простой фрагмент кода:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

На первый взгляд этот код может показаться вполне приемлемым. Если пользователь вводит значения `fredsmith` и `mypass`, которые присваиваются переменным `$user` и `$pass` соответственно, то строка запроса, которая передается MySQL, приобретает следующий вид:

```
SELECT * FROM users WHERE user='fredsmith' AND pass='mypass'
```

Все это выглядит вполне пристойно, но что будет, если кто-нибудь введет для переменной `$user` следующую строку (а для переменной `$pass` вообще ничего не станет вводить):

```
admin' #
```

Посмотрим на строку, которая будет отправлена MySQL:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Вы поняли, в чем тут проблема? Произошла атака *SQL-инъекции*. В MySQL с символа # начинается комментарий. Поэтому пользователь войдет в систему под именем `admin` (предположим, что в системе есть пользователь `admin`), и ему не нужно будет вводить пароль. В следующей строке исполняемая часть запроса выделена полужирным шрифтом, а все остальные символы будут проигнорированы.

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Но вам еще очень посчастливится, если весь вред, нанесенный злоумышленником, этим и ограничится. По крайней мере вам, может быть, удастся войти в свое приложение и отменить все изменения, внесенные туда пользователем, вошедшим в систему под именем `admin`. А что, если ваша прикладная программа удаляет пользователя из базы данных? Тогда код может иметь следующий вид:

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$query = "DELETE FROM users WHERE user='$user' AND pass='$pass'";
```

На первый взгляд, с ним опять вроде бы все в порядке, но что получится, если кто-нибудь введет для переменной `$user` следующее значение:

```
anything' OR 1=1 #
```

MySQL интерпретирует это следующим образом (также выделено полужирным шрифтом):

```
DELETE FROM users WHERE user='anything' OR 1=1 #' AND pass=''
```

Условие этого SQL-запроса всегда будет истинным, поэтому вы лишитесь всей базы данных, в которой содержатся сведения о пользователях. Так что же нужно делать в случае подобной атаки?

Возможные меры противодействия

Не стоит полагаться на встроенное в PHP свойство «волшебных кавычек» (magic quotes), которое автоматически отключает любые символы одинарных и двойных кавычек путем установки перед ними символа обратного слеша (\). Почему? Да потому, что это свойство может быть отключено. Многие программисты именно так и делают, чтобы вставить собственный код, обеспечивающий безопасность. Поэтому нет никаких гарантий того, что на вашем рабочем сервере это свойство не было отключено. Фактически в PHP 5.3.0 его использование не приветствовалось, а из PHP 5.4.0 оно вообще было удалено.

Вместо этого для всех обращений к MySQL нужно всегда использовать метод `real_escape_string`. В примере 10.16 показана функция, которую можно использовать, чтобы удалить любые «волшебные кавычки», добавленные во введенную пользователем строку, а затем соответствующим образом обезвредить все имеющиеся в ней опасные компоненты.

Пример 10.16. Способ обезвреживания данных, введенных пользователем, приемлемый для MySQL

```
<?php
  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Функция `get_magic_quotes_gpc` возвращает `TRUE`, если свойство «волшебных кавычек» находится в активном состоянии. Если это так, любые добавленные к строке слеши подлежат удалению, в противном случае метод `real_eascape_string` может отключить некоторые символы дважды, сделав строки непригодными для дальнейшего использования. В примере 10.17 показано, как можно вставить функцию `mysql_fix_string` в ваш код.

Пример 10.17. Способ безопасного доступа к MySQL при использовании данных, введенных пользователем

```
<?php
  require_once 'login.php';
```

```
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $user = mysql_fix_string($conn, $_POST['user']);
  $pass = mysql_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

  // И т. д.

  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Актуальность этих мер предосторожности снижается, поскольку есть более простой и безопасный способ доступа к MySQL, исключающий необходимость использования подобного рода функций и заключающийся в применении рассматриваемых далее указателей мест заполнения.

Указатели мест заполнения

Все рассмотренные до сих пор способы работают с MySQL, но требуют применения особых мер безопасности, поскольку строки постоянно нужно обезвреживать. Но теперь, после изучения основ, давайте приступим к более рациональному и рекомендуемому способу работы с MySQL, существенно превосходящему по степени защищенности прежние методы работы. Как только вы усвоите материал этого подраздела, нужно будет отказаться от непосредственной вставки данных в MySQL (хотя, как это делается, было важно показать) и постоянно использовать исключительно указатели мест заполнения,

Так что же собой представляют эти указатели? Это позиции внутри подготовленных инструкций, в которых данные переносятся непосредственно в базу данных, что исключает возможность интерпретации переданных пользователем (или кем-то другим) данных в качестве MySQL-инструкций (и возможность потенциального взлома в результате такой интерпретации).

Эта технология требует от вас сначала определить запрос для выполнения в MySQL, в котором в местах, ссылающихся на данные, используются простые вопросительные знаки.

В обычном MySQL предварительно определенные запросы имеют вид, показанный в примере 10.18.

Пример 10.18. Использование указателей мест заполнения

```
PREPARE statement FROM "INSERT INTO classics VALUES(?,?,?,?,?)";

SET @author   = "Emily Brontë",
    @title    = "Wuthering Heights",
    @category = "Classic Fiction",
    @year     = "1847",
    @isbn     = "9780553212587";

EXECUTE statement USING @author,@title,@category,@year,@isbn;
DEALLOCATE PREPARE statement;
```

Чтобы не усложнять передачу запросов MySQL, расширение `mysqli` упрощает обработку указателей мест заполнения, предоставляя готовый метод под названием `prepare`, вызываемый следующим образом:

```
$stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
```

Объект `$stmt` (сокращение от слова «инструкция» — statement), возвращаемый этим методом, используется затем для отправки на сервер данных, замещающих вопросительные знаки. В первую очередь он применяется для последовательной привязки к каждому вопросительному знаку (к параметрам указателей мест заполнения) PHP-переменных:

```
$stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);
```

Первым аргументом метода `bind_param` является строка, представляющая собой череду типов аргументов. В данном случае в ней содержатся пять символов `s`, представляющих строковые значения, но здесь могут указываться любые комбинации следующих типов:

- `i` — данные, являющиеся целым числом;
- `d` — данные, являющиеся числом с двойной точностью;
- `s` — данные, являющиеся строкой;
- `b` — данные, являющиеся большим двоичным объектом — BLOB (отправляемым в пакетах).

После того как переменные привязаны к предварительно определенным запросам, необходимо присвоить этим переменным значения данных, чтобы можно было передать их MySQL:

```
$author   = 'Emily Brontë':
$title    = 'Wuthering Heights':
$category = 'Classic Fiction':
$year     = '1847';
$isbn     = '9780553212587';
```

Теперь у сценария PHP есть все, что нужно, чтобы выполнить предварительно определенный запрос, поэтому мы выдаем следующую команду, вызывающую метод `execute` ранее созданного объекта `$stmt`:

```
$stmt->execute();
```

Перед тем как продолжить, имеет смысл провести очередную проверку на успешное выполнение команды, что можно сделать, проверив значение свойства `affected_rows` объекта `$stmt`:

```
printf("%d Row inserted.\n", $stmt->affected_rows);
```

В данном случае выводимая информация должна свидетельствовать о вставке одной строки.

Как только вы порадовались успешному выполнению запроса (или же столкнулись с какими-либо ошибками), объект `$stmt` можно закрывать:

```
$stmt->close();
```

И в последнюю очередь закройте объект `$conn` (при условии, что с ним вы также закончили работу):

```
$conn->close();
```

Когда все будет собрано вместе, получится результат, показанный в примере 10.19.

Пример 10.19. Использование указателей мест заполнения с PHP

```
<?php
  require 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
  $stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);

  $author   = 'Emily Brontë';
  $title    = 'Wuthering Heights';
  $category = 'Classic Fiction';
  $year     = '1847';
  $isbn     = '9780553212587';

  $stmt->execute();
  printf("%d Row inserted.\n", $stmt->affected_rows);
  $stmt->close();
  $conn->close();
?>
```

При каждом использовании предварительно определяемых запросов вместо изначально готовых запросов вы будете закрывать потенциальную брешь в системе безопасности, поэтому стоит потратить время на изучение способов их использования.

Предотвращение внедрения HTML-кода

Нужно позаботиться о защите еще от одного вида внедрения, который связан не с безопасностью ваших собственных сайтов, а с конфиденциальностью и защитой пользовательских данных. Речь идет о *межсайтовом выполнении сценариев* (Cross Site Scripting), называемом также *XSS-атакой*.

Эта разновидность внедрения происходит в том случае, когда вы разрешаете пользователю вводить, а затем отображать на вашем сайте HTML-, или, что случается чаще, JavaScript-код. Одним из мест, где это часто происходит, является форма для комментариев. Обычно злоумышленник пытается написать код, ворующий у пользователей вашего сайта cookie, позволяющие ему узнать пары «имя пользователя — пароль», если они обрабатываются недостаточно безопасно, или другую информацию, позволяющую захватить сеанс работы (в котором пользовательская учетная запись захвачена злоумышленником, получившим теперь возможность пользоваться ею!). Или же злоумышленник может предпринять атаку с целью загрузки на пользовательский компьютер троянского коня.

Чтобы предотвратить это внедрение, нужно лишь вызвать функцию `htmlentities`, выявляющую все коды разметки HTML и заменяющую их формой, которая отображает символы, но не позволяет браузеру действовать в соответствии с их предназначением. Рассмотрим, к примеру, следующий код HTML:

```
<script src='http://x.com/hack.js'>
</script><script>hack();</script>
```

Этот код загружает программу на JavaScript, а затем выполняет вредоносные функции. Но если сначала этот код будет пропущен через функцию `htmlentities`, то он превратится в такую абсолютно безвредную строку:

```
&lt;script src='http://x.com/hack.js'&gt; &lt;/script&gt;
&lt;script&gt;hack();&lt;/script&gt;
```

Поэтому если вы когда-нибудь соберетесь отобразить какие-нибудь данные, введенные пользователем, то нужно немедленно или сразу же после первого сохранения в базе данных обезвредить их с помощью функции `htmlentities`. Для этого я рекомендую вам создать новую функцию наподобие первой функции, показанной в примере 10.20, которая способна обезвредить попытки как SQL-, так и XSS-внедрения.

Пример 10.20. Функции для предотвращения атак внедрения SQL и XSS

```
<?php
  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }

  function mysql_fix_string($conn, $string)
```

```
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Функция `mysql_entities_fix_string` сначала вызывает функцию `mysql_fix_string`, а затем, прежде чем вернуть полностью обезвреженную строку, пропускает результат через функцию `htmlentities`. Для использования любой из этих функций у вас уже должен быть активный объект подключения к базе данных MySQL.

В примере 10.21 показана новая, «более защищенная» версия примера 10.17.

Пример 10.21. Способ безопасного доступа к MySQL и предотвращения XSS-атак

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Fatal Error");

  $user = mysql_entities_fix_string($conn, $_POST['user']);
  $pass = mysql_entities_fix_string($conn, $_POST['pass']);
  $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

  // И т. д....

  function mysql_entities_fix_string($conn, $string)
  {
    return htmlentities(mysql_fix_string($conn, $string));
  }

  function mysql_fix_string($conn, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
  }
?>
```

Процедурный метод использования mysqli

Если вы предпочитаете обращаться к `mysqli` в процедурной (а не в объектно-ориентированной) манере, то для этого предусмотрен альтернативный набор функций. Например, вместо создания объекта `$conn`:

```
$conn = new mysqli($hn, $un, $pw, $db);
```

можно воспользоваться следующим кодом:

```
$link = mysqli_connect($hn, $un, $pw, $db);
```

Для проверки факта подключения и управления этим подключением можно воспользоваться следующим кодом:

```
if (mysqli_connect_errno()) die("Fatal Error");
```

А для осуществления запроса к MySQL можно использовать такой код:

```
$result = mysqli_query($link, "SELECT * FROM classics");
```

После возвращения управления из этого кода в переменной `$result` будут содержаться данные. Определить количество возвращенных строк можно с помощью следующего кода:

```
$rows = mysqli_num_rows($result);
```

В переменной `$rows` будет возвращено целочисленное значение. Построчное извлечение данных можно осуществить следующим способом, возвращающим числовой массив:

```
$row = mysqli_fetch_array($result, MYSQLI_NUM);
```

В данном примере элемент `$row[0]` будет содержать первый столбец данных, элемент `$row[1]` — второй столбец и т. д. Как показано ранее в пункте «Извлечение строки» на с. 273, строки могут также быть возвращены в виде ассоциативных массивов или в виде массивов обоих типов в зависимости от значения, переданного во втором аргументе.

Когда после операции вставки нужно узнать о вставленном идентификаторе (ID), всегда можно вызвать функцию `mysqli_insert_id`:

```
$insertID = mysqli_insert_id($result);
```

Процедурный способ обезвреживания строк средствами `mysqli` сводится к простому использованию следующего кода:

```
$escaped = mysqli_real_escape_string($link, $val);
```

Подготовка запроса с помощью `mysqli` также не представляет особой сложности:

```
$stmt = mysqli_prepare($link, 'INSERT INTO classics VALUES(?,?,?,?,?)');
```

Для привязки переменных к предварительно определенным запросам можно воспользоваться следующим кодом:

```
mysqli_stmt_bind_param($stmt, 'sssss', $author, $title, $category,
  $year, $isbn);
```

А для выполнения предварительно определенного запроса после присваивания переменным нужных значений можно использовать такой вызов:

```
mysqli_stmt_execute($stmt);
```

Чтобы закрыть запрос, нужно выдать следующую команду:

```
mysqli_stmt_close($stmt);
```

А чтобы закрыть подключение к MySQL, нужно воспользоваться такой командой:

```
mysqli_close($link);
```

Исчерпывающие подробности использования предварительно определенных запросов (процедурным или иным образом) можно найти в руководстве по PHP по адресу tinyurl.com/mysqlistmt, а советы по всем особенностям работы с mysqli находятся по адресу tinyurl.com/usingmysqli.

После изучения нескольких способов объединения PHP с MySQL в следующей главе мы перейдем к созданию удобных для пользователей форм и работе с данными, отправляемыми из этих форм.

Вопросы

1. Как подключиться к базе данных MySQL с помощью `mysqli`?
2. Как, используя `mysqli`, отправить запрос к MySQL?
3. Как извлечь строку, содержащую сообщение об ошибке, в случае возникновения ошибки в работе `mysqli`?
4. Как определить количество строк, возвращенных `mysqli`-запросом?
5. Как извлечь конкретную строку данных из набора `mysqli`-результатов?
6. Какой `mysqli`-метод может использоваться для обезвреживания пользовательского ввода и предотвращения внедрения вредоносного кода?
7. Какие негативные последствия могут наступить, если не закрыть объекты, созданные методами `mysqli`?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

11 Обработка форм

Основной способ взаимодействия пользователей с PHP и MySQL — применение HTML-форм. Эти формы появились на заре разработки Всемирной паутины, в 1993 году, даже раньше, чем электронная коммерция, и благодаря простоте и легкости использования не утратили своего значения и по сей день.

Разумеется, с годами HTML-формы совершенствовались, получая дополнительные функциональные возможности обработки информации, поэтому данная глава познакомит вас с современными методами обработки формы и продемонстрирует самые лучшие способы реализации форм для достижения наибольшей степени удобства и безопасности. Плюс к этому чуть позже вы увидите, что в спецификации HTML5 предусмотрено улучшенное использование форм.

Создание форм

Обработка форм — многоступенчатый процесс. Сначала создается форма, в которую пользователь может вводить необходимые данные. Затем эти данные отправляются веб-серверу, где происходит их разбор, зачастую совмещаемый с проверкой на отсутствие ошибок. Если код PHP найдет одно или несколько полей, требующих повторного ввода, форма может быть заново отображена вместе с сообщением об ошибке. Когда качество введенных данных удовлетворяет программу, она предпринимает некоторые действия, нередко привлекая для этого базы данных, к примеру для ввода сведений о покупке.

Для создания формы потребуются как минимум следующие элементы:

- открывающий и закрывающий теги — `<form>` и `</form>` соответственно;
- тип передачи данных, задаваемый одним из двух методов — GET или POST;
- одно или несколько полей для ввода данных;
- URL-адрес назначения, по которому будут отправлены данные формы.

В примере 11.1 показана очень простая форма, созданная с использованием кода PHP. Наберите этот код и сохраните его в файле `formtest.php`.

Пример 11.1. formtest.php — простой обработчик формы на PHP

```
<?php // formtest.php
  echo <<< END
    <html>
      <head>
        <title>Form Test</title>
      </head>
    <body>
    <form method="post" action="formtest.php">
      Как Вас зовут?
      <input type="text" name="name">
      <input type="submit">
    </form>
    </body>
  </html>
_END;
?>
```

В первую очередь следует отметить, что в этом примере использован прием, уже встречавшийся в данной книге: вместо многократного входа в PHP-код и выхода из него для вывода многострочного HTML-кода я обычно применяю конструкцию `echo <<<_END..._END`.

Внутри этого многострочного вывода находится стандартный код, с которого начинается HTML-документ, отображающий заголовок документа и обозначающий начало его тела. Затем следует форма, настроенная на отправку своих данных с использованием POST-метода в адрес PHP-программы `formtest.php`, то есть этой самой программы.

Остальной код программы закрывает все открытые элементы: форму, тело HTML-документа и PHP-инструкцию `echo <<<_END`. Результат запуска этой программы в браузере показан на рис. 11.1.

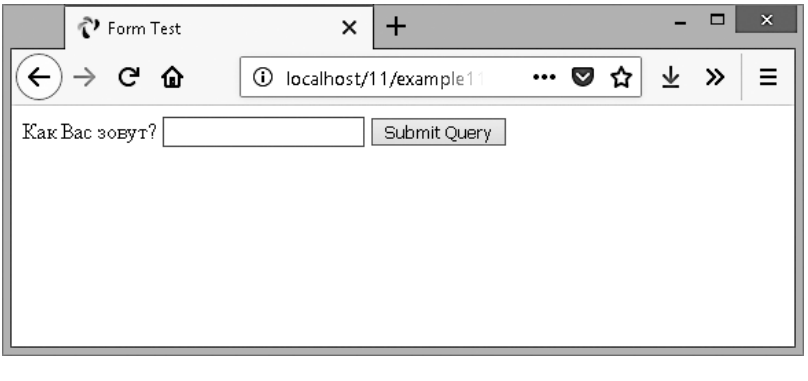

Рис. 11.1. Результат запуска программы formtest.php в браузере

Извлечение отправленных данных

В примере 11.1 представлена только одна из частей многоступенчатого процесса обработки формы. Если ввести имя и нажать кнопку Submit Query (Отправить запрос), то абсолютно ничего, кроме повторного отображения формы, не произойдет. Поэтому сейчас нужно добавить PHP-код, обрабатывающий отправляемые формой данные.

В примере 11.2 показана расширенная версия предыдущей программы, включающая обработку данных. Наберите этот код или измените код программы `formtest.php`, добавив в него новые строки, сохраните программу в файле `formtest2.php` и попробуйте ее запустить. Результат запуска программы и введенное имя показаны на рис. 11.2.

Пример 11.2. Обновленная версия formtest.php

```
<?php // formtest2.php
  if (isset($_POST['name'])) $name = $_POST['name'];
  else $name = "(Не введено)";

  echo <<<_END
  <html>
    <head>
      <title>Form Test</title>
    </head>
    <body>
      Вас зовут: $name<br>
      <form method="post" action="formtest2.php">
        Как Вас зовут?
        <input type="text" name="name">
        <input type="submit">
      </form>
    </body>
  </html>
_END;
?>
```

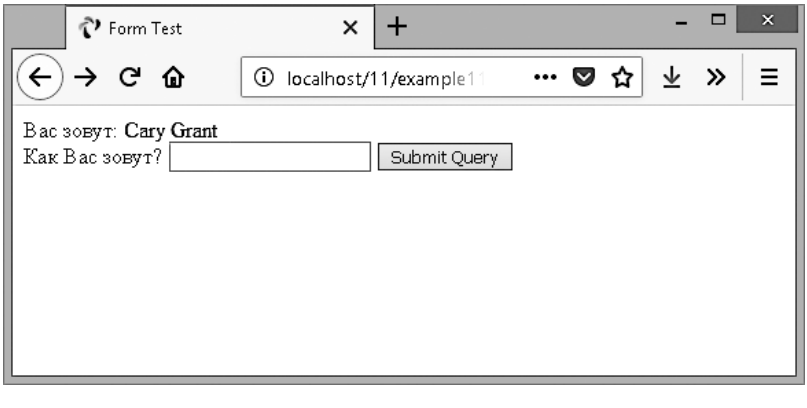

Рис. 11.2. Программа formtest.php с обработкой данных

Изменения касаются двух строк в начале программы, где проверяется содержимое поля name в ассоциативном массиве $_POST, которое отправляется назад пользователю с помощью команды echo. Ассоциативный массив $_POST был рассмотрен в главе 10, он включает в себя элемент для каждого поля HTML-формы. В примере 11.2 для вводимого имени использовалось поле name, а для отправки данных формы был избран метод POST, поэтому значение элемента name массива $_POST содержится в элементе массива $_POST['name'].

PHP-функция isset используется для проверки наличия значения у элемента $_POST['name']. Если значение не было отправлено, то программа присваивает переменной $name значение (Не введено). А если значение было отправлено, то оно сохраняется в этой переменной. После тега <body> была введена еще одна строка, предназначенная для отображения значения, сохраненного в переменной $name.

Значения по умолчанию

Иногда представляется удобным предложить посетителям вашего сайта принять в веб-форме значения по умолчанию. Предположим, вы разместили на сайте по недвижимости приложение, представляющее собой калькулятор погашения кредита. При этом есть смысл ввести в него значения по умолчанию, скажем 25 лет и 6 % годовых, чтобы пользователю осталось только ввести либо основную сумму заимствования, либо посильную для него сумму ежемесячных выплат. HTML-код для этих двух значений мог бы иметь вид, показанный в примере 11.3.

Пример 11.3. Установка значений по умолчанию

```
<form method="post" action="calc.php"><pre>
Сумма заимствования <input type="text" name="principle">
Ежемесячная выплата <input type="text" name="monthly">
     Количество лет <input type="text" name="years" value="25">
    Процент годовых <input type="text" name="rate"  value="6">
                    <input type="submit">
</pre></form>
```

Если есть желание испробовать в работе этот HTML-код (а также другие подобные примеры), наберите и сохраните его в файле с расширением HTML (или HTM), например в файле test.html (или test.htm), а затем загрузите этот файл в свой браузер.

Обратите внимание на третий и четвертый элементы ввода данных. За счет указания значения для атрибута value в поле отображается значение по умолчанию, которое пользователи в дальнейшем смогут изменить, если у них появится такое желание. Задавая вполне обоснованные значения по умолчанию, можно добиться более дружелюбного поведения от своих веб-форм за счет минимизации необязательного ввода данных. Результат работы предыдущего кода показан на рис. 11.3.

Разумеется, он был создан только для иллюстрации значений по умолчанию, и поскольку программа `calc.php` не была написана, форма после передачи данных никак на это не отреагирует.

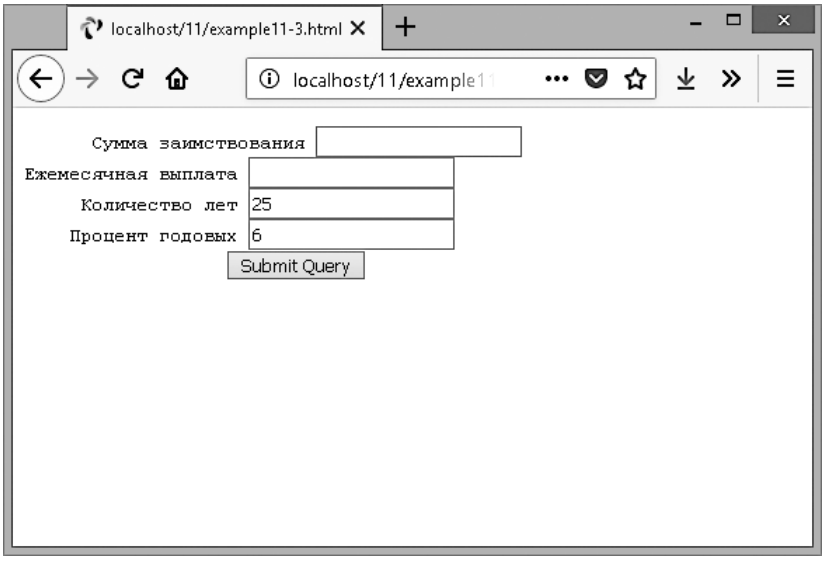

Рис. 11.3. Использование значений по умолчанию для избранных полей формы

Значения по умолчанию используются также для скрытых полей, которые применяются тогда, когда вы хотите наряду с данными, введенными пользователем, отправить из веб-страницы в адрес программы какую-нибудь дополнительную информацию. Скрытые поля будут рассмотрены в этой главе чуть позже.

Типы элементов ввода данных

HTML-формы обладают завидной универсальностью, позволяя отправлять данные из довольно широкого диапазона различных типов элементов ввода, начиная с текстовых полей и текстовых областей и заканчивая флажками, переключателями и т. п.

Текстовое поле

Наверное, самым распространенным типом элемента, применяемым для ввода данных, является текстовое поле. Оно воспринимает широкий диапазон буквенно-цифрового текста и других символов в пределах однострочного окна. Типовой формат текстового поля для ввода информации имеет следующий вид:

```
<input type="text" name="имя" size="размер" maxlength="длина" value="значение">
```

Атрибуты `name` (имя) и `value` (значение) мы уже рассматривали, но здесь представлены еще два атрибута: `size` (размер) и `maxlength` (максимальная длина). Атрибут `size` определяет ширину поля в символах текущего шрифта, каким оно появится на экране, а `maxlength` задает максимальное количество символов, которое пользователю разрешено вводить в это поле.

Единственными обязательными атрибутами являются `type` (тип), сообщающий браузеру ожидаемый тип элемента ввода данных, и `name` (имя), дающий вводимым данным имя, которое используется в дальнейшем для обработки поля после получения отправленной формы.

Текстовая область

Когда нужно принять вводимые данные, превышающие по объему короткую строку текста, используется текстовая область. Она похожа на текстовое поле, но, поскольку в нее разрешается вводить сразу несколько строк, имеет несколько иные атрибуты. Ее типовой формат выглядит следующим образом:

```
<textarea name="имя" cols="ширина" rows="высота" wrap="тип">
</textarea>
```

Первое, на что следует обратить внимание, — использование текстовой областью собственного тега `<textarea>`, который не является подвидом тега `<input>`, поэтому для него нужен закрывающий тег `</textarea>`, чтобы закрыть элемент ввода данных.

Если есть какой-нибудь текст, который нужно отобразить по умолчанию, то вместо использования атрибута, позволяющего задавать подобное значение, нужно поместить этот текст перед закрывающим тегом `</textarea>`, и тогда он будет отображен и сможет редактироваться пользователем:

```
<textarea name="имя" cols="ширина" rows="высота" wrap="тип">
  Это текст, отображаемый по умолчанию.
</textarea>
```

Для управления шириной и высотой текстовой области используются атрибуты `cols` (столбцы) и `rows` (строки). Для задания размеров области в обоих атрибутах в качестве единицы измерения применяются пространства, занимаемые символом текущего шрифта. Если эти значения опустить, то будет создана текстовая область с размерами по умолчанию, которые зависят от используемого браузера, поэтому, чтобы точно знать, в каком виде должна появиться ваша веб-форма, нужно всегда задавать значения этих атрибутов.

И наконец, с помощью атрибута `wrap` (перенос) можно управлять порядком переноса вводимого в область текста (и тем, как этот перенос будет отправляться на сервер). В табл. 11.1 показаны доступные типы переноса. Если не указывать значение атрибута `wrap`, будет задействован мягкий перенос.

Таблица 11.1. Типы переноса, доступные в области ввода <textarea>

Тип	Действие
off	Текст не переносится, и строки появляются в строгом соответствии с тем, как их вводит пользователь
soft	Текст переносится, но отправляется на сервер одной длинной строкой без символов возврата каретки и перевода строки
hard	Текст переносится и отправляется на сервер в формате переноса с «мягким» возвратом в начало следующей строки и переводом строки

Флажки

Если пользователю нужно предложить выбор из нескольких вариантов данных, при котором он может остановиться на одном или нескольких вариантах, то для этого всегда применяются флажки. Формат флажков выглядит следующим образом:

```
<input type="checkbox" name ="имя" value="значение" checked="checked">
```

По умолчанию флажки имеют форму квадрата. Если в этот формат включается атрибут `checked` (установлен), флажок появляется в браузере в уже установленном виде. Строка, присваиваемая атрибуту, должна быть либо парой двойных или одинарных кавычек, либо значением `"checked"`, либо же присваиваемого значения быть не должно (только `checked`). Если данный атрибут не включать в формат, флажок будет отображен в неустановленном виде. Примером задания такого флажка может послужить следующий код:

```
Я согласен <input type="checkbox" name="agree">
```

Если пользователь не установит флажок, значение передано не будет. Но если флажок будет установлен, то для поля по имени `agree` будет передано значение `on`. Если вы предпочитаете вместо `on` отправить собственное значение (например, число 1), можно воспользоваться таким синтаксисом:

```
Я согласен <input type="checkbox" name="agree" value="1">
```

В то же время, если вы хотите при отправке формы предложить своим читателям информационный бюллетень, то может появиться желание отобразить флажок установленным по умолчанию:

```
Подписаться? <input type="checkbox" name="news" checked="checked">
```

Если есть потребность в одновременном выборе группы элементов, то всем им нужно присвоить одинаковые имена. Но при этом нужно иметь в виду, что, если в качестве имени не будет передано имя массива, будет отправлен только последний отмеченный элемент формы. Код из примера 11.4 позволяет пользователю выбрать любимые сорта мороженого (результат работы этого кода в браузере показан на рис. 11.4).

Пример 11.4. Предложение сделать выбор, установив сразу несколько флажков

```
  Ванильное <input type="checkbox" name="ice" value="Vanilla">
 Шоколадное <input type="checkbox" name="ice" value="Chocolate">
Земляничное <input type="checkbox" name="ice" value="Strawberry">
```

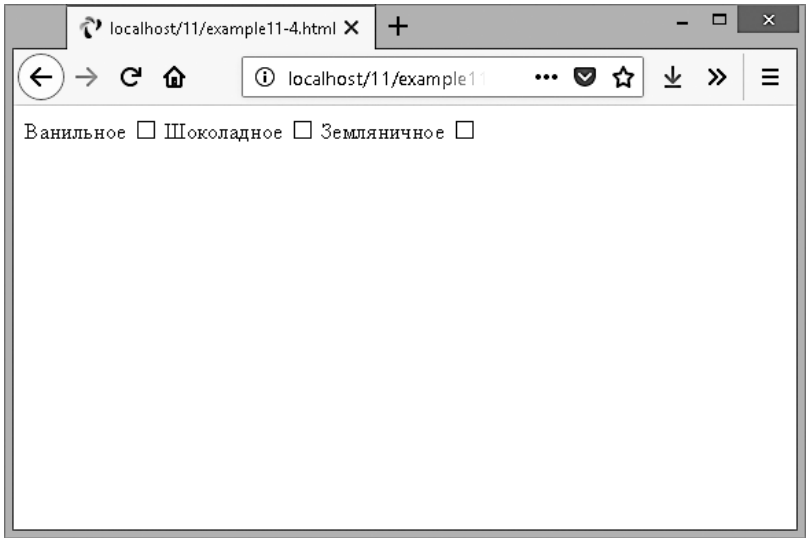

Рис. 11.4. Использование флажков для быстрого выбора

Если установлен только один флажок, например второй, то будет передан лишь этот элемент (полю с именем `ice` будет присвоено значение `Шоколадное`). Но если будут выбраны два и более флажка, будет отправлено только последнее значение, а все предыдущие будут проигнорированы.

Если нужно добиться исключающего поведения, то есть передачи только одного элемента, лучше воспользоваться переключателями (см. следующий подраздел), но, чтобы позволить отправку сразу нескольких значений, необходимо немного изменить код HTML, как показано в примере 11.5 (обратите внимание на добавление квадратных скобок (`[]`) после значений `ice`):

Пример 11.5. Отправка нескольких значений с помощью массива

```
  Ванильное <input type="checkbox" name="ice[]" value="Vanilla">
 Шоколадное <input type="checkbox" name="ice[]" value="Chocolate">
Земляничное <input type="checkbox" name="ice[]" value="Strawberry">
```

Теперь, если при отправке формы установлены какие-нибудь из этих флажков, будет отправлен массив по имени `ice`, содержащий все выбранные значения. В любом случае можно извлечь в переменную либо отдельное значение, либо массив значений:

```
$ice = $_POST['ice'];
```

Если поле `ice` было отправлено в виде отдельного значения, то переменная `$ice` будет содержать отдельную строку, например `Земляничное`. Но если в форме под именем `ice` был определен массив (как в примере 11.5), переменная `$ice` будет массивом и номера элементов этого массива будут номерами отправленных значений.

В табл. 11.2 показаны семь возможных наборов значений, которые могут быть переданы этим HTML-кодом для одного, двух или трех установленных флажков. В каждом из случаев будет создаваться массив из одного, двух или трех элементов.

Таблица 11.2. Семь возможных наборов значений для массива $ice

При отправке одного значения	При отправке двух значений	При отправке трех значений
$ice[0] => Ванильное	$ice[0] => Ванильное $ice[1] => Шоколадное	$ice[0] => Ванильное $ice[1] => Шоколадное $ice[2] => Земляничное
$ice[0] => Шоколадное	$ice[0] => Ванильное $ice[1] => Земляничное	
$ice[0] => Земляничное	$ice[0] => Шоколадное $ice[1] => Земляничное	

Если переменная `$ice` является массивом, то для отображения ее содержимого можно использовать очень простой PHP-код:

```
foreach($ice as $item) echo "$item<br>";
```

В нем применяется стандартная PHP-конструкция `foreach`, осуществляющая последовательный перебор элементов массива `$ice` и передающая значение каждого элемента переменной `$item`, содержимое которой затем отображается с помощью команды `echo`. Тег `<br>` служит только для HTML-форматирования, чтобы после отображения каждого сорта осуществлялся перевод на новую строку.

Переключатели

Переключатели (radio buttons — «кнопки радиоприемника») названы так по аналогии с утапливаемыми кнопками настройки на фиксированные частоты, которые встречались во многих старых радиоприемниках, где любая ранее утопленная кнопка при нажатии другой кнопки возвращалась в первоначальную позицию. Переключатели применяются в тех случаях, когда нужно из двух и более вариантов выбрать и вернуть только один. Все кнопки группы должны использовать одно и то же имя, и, поскольку возвращается только одно значение, массив передавать не требуется.

К примеру, если сайт предлагает выбор времени доставки покупок из вашего магазина, то для этого можно воспользоваться HTML-кодом, показанным в примере 11.6 (результат его работы изображен на рис. 11.5). По умолчанию переключатели имеют форму окружностей.

Пример 11.6. Использование переключателей

```
  8.0-12.00<input type="radio" name="time" value="1">
12.00-16.00<input type="radio" name="time" value="2" checked="checked">
16.00-20.00<input type="radio" name="time" value="3">
```

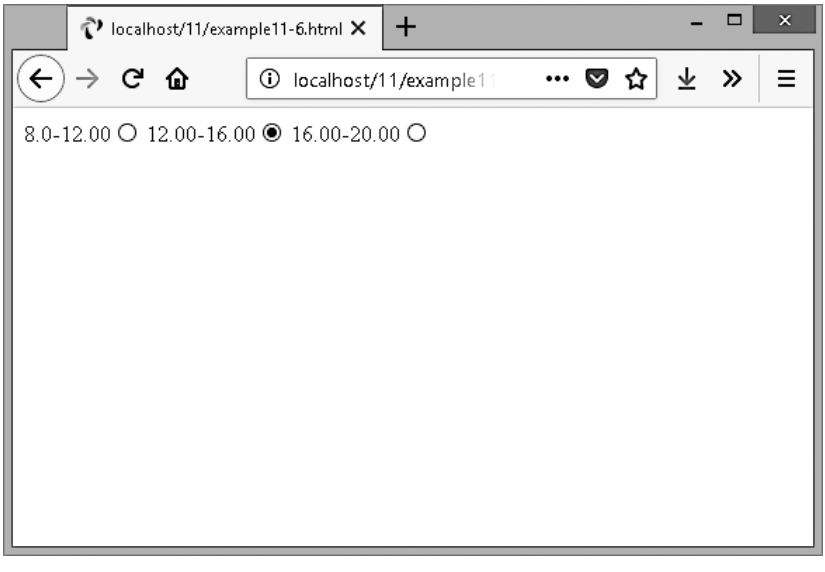

Рис. 11.5. Выбор единственного значения с помощью переключателей

Здесь по умолчанию задается второй вариант: 12.00–16.00. Наличие значения по умолчанию гарантирует, что пользователь выберет хотя бы одно время доставки, которое затем может быть изменено на любое из двух оставшихся в соответствии с их предпочтениями. Если бы не был заранее выбран один из этих вариантов, пользователь мог бы забыть сделать свой выбор и для времени доставки не было бы передано никакого значения.

Скрытые поля

Иногда бывает удобно пользоваться скрытыми полями формы, чтобы получить возможность отслеживать состояние ее ввода. Например, может потребоваться узнать, отправлена форма или нет. Эти сведения можно получить, добавив к PHP-коду фрагмент кода HTML:

```
echo '<input type="hidden" name="submitted" value="yes">'
```

Это простая PHP-инструкция `echo`, добавленная к полю ввода HTML-формы. Предположим, форма была создана вне программы и показана пользователю. При первом получении PHP-программой введенных данных эта строка кода не будет запущена, поэтому поля с именем `submitted` не будет. Программа PHP воссоздает форму, добавляя к ней поле ввода.

Поэтому, когда пользователь отправит форму еще раз, PHP-программа получит ее с полем `submitted`, имеющим значение `yes`. Существование этого поля можно легко проверить с помощью следующего кода:

```
if (isset($_POST['submitted']))
{...
```

Скрытые поля могут пригодиться также для хранения других сведений, например идентификационной строки сеанса, которая может быть создана для идентификации пользователя, и т. д.

Скрытые поля нельзя считать безопасными, поскольку они этим свойством не обладают. Код HTML, которым задаются эти поля, может быть легко просмотрен с помощью свойства браузера, позволяющего просматривать исходный код страницы. Злоумышленник может также создать сообщение, которое удаляет, добавляет или изменяет скрытое поле.

<select>

Тег `<select>` позволяет создавать раскрывающийся список, предлагающий выбор одного или нескольких значений. Для его создания используется следующий синтаксис:

```
<select name="имя" size="размер" multiple="multiple">
```

Атрибутом `size` (размер) задается количество отображаемых строк. Нажатие кнопки отображения приводит к раскрытию списка, показывающего все варианты. Если применяется атрибут `multiple` (множественный выбор), из списка путем удерживания во время нажатия клавиши Ctrl могут быть выбраны сразу несколько вариантов. Чтобы спросить у пользователя, какой из пяти овощей он любит больше всего, можно воспользоваться кодом, предлагающим единичный выбор (пример 11.7).

Пример 11.7. Использование поля со списком

```
Овощи
<select name="veg" size="1">
  <option value="Горох">Горох</option>
  <option value="Фасоль">Фасоль</option>
  <option value="Морковь">Морковь</option>
  <option value="Капуста">Капуста</option>
  <option value="Брокколи">Брокколи</option>
</select>
```

Этот HTML-код предлагает пять вариантов, на первый из которых, Горох, выпадает предварительный выбор (благодаря тому что он стоит первым в списке). На рис. 11.6 показан внешний вид раскрытого щелчком списка. Если требуется выбрать другой вариант, предлагаемый по умолчанию (например, Фасоль), нужно воспользоваться атрибутом `selected`:

```
<option selected="selected" value="Фасоль">Фасоль</option>
```

Можно также дать пользователям возможность выбрать несколько вариантов (пример 11.8).

Пример 11.8. Использование тега <select> с атрибутом multiple

```
Овощи
<select name="veg" size="5" multiple="multiple">
  <option value="Горох">Горох</option>
  <option value="Фасоль">Фасоль</option>
  <option value="Морковь">Морковь</option>
  <option value="Капуста">Капуста</option>
  <option value="Брокколи">Брокколи</option>
</select>
```

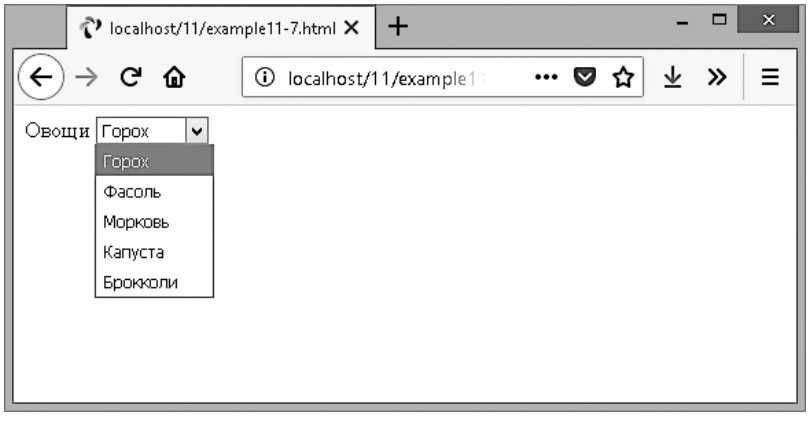

Рис. 11.6. Создание раскрывающегося списка с помощью тега <select>

Код HTML не претерпел при этом значительных изменений, было лишь изменено значение атрибута `size` на `5` и добавлен атрибут `multiple`. Но теперь, судя по рис. 11.7, можно выбрать более одного варианта, удерживая при щелчке нажатой клавишу Ctrl. При желании можете отказаться от атрибута `size`, внешний вид от этого не изменится, но с более длинным списком может отображаться больше элементов, поэтому я рекомендую вам подобрать подходящее количество строк и придерживаться своего выбора. Я также советую не делать поля со списком, работающие в режиме множественного выбора, меньше двух строк в высоту, поскольку некоторые браузеры могут при этом некорректно отображать полосы прокрутки, необходимые для доступа к данным.

Рис. 11.7. Использование поля со списком с атрибутом multiple

При определении поля со списком, работающего в режиме множественного выбора, можно также воспользоваться атрибутом `selected` для задания при необходимости более одного заранее выбранного варианта.

Теги <label>

За счет использования тегов `<label>` можно сделать работу пользователя еще удобнее. В эти теги можно заключить элемент формы, обеспечивая его выбор щелчком на любой видимой части, содержащейся между открывающим и закрывающим тегами `<label>`.

Возвращаясь к примеру выбора времени доставки, можно позволить пользователю щелкать как на самом переключателе, так и на связанном с ним тексте:

```
<label>8.00-12.00<input type="radio" name="time" value="1"></label>
```

При этом текст не будет подчеркиваться при прохождении над ним указателя мыши, как это происходит с гиперссылкой, но указатель мыши из текстового курсора будет превращаться в стрелку, показывая, что щелкать можно на всем тексте.

Кнопка отправки

Чтобы согласовать текст на кнопке отправки с разновидностью отправляемой формы, его можно изменить по своему усмотрению, воспользовавшись атрибутом `value`:

```
<input type="submit" value="поиск">
```

Можно также заменить стандартный текст на кнопке выбранным вами графическим изображением, используя следующий код HTML:

```
<input type="image" name="submit" src="image.gif">
```

Обезвреживание введенных данных

Вернемся к программированию на PHP. Нелишне будет еще раз напомнить, что обработка данных, введенных пользователем, представляет большую угрозу для безопасности системы, и поэтому очень важно научиться с самого начала работать с ними предельно осторожно. На самом деле очистить введенные пользователем данные от потенциальных попыток взлома не так уж сложно, но сделать это совершенно необходимо.

Прежде всего нужно запомнить, что, невзирая на те ограничения, которые были наложены на HTML-форму в отношении типов элементов и размеров вводимых данных, взломщику ничего не стоит воспользоваться свойством браузера, позволяющим просмотреть исходный код страницы, извлечь форму и внести в нее изменения для отправки на ваш сайт вредоносного кода под видом введенных данных.

Поэтому не следует доверять какой-либо переменной, извлеченной из массива `$_GET` или `$_POST`, до тех пор, пока она не пройдет обезвреживание. Если такую обработку не провести, пользователи могут предпринять попытку внедрить в данные код JavaScript, мешающий работе ваших сайтов, или даже добавить команды MySQL, подвергающие угрозе содержимое вашей базы данных.

Таким образом, нужно не только считывать введенные пользователем данные с помощью следующего кода:

```
$variable = $_POST['user_input'];
```

но и воспользоваться еще одной или несколькими строками кода. К примеру, чтобы предотвратить внедрение escape-символов в строку, которая будет представлена MySQL, можно применить код, приведенный ниже. Следует напомнить, что эта функция учитывает текущий набор символов, используемый при подключении к MySQL, поэтому она должна быть задана с объектом подключения `mysqli` (в данном случае `$connection`) в соответствии с порядком, рассмотренным в главе 10:

```
$variable = $connection->real_escape_string($variable);
```

Следует помнить, что самым безопасным способом уберечь MySQL от попыток взлома является рассмотренное в главе 10 использование указателей мест заполнения и предварительно определенных запросов. Если этот способ применять для всех обращений к MySQL, отпадет необходимость в обезвреживании данных, переносимых в базу данных или из этой базы. Но пользовательский ввод перед включением его в HTML обезвреживать все же придется.

Чтобы избавиться от нежелательных слеш-символов, нужно сначала проверить в PHP включение свойства «волшебные кавычки» (при котором кавычки выклю-

чаются путем добавления слеш-символов) и, если оно включено, вызвать функцию `stripslashes`:

```
if (get_magic_quotes_gpc())
  $variable = stripslashes($variable);
```

А для удаления из строки любого HTML-кода используется такой код PHP:

```
$variable = htmlentities ($variable);
```

Например, этот код интерпретируемого HTML `<b>hi</b>` заменяется строкой `<b>hi</b>`, которая отображается как простой текст и не будет интерпретироваться как теги HTML.

И наконец, если нужно полностью очистить введенные данные от HTML, используется следующий код (но указывать его нужно до вызова функции `htmlentities`, которая заменяет все угловые скобки, используемые в качестве составляющих HTML-тегов):

```
$variable = strip_tags($variable);
```

А пока вы не решите, какое именно обезвреживание требуется для вашей программы, рассмотрите показанные в примере 11.9 две функции, в которых собраны вместе все эти ограничения, обеспечивающие довольно высокий уровень безопасности.

Пример 11.9. Функции sanitizeString и sanitizeMySQL

```
<?php
  function sanitizeString($var)
  {
    if (get_magic_quotes_gpc())
      $var = stripslashes($var);
    $var = strip_tags($var);
    $var = htmlentities($var);
    return $var;
  }

  function sanitizeMySQL($connection, $var)
  {
    $var = $connection->real_escape_string($var);
    $var = sanitizeString($var);
    return $var;
  }
?>
```

Добавьте этот код в последние строки своих программ, и тогда вы сможете вызвать его для обезвреживания всех вводимых пользователями данных:

```
$var = sanitizeString($_POST['user_input']);
```

или, если имеется открытое подключение к MySQL и объект подключения `mysqli` (который в данном случае называется `$connection`):

```
$var = sanitizeMySQL($connection, $_POST['user_input']);
```

Если используется процедурная версия расширения mysqli, нужно будет изменить функцию sanitizeMySQL для вызова функции mysqli_real_es cape_string, получив примерно такой код (в этом случае $connection будет не объектом, а описателем):

```
$var = mysqli_real_escape_string($connection, $var);
```

Пример программы

Рассмотрим, как происходит настоящее объединение PHP-программы с HTML-формой, для чего создадим программу `convert.php`, код которой показан в примере 11.10. Наберите этот код и проверьте его работу.

Пример 11.10. Программа перевода значений между шкалами Фаренгейта и Цельсия

```
<?php // convert.php
  $f = $c = '';

  if (isset($_POST['f'])) $f = sanitizeString($_POST['f']);
  if (isset($_POST['c'])) $c = sanitizeString($_POST['c']);

  if (is_numeric($f))
  {
    $c = intval((5 / 9) * ($f - 32));
    $out = "$f &deg;f соответствует $c &deg;c";
  }
  elseif(is_numeric($c))
  {
    $f = intval((9 / 5) * $c + 32);
    $out = "$c &deg;c equals $f &deg;f";
  }
  else $out = "";

  echo <<<_END
<html>
  <head>
    <title>Программа перевода температуры</title>
  </head>
  <body>
    <pre>
      Введите температуру по Фаренгейту или по Цельсию
      и нажмите кнопку Перевести

      <b>$out</b>
      <form method="post" action=" ">
        По Фаренгейту <input type="text" name="f" size="7">
            По Цельсию <input type="text" name="c" size="7">
                       <input type="submit" value="Перевести">
      </form>
    </pre>
  </body>
</html>
```

```
_END;

  function sanitizeString($var)
  {
    if (get_magic_quotes_gpc())
      $var = stripslashes($var);
    $var = strip_tags($var);
    $var = htmlentities($var);
    return $var;
  }
?>
```

Когда программа `convert.php` будет вызвана в браузере, результат будет похож на копию экрана, показанную на рис. 11.8.

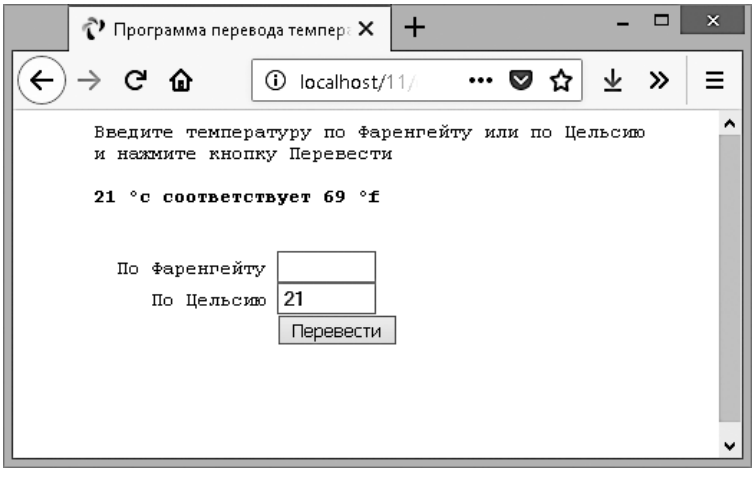

Рис. 11.8. Работающая программа перевода температуры

Проанализируем эту программу. В первой строке инициализируются переменные `$c` `$f` на тот случай, если их значения не были отправлены программе. В следующих двух строках извлекаются значения либо из поля `f`, либо из поля `c`. Эти поля предназначены для ввода значений температуры по Фаренгейту или по Цельсию. Если пользователь введет оба значения, то значение по Цельсию будет проигнорировано, а переведено будет значение по Фаренгейту. В качестве меры безопасности в программе также используется новая функция `sanitizeString` из примера 11.9.

Итак, располагая либо отправленными значениями, либо пустыми строками в обеих переменных `$f` и `$c`, следующая часть кода использует структуру `if...elseif...else`, которая сначала проверяет, имеет ли числовое значение переменная `$f`. Если эта переменная не имеет числового значения, проверяется переменная `$c`. Если переменная `$c` также не имеет числового значения, переменной `$out` присваивается пустая строка (к этому месту мы еще вернемся).

Если обнаружится, что у переменной `$f` есть числовое значение, переменной `$c` будет присвоено простое математическое выражение, которое переводит значение переменной `$f` со значения по Фаренгейту в значение по Цельсию. Для этого используется формула: По_Цельсию = (5 / 9) × (По_Фаренгейту – 32). Затем переменной `$out` присваивается строковое значение, в котором содержится сообщение о результатах перевода.

Если же окажется, что у переменной `$c` есть числовое значение, выполнится обратная операция по переводу значения `$c` из значения по Цельсию в значение по Фаренгейту с присваиванием результата переменной `$f`. При этом используется следующая формула: По_Фаренгейту = (9 / 5) × По_Цельсию + 32. Как и в предыдущем разделе, переменной `$out` затем присваивается строковое значение, в котором содержится сообщение о результатах перевода.

Для превращения результатов перевода в целое число в обоих переводах вызывается PHP-функция `intval`. В этом нет особой необходимости, но результат выглядит лучше.

Теперь, после выполнения всех арифметических вычислений, программа выдает HTML-код, который начинается с базовых элементов `head` и `title` и содержит вводный текст, предшествующий отображению значения переменной `$out`. Если перевода температуры не осуществлялось, переменная `$out` будет иметь значение `NULL` и выводиться на экран ничего не будет, что, собственно, нам и нужно до тех пор, пока не будут отправлены данные формы. Но, если перевод состоялся, переменная `$out` содержит результат, который отображается на экране.

Затем следует форма, настроенная на отправку данных самой программе (представленной парой двойных кавычек, чтобы файл мог быть сохранен с любым именем) с использованием метода POST. Внутри формы содержатся два поля для ввода температуры как по Фаренгейту, так и по Цельсию. Затем отображается кнопка отправки данных, имеющая надпись Перевести, и форма закрывается.

После вывода HTML-кода, закрывающего документ, программа завершается функцией `sanitizeString` из примера 11.9. Проверьте пример в работе, вводя в поля различные значения. А сможете ли вы подобрать значение, для которого температура как по Фаренгейту, так и по Цельсию будет одинакова?

Все примеры, показанные в данной главе, используют для отправки данных формы метод POST. Я рекомендую применять именно его как наиболее подходящий и безопасный. Разумеется, формы можно легко перестроить под использование метода GET, тогда значения нужно будет извлекать не из массива $_POST, а из массива $_GET. Причины применения другого метода могут заключаться в предоставлении возможности создания закладок или непосредственных ссылок с другой страницы на результаты поиска.

Усовершенствования, появившиеся в HTML5

Благодаря HTML5 разработчики могут воспользоваться рядом полезных усовершенствований по обработке форм, упрощающих работу как никогда ранее. В языке разметки появились новые атрибуты, окна выбора цвета, даты и времени, новые типы вводимых данных, хотя некоторые из этих свойств пока реализованы не во всех основных браузерах.

Атрибут autocomplete

Атрибут `autocomplete` можно применить либо к элементу `<form>`, либо к любому из типов элемента `<input>`: `color`, `date`, `email`, `password`, `range`, `search`, `tel`, `text` или `url`.

При включении атрибута `autocomplete` заново вызываются ранее введенные пользователем данные, которые автоматически вводятся в поля в качестве предложений. Это свойство также можно отключить путем переключения `autocomplete` на `off`.

В следующем коде показано, как включить `autocomplete` для всей формы, но отключить этот атрибут для конкретных полей (выделено полужирным шрифтом):

```
<form action='myform.php' method='post' autocomplete='on'>
  <input type='text'     name='username'>
  <input type='password' name='password' autocomplete='off'>
</form>
```

Атрибут autofocus

Атрибут `autofocus` приводит к моментальной установке фокуса на элемент при загрузке страницы. Может быть применен к любому элементу `<input>`, `<textarea>` или `<button>`, например:

```
<input type='text' name='query' autofocus='autofocus'>
```

Браузеры, использующие интерфейсы сенсорных экранов (Android, iOS или Windows Phone), обычно игнорируют атрибут autofocus, оставляя за пользователем право прикоснуться к изображению элемента, чтобы он получил фокус. Если бы это было не так, то генерируемые включением этого атрибута увеличения элемента, фокусировки и появления экранной клавиатуры очень скоро стали бы сильно раздражать пользователей.

Поскольку это свойство вызывает перемещение фокуса на элемент ввода данных, нажатие клавиши Backspace больше не позволяет пользователю вернуться на ранее просмотренную веб-страницу (хотя сочетания Alt+← и Alt+→ по-прежнему можно применять для переходов по истории просмотров назад и вперед).

Атрибут placeholder

Атрибут `placeholder` позволяет помещать в пустое поле ввода полезную подсказку, объясняющую пользователям, что именно им нужно ввести. Он применяется следующим образом:

```
<input type='text' name='name' size='50' placeholder='Имя и фамилия'>
```

В поле ввода текст заполнителя будет показан в качестве подсказки до тех пор, пока пользователь не начнет набирать текст. В этот момент заполнитель исчезнет.

Атрибут required

Атрибут `required` предназначен для обеспечения обязательного заполнения поля перед отправкой формы:

```
<input type='text' name='creditcard' required='required'>
```

Когда браузер обнаружит попытку отправки формы с незаполненным обязательным вводом, пользователю будет выведено приглашение на заполнение поля.

Атрибуты подмены

С помощью этих атрибутов можно подменить настройки формы на поэлементной основе. К примеру, используя атрибут `formaction`, можно указать, что при нажатии кнопки отправки данные формы будут отправлены по URL-адресу, отличающемуся от того адреса, который указан в самой форме (исходный URL-адрес действия и тот адрес, которым он подменяется, показаны полужирным шрифтом):

```
<form action='url1.php' method='post'>
  <input type='text' name='field'>
  <input type='submit' formaction='url2.php'>
</form>
```

В HTML5 также появилась поддержка для атрибутов подмены `formenctype`, `formmethod`, `formnovalidate` и `formtarget`, которые могут использоваться точно так же, как и атрибут `formaction` для подмены одной из соответствующих их именам настроек.

Атрибуты width и height

Используя эти новые атрибуты, можно изменить размеры вводимого изображения:

```
<input type='image' src='picture.png' width='120' height='80'>
```

Атрибуты min и max

Используя атрибуты `min` и `max`, можно указать для полей ввода минимальное и максимальное значения. Атрибуты применяются следующим образом:

```
<input type='time' name='alarm' value='07:00' min='05:00' max='09:00'>
```

Затем браузер либо предложит для диапазона разрешенных значений селекторы «больше-меньше» («вверх-вниз»), либо просто запретит ввод значений, выходящих за пределы диапазона.

Атрибут step

Атрибут `step` зачастую используется с атрибутами `min` и `max` и поддерживает пошаговый перебор значений, связанных с числами и датами, например:

```
<input type='time' name='meeting' value='12:00'
  min='09:00' max='16:00' step='3600'>
```

При пошаговом переборе значений дат и времени единицей измерения служит одна секунда.

Атрибут form

В HTML5 теги `<input>` уже не нужно помещать в теги `<form>`, поскольку форму, к которой применяется элемент ввода, можно указать, предоставив этому элементу атрибут `form`. В следующем коде показана созданная форма, но с элементом ввода, находящимся за пределами тегов `<form>` и `</form>`:

```
<form action='myscript.php' method='post' id='form1'>
</form>

<input type='text' name='username' form='form1'>
```

Чтобы иметь такую возможность, форме нужно присвоить идентификатор, воспользовавшись атрибутом `id`, и сослаться на этот идентификатор в атрибуте `form` элемента `input`. В наибольшей степени это пригодится для добавления скрытых полей ввода, поскольку вы не в состоянии контролировать, как именно поле расположено внутри формы, или для использования JavaScript с целью изменения форм и полей ввода в процессе обработки.

Атрибут list

В HTML5 поддерживаются прикрепляемые списки для ввода данных, которые упрощают пользователям выбор из предопределенных списков. Их можно применить следующим образом:

```
Выберите нужный сайт:
<input type='url' name='site' list='links'>

<datalist id='links'>
  <option label='Google' value='http://google.com'>
  <option label='Yahoo!' value='http://yahoo.com'>
  <option label='Bing'   value='http://bing.com'>
  <option label='Ask'    value='http://ask.com'>
</datalist>
```

Тип ввода color

Тип ввода `color` вызывает на экран окно выбора цвета, позволяющее выбрать цвет простым щелчком кнопкой мыши. Он используется следующим образом:

```
Выберите цвет <input type='color' name='color'>
```

Типы ввода number и range

Типы ввода `number` и `range` ограничивают ввод числом, а также, как вариант, допустимым диапазоном чисел, например:

```
<input type='number' name='age'>
<input type='range' name='num' min='0' max='100' value='50' step='1'>
```

Окно выбора даты и времени

При выборе типа ввода `date`, `month`, `week`, `time`, `datetime` или `datetimelocal` в поддерживающих это свойство браузерах будет появляться окно выбора, в котором пользователь может сделать свой выбор, как, например, в следующем коде, где вводится время:

```
<input type='time' name='time' value='12:34'>
```

В следующей главе будет показано, как cookie и аутентификация применяются с целью сохранения и загрузки пользовательских предпочтений и как можно управлять всей сессией пользователя.

Вопросы

1. Данные, содержащиеся в форме, могут быть отправлены с использованием одного из двух методов — POST или GET. Какие ассоциативные массивы применяются для передачи этих данных PHP-программе?

2. Чем отличаются друг от друга текстовое поле и текстовая область?
3. Если форма предлагает пользователю три варианта выбора, каждый из которых исключает все оставшиеся (то есть может быть выбран только один из вариантов), то какой тип элемента ввода данных нужно использовать в этом случае, если есть выбор между флажками и переключателями?
4. Как из веб-формы отправить группу значений из поля со списком, используя только одно имя поля?
5. Как отправить данные поля формы, не отображая их на экране браузера?
6. В какой HTML-тег можно поместить элемент формы, чтобы весь его текст или изображение превратились в область выбора этого элемента по щелчку кнопкой мыши?
7. Какая PHP-функция предназначена для преобразования кода HTML в формат, который может быть отображен на экране, но не может интерпретироваться браузером в качестве кода HTML?
8. Какой атрибут формы может быть применен, чтобы помочь пользователям заполнить поля ввода?
9. Как обеспечить обязательное заполнение поля ввода перед отправкой формы?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

12 Cookie, сессии и аутентификация

По мере укрупнения и усложнения вашего проекта будут возрастать и потребности в учете его пользователей. Даже если не предлагается ввод имени пользователя и пароля, то довольно часто возникает необходимость в хранении сведений о ходе текущей сессии пользователя и, возможно, в том, чтобы узнать его при возвращении на ваш сайт.

Подобное взаимодействие с пользователем поддерживается с помощью нескольких технологий: от простых браузерных cookies до обработки сессий и HTTP-аутентификации. В совокупности они позволяют настроить сайт на пользовательские предпочтения, обеспечивая комфортное путешествие по его страницам.

Использование cookie в PHP

Cookie представляет собой фрагмент данных, который веб-сервер с помощью браузера сохраняет на жестком диске вашего компьютера. Этот фрагмент может содержать практически любую буквенно-цифровую информацию (объемом не более 4 Кбайт) и может быть извлечен из вашего компьютера и возвращен на сервер. Чаще всего cookie используется для отслеживания хода сессий, обобщения данных нескольких визитов, хранения содержимого корзины покупателя, хранения сведений, необходимых для входа в систему, и т. д.

В силу своей закрытости cookie может быть считан только из создавшего его домена. Иными словами, если cookie, к примеру, был создан на oreilly.com, он может быть извлечен лишь веб-сервером, использующим этот домен. Это не позволяет другим сайтам получить доступ к сведениям, на владение которыми у них нет разрешения.

Из-за особенностей работы Интернета многие элементы веб-страницы могут быть вставлены из нескольких доменов, каждый из которых может создавать свои собственные cookies. Они называются *сторонними* cookies. Чаще всего они создаются

рекламными компаниями с целью отслеживания пользователей на нескольких сайтах.

Поэтому большинство браузеров позволяют пользователям отключать cookie либо от домена текущего сервера, либо от сторонних серверов, либо и от тех и от других. К счастью, большинство пользователей, отключающих cookie, делают это только в отношении сторонних сайтов.

Обмен cookie осуществляется во время передачи заголовков еще до того, как будет отправлен код HTML веб-страницы. Отправить cookie после передачи HTML-кода уже невозможно. Поэтому четкое планирование использования cookie приобретает особую важность. На рис. 12.1 показан типичный диалог с передачей cookie в форме «запрос — ответ» между браузером и веб-сервером.

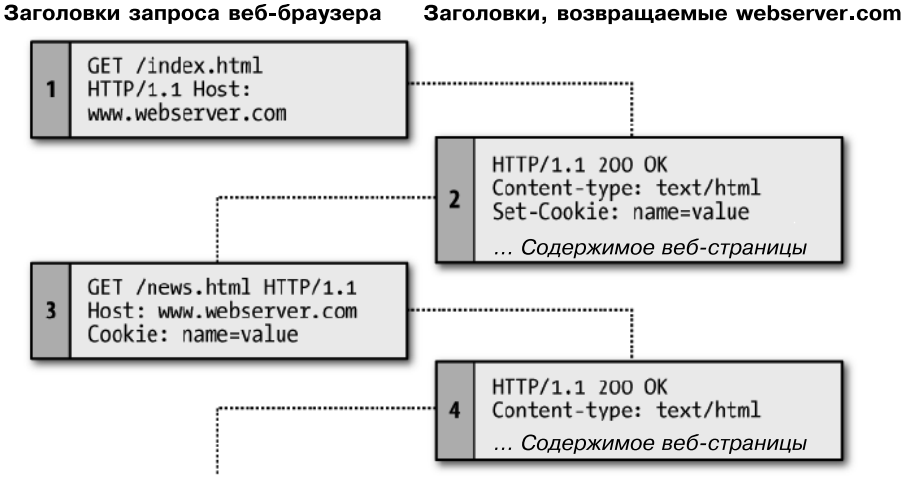

Рис. 12.1. Диалог браузера и сервера в режиме «запрос — ответ» с использованием cookie

В этом обмене данными показан браузер, получающий две страницы.

1. Браузер выдает запрос на извлечение главной страницы `index.html` с сайта http://www.webserver.com. В первом заголовке указывается файл, а во втором — сервер.
2. Когда веб-сервер на webserver.com получает эту пару заголовков, он возвращает несколько своих заголовков. Во втором заголовке определяется тип отправляемого содержимого (`text/html`), а в третьем отправляется cookie по имени `name`, имеющий значение `value`. И только после этого передается содержимое веб-страницы.
3. После того как браузер получит cookie, он должен возвращать его с каждым последующим запросом, сделанным в адрес сервера, создавшего cookie, пока у cookie не истечет срок действия или этот фрагмент данных не будет удален. Поэтому когда браузер запрашивает новую страницу `/news.html`, он также возвращает cookie `name` со значением `value`.

4. Поскольку на момент отправки `/news.html` cookie уже был установлен, сервер не должен заново посылать этот cookie и возвращает только запрошенную страницу.

Cookie-файлы легко редактируются непосредственно из браузера с помощью встроенного инструмента разработчика или специальных расширений. Поэтому, исходя из того, что пользователи могут изменять значения cookie-файлов, в эти файлы не следует помещать такую информацию, как имя пользователя, иначе можно столкнуться с манипулированием вашим веб-сайтом неожиданным для вас образом. Cookie-файлы лучше применять для сохранения данных вроде настроек используемого языка или валюты.

Установка cookie

Установка cookie в PHP осуществляется довольно просто. До передачи кода HTML нужно вызвать функцию `setcookie`, для чего используется следующий синтаксис (табл. 12.1):

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Таблица 12.1. Параметры функции setcookie

Параметр	Описание	Пример
name	Имя cookie. Это имя ваш сервер будет использовать для доступа к cookie при последующих запросах браузера	username
value	Значение cookie или его содержимое. Объем может составлять до 4 Кбайт буквенно-цифрового текста	USA
expire	(Необязательный.) Время истечения срока действия в формате метки времени UNIX. Как правило, для установки этого параметра будет использоваться функция time(), к которой будет прибавляться количество секунд. Если параметр не установлен, срок действия cookie заканчивается с закрытием браузера	time() + 2592000
path	(Необязательный.) Путь к cookie на сервере. Если в качестве пути используется прямой слеш (/), cookie доступен для всего домена, например для домена www.webserver.com. Если указан подкаталог, cookie доступен только в пределах этого подкаталога. По умолчанию путь указывает на текущий каталог, где был установлен cookie, и, как правило, используется именно такая настройка	/
domain	(Необязательный.) Интернет-домен, которому принадлежит cookie. Если это webserver.com, то cookie будет доступен для всего домена webserver.com и его поддоменов, например www.webserver.com и для images.webserver.com. Если это images.webserver.com, то cookie доступен только для images.webserver.com и его поддоменов, например sub.images.webserver.com, но не для www.webserver.com	webserver.com

Параметр	Описание	Пример
secure	(Необязательный.) Определяет, должен ли cookie использовать безопасное подключение (https://). Если значение параметра установлено в TRUE, cookie может быть передан только по безопасному подключению. По умолчанию устанавливается значение FALSE	FALSE
httponly	(Необязательный; реализован в PHP, начиная с версии 5.2.0.) Определяет, должен ли cookie использовать протокол HTTP. Если значение параметра установлено в TRUE, то такие языки сценариев, как JavaScript, не могут получить доступ к cookie. (Это свойство поддерживается не во всех браузерах.) По умолчанию задается значение FALSE	FALSE

Для создания cookie-файла по имени `location` со значением `USA`, к которому имеется доступ со всего веб-сервера текущего домена и который будет удален из браузерного кэша через семь дней, используется следующая строка кода:

```
setcookie('location', 'USA', time() + 60 * 60 * 24 * 7, '/');
```

Доступ к cookie

Для чтения значения cookie нужно просто обратиться к системному массиву `$_COOKIE`. Например, если необходимо посмотреть, хранится ли на текущем браузере cookie по имени `location`, и, если хранится, прочитать его значение, то используется следующая строка кода:

```
if (isset($_COOKIE['location'])) $location = $_COOKIE['location'];
```

Учтите, что прочитать значение cookie можно только после того, как он был отправлен браузеру. Это означает, что при установке cookie его нельзя прочитать до тех пор, пока браузер не перезагрузит страницу (или не совершит какое-нибудь другое действие с доступом к cookie) с вашего сайта и не передаст cookie в ходе этого процесса обратно на сервер.

Удаление cookie

Для удаления cookie его нужно повторно установить с настройкой даты истечения срока действия на прошедшее время. При этом важно, чтобы все параметры нового вызова функции `setcookie`, за исключением `timestamp`, в точности повторяли те параметры, которые указывались при создании cookie, в противном случае удаление не состоится. Поэтому для удаления ранее созданного cookie нужно воспользоваться следующей строкой кода:

```
setcookie('location', 'USA', time() - 2592000, '/');
```

Поскольку указано уже прошедшее время, cookie будет удален. Здесь я использовал время, равное 2 592 000 с в прошлом (что соответствует одному месяцу). Это сделано в расчете на неправильную установку даты и времени на компьютере клиента. Можно также указать для значения cookie пустую строку (или значение FALSE), и тогда PHP автоматически установит для вас свое время на прошлое.

HTTP-аутентификация

HTTP-аутентификация использует веб-сервер для управления пользователями и паролями при работе приложения. Ее можно применять в простых приложениях, требующих от пользователей входа в приложение, хотя для многих приложений нужны особые меры или соблюдение более строгих требований безопасности, для чего следует обратиться к другим технологическим приемам.

Чтобы воспользоваться HTTP-аутентификацией, PHP отправляет заголовок запроса, инициирующий аутентификационный диалог с браузером. Чтобы эта технология заработала, на сервере должно быть включено соответствующее свойство, но, скорее всего, в силу своей высокой востребованности это свойство на вашем сервере уже включено.

Хотя, как правило, аутентификационный модуль HTTP устанавливается вместе с Apache, это еще не означает, что он установлен на используемом вами сервере. Поэтому при попытке запуска представленных здесь примеров может быть сгенерирована ошибка и выдано сообщение о том, что это свойство недоступно. В таком случае нужно либо установить соответствующий модуль и изменить для загрузки модуля файл конфигурации, либо попросить своего системного администратора внести все эти изменения.

Пользователи при вводе в браузер URL-адреса или при переходе по ссылке на страницу видят окно с требованием пройти аутентификацию Требуется аутентификация, в котором выводится приглашение заполнить два поля: для имени пользователя и пароля (на рис. 12.2 показано, как это выглядит в браузере Firefox). Код, обеспечивающий аутентификацию, приведен в примере 12.1.

Пример 12.1. PHP-аутентификация

```
<?php
  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    echo "Пользователь: " . htmlspecialchars($_SERVER['PHP_AUTH_USER']).
         "Пароль:       " . htmlspecialchars($_SERVER['PHP_AUTH_PW']);
  }
  else
  {
```

```
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die("Пожалуйста, введите имя пользователя и пароль");
  }
?>
```

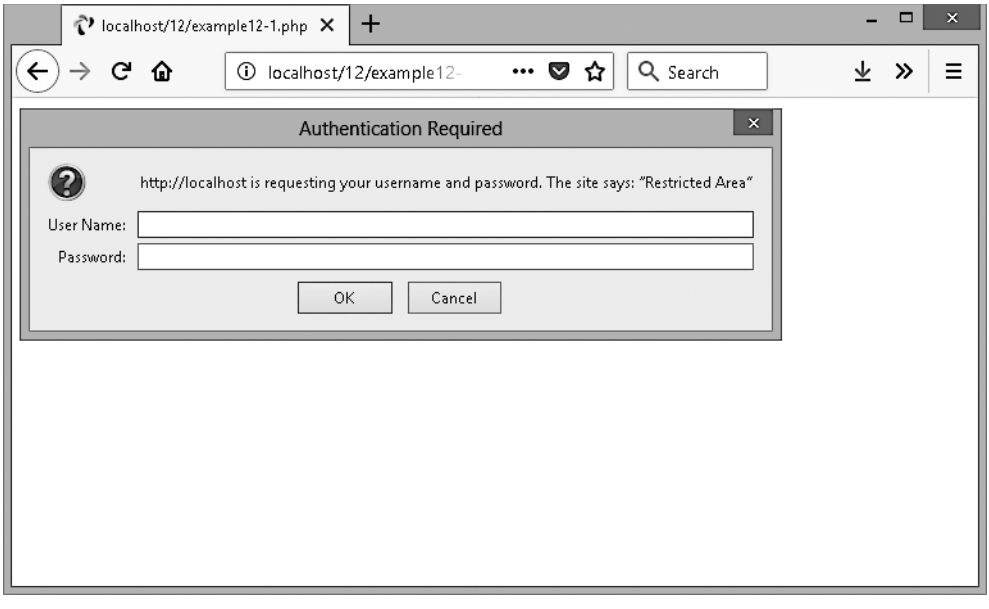

Рис. 12.2. Приглашение войти в систему в режиме HTTP-аутентификации

Сначала программа ищет значения конкретных элементов массива: `$_SERVER['PHP_AUTHJJSER']` и `$_SERVER['PHP_AUTH_PW']`. В этих элементах в случае их существования содержатся имя пользователя и пароль, введенные пользователем при приглашении пройти аутентификацию.

Обратите внимание, что значения, возвращаемые в массиве `$_SERVER`, прежде чем выводиться на экран, обрабатываются функцией `htmlspecialchars`. Дело в том, что эти значения вводятся пользователем, и из-за этого не могут вызывать доверие, поскольку злоумышленником может быть предпринята попытка межсайтового выполнения сценария путем добавления во вводимую информацию символов HTML и каких-либо иных символов. Функция преобразует подобный ввод в безвредное HTML-содержимое.

Если какое-нибудь из значений отсутствует, то пользователь еще не прошел аутентификацию и появляющееся окно с приглашением, показанное на рис. 12.2, отображается с выдачей следующего заголовка, составной частью которого является имя защищенного раздела — Basic realm:

```
WWW-Authenticate: Basic realm="Restricted Section"
```

Если пользователь заполнит поля, PHP-программа будет запущена снова с самого начала. Но если пользователь нажмет кнопку Отмена, программа перейдет к следующим двум строкам, которые отправят такой заголовок и сообщение об ошибке:

```
HTTP/1.0 401 Unauthorized
```

Инструкция `die` выведет следующий текст: Пожалуйста, введите имя пользователя и пароль (рис. 12.3).

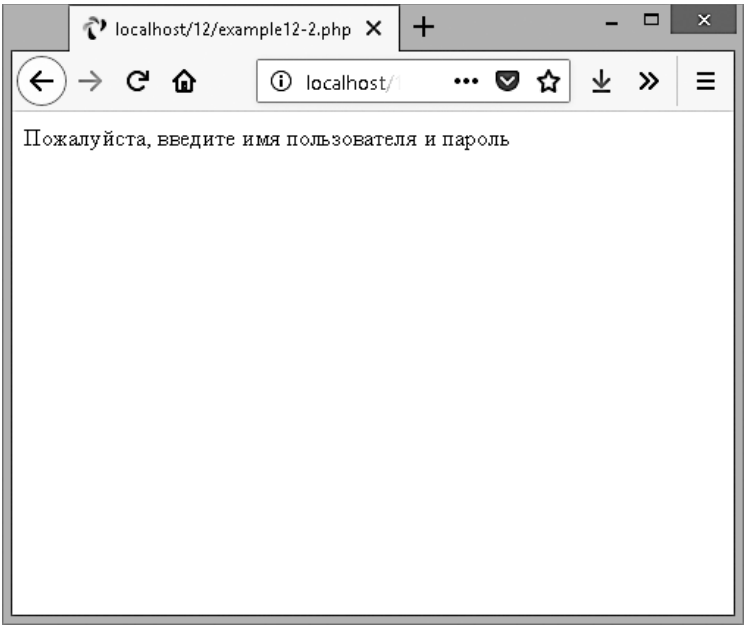

Рис. 12.3. Результат нажатия кнопки Отмена

После прохождения пользователем аутентификации заставить появиться диалоговое окно для аутентификации уже не удастся до тех пор, пока пользователь не закроет и не откроет снова все окна браузера, поскольку браузер будет постоянно возвращать PHP-программе имя пользователя и пароль. Чтобы при изучении этого раздела испытать все возможные режимы работы, может потребоваться несколько раз закрыть и снова открыть ваш браузер. Проще всего добиться желаемого результата, открыв для запуска этих примеров новое отдельное окно, и тогда не придется закрывать весь браузер.

Теперь проверим допустимость пользовательского имени и пароля. Для добавления этой проверки вносить большие изменения в код примера 12.1 не придется: нам нужно лишь изменить прежнее приветственное сообщение и превратить его в проверку правильности имени пользователя и пароля, за которой последует

приветствие. Неудачная аутентификация вызовет отправку сообщения об ошибке (пример 12.2).

Пример 12.2. PHP-аутентификация с проверкой вводимой информации

```
<?php
  $username = 'admin';
  $password = 'letmein';

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    if ($_SERVER['PHP_AUTH_USER'] === $username &&
        $_SERVER['PHP_AUTH_PW']   === $password)
          echo "Регистрация прошла успешно";
    else die("Неверная комбинация имя пользователя – пароль");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
  }
?>
```

При сравнении имен пользователей и паролей используется не оператор равенства (==), а оператор тождественности (===). Дело в том, что нами ведется проверка на *полное соответствие* одного значения другому. Например `'0e123' == '0e456'`, и это неподходящее соответствие ни для имени пользователя, ни для пароля.

Теперь у нас есть механизм аутентификации пользователей, но только для одного имени пользователя и пароля. К тому же пароль появляется в файле PHP в виде простого текста, и если кому-нибудь удастся взломать ваш сервер, он тут же вскроет и пароль. Поэтому рассмотрим более подходящий способ работы с именами пользователей и паролями.

Сохранение имен пользователей и паролей

Безусловно, самым естественным способом сохранения имен пользователей и паролей будет задействование MySQL. Но опять-таки хранить пароли в виде простого текста не хочется, поскольку, если база данных будет взломана, сайт подвергнется опасности. Вместо этого будет использован тонкий прием с применением так называемой *односторонней функции*.

Функции этого типа просты в использовании и способны превращать строку текста в строку, напоминающую набор произвольных символов. Односторонние функции невозможно применять в обратном направлении, поэтому производимая ими выходная информация может безопасно храниться в базе данных и ее похититель ничего не узнает об используемых паролях.

В предыдущем издании этой книги для обеспечения безопасности данных рекомендовалось использовать алгоритм хеширования MD5. Но времена меняются, и теперь считается, что алгоритм MD5 легко поддается взлому, поэтому его применение не обеспечивает должной безопасности. И даже рекомендуемый ранее ему на замену алгоритм SHA-1, по-видимому, также может быть взломан.

Поэтому теперь, когда практически повсеместно минимальным стандартом является применение PHP 5.5, я перешел к использованию встроенной функции хеширования, которая проводит обработку куда более безопасно и аккуратно.

В прежние времена для безопасного сохранения пароля к нему нужно было добавлять произвольные данные, то есть дополнительные символы, которые не вводились пользователем (чтобы его завуалировать еще больше). Затем нужно было пропускать полученный результат через одностороннюю функцию, чтобы превратить его в, казалось бы, случайный набор символов, трудно поддающийся взлому.

Например, выполнение следующего кода (который при нынешнем быстродействии и эффективности современных графических процессоров уже не обеспечивает должного уровня безопасности):

```
echo hash('ripemd128', 'saltstringmypassword');
```

приведет к выводу на экран такого значения:

```
9eb8eb0584f82e5d505489e6928741e7
```

Следует запомнить, что подобный метод не рекомендуется использовать ни в коем случае. Считайте это примером того, как *не нужно* делать, поскольку это небезопасно. И вместо этого, пожалуйста, прочтите следующие разделы.

Функция password_hash

Начиная с версии 5.5, в PHP появился куда более эффективный способ добавления произвольных данных с созданием хешированного пароля: применение функции `password_hash`. Чтобы заставить функцию выбрать наиболее безопасную на данный момент функцию хеширования, следует в качестве ее второго (обязательного) аргумента указать `PASSWORD_DEFAULT`. Функция `password_hash` выберет также для каждого пароля произвольные данные. (Не пытайтесь добавить от себя дополнительные произвольные данные, поскольку это может подорвать безопасность алгоритма.) Следующий код:

```
echo password_hash("mypassword", PASSWORD_DEFAULT);
```

возвратит показанную ниже строку, включающую произвольные данные и всю информацию, необходимую для проверки допустимости пароля:

```
$2y$10$k0YljbC2dmmCq8WKGf8oteBGiXlM9Zx0ss4PEtb5kz22EoIkXBtbG
```

Разрешая PHP выбрать для вас алгоритм хеширования, нужно позволить возвращаемому хешу со временем увеличиваться в размере по мере повышения уровня безопасности. Разработчики PHP рекомендуют сохранять хеши в поле базы данных, способном расширяться как минимум до 255 символов (даже если сейчас средней используемой длиной является 60–72 символа). При желании можно вручную выбрать алгоритм BCRYPT, чтобы гарантированно ограничить длину хеш-строки 60 символами, для чего использовать в качестве второго аргумента функции константу PASSWORD_BCRYPT. Но я не рекомендую делать этого без особо веских причин.

Для дальнейшего определения порядка вычисления хешей можно указать параметры (в виде необязательного третьего аргумента), например затраты или объем процессорного времени, распределяемого на хеширование (чем больше времени, тем выше безопасность, но медленнее сервер). По умолчанию параметр затрат определяется числом 10, которое является минимально возможным при использовании алгоритма BCRYPT.

Чтобы не отпугнуть вас излишней информацией, ограничусь тем минимумом, который необходим для безопасного сохранения хешей паролей без чрезмерных затрат, поэтому за более подробными сведениями о доступных вариантах можно при желании обратиться к соответствующей документации, находящейся по адресу http://php.net/password-hash. Вы можете даже выбрать свои собственные подмешиваемые произвольные данные (хотя начиная с версии PHP 7.0 такой прием не приветствуется из-за недостаточной безопасности, так что не стоит и пытаться).

Функция password_verify

Для проверки соответствия пароля хешу следует воспользоваться функцией `password_verify`, передав ей строку пароля, только что введенную пользователем, и значение сохраненного хеша для пользовательского пароля (как правило, извлекаемое из вашей базы данных).

Итак, при условии, что ваш пользователь ранее ввел весьма примитивный с точки зрения безопасности пароль `mypassword` и у вас уже есть строка хеша этого пароля (с того времени, когда пользователь создал свой пароль), сохраненная в переменной `$hash`, вы можете проверить их взаимное соответствие:

```
if (password_verify("mypassword", $hash))
  echo "Подходящий";
```

Если для хеша предоставлен правильный пароль, функция `password_verify` вернет значение `TRUE` и показанная здесь инструкция `if` выведет на экран слово `Подходящий`. Если пароль не будет соответствовать хешу, будет возвращено значение `FALSE` и вы попросите пользователя повторить попытку.

Пример программы

Посмотрим на совместную работу этих функций в сочетании с MySQL. Сначала нужно создать новую таблицу для хранения хешей паролей, поэтому наберите код программы, показанной в примере 12.3, и сохраните его в файле с именем `setupusers.php` (или загрузите этот файл с сопровождающего книгу сайта), а затем откройте этот файл в своем браузере.

Пример 12.3. Создание таблицы users и добавление к ней двух учетных записей

```
<?php // setupusers.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  $query = "CREATE TABLE users (
    forename VARCHAR(32) NOT NULL,
    surname  VARCHAR(32) NOT NULL,
    username VARCHAR(32) NOT NULL UNIQUE,
    password VARCHAR(32) NOT NULL
  )";

  $result = $connection->query($query);
  if (!$result) die("Fatal Error");

  $forename = 'Bill';
  $surname  = 'Smith';
  $username = 'bsmith';
  $password = 'mysecret';
  $hash     = password_hash($password, PASSWORD_DEFAULT);

  add_user($connection, $forename, $surname, $username, $hash);

  $forename = 'Pauline';
  $surname  = 'Jones';
  $username = 'pjones';
  $password = 'acrobat';
  $hash     = password_hash($password, PASSWORD_DEFAULT);

  add_user($connection, $forename, $surname, $username, $hash);

  function add_user($connection, $fn, $sn, $un, $pw)
  {
    $stmt = $connection->prepare('INSERT INTO users VALUES(?,?,?,?)');
    $stmt->bind_param('ssss', $fn, $sn, $un, $pw);
    $stmt->execute();
    $stmt->close();
?>
```

Программа создаст в вашей базе данных `publications` (или в той базе данных, на которую вы настроились в файле `login.php` в главе 10) таблицу `users`. В этой таблице будут созданы два пользователя — Bill Smith и Pauline Jones с именами пользователей и паролями `bsmith` — `mysecret` и `pjones` — `acrobat` соответственно.

Теперь, используя данные, имеющиеся в этой таблице, мы можем модифицировать код примера 12.2 для вполне приемлемой аутентификации пользователей. Необходимый для этого код показан в примере 12.4. Наберите этот код или загрузите соответствующий файл с сопровождающего книгу сайта, сохраните его в файле `authenticate.php` и вызовите эту программу в своем браузере.

Пример 12.4. PHP-аутентификация с использованием MySQL

```
<?php // authenticate.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    $un_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_PW']);
    $query   = "SELECT * FROM users WHERE username='$un_temp'";
    $result  = $connection->query($query);

    if (!$result) die("User not found");
    elseif ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_NUM);

      $result->close();

      if (password_verify($pw_temp, $row[3])) echo
        htmlspecialchars("$row[0] $row[1] :
        Hi $row[0], you are now logged in as '$row[2]'");
      else die("Неверная комбинация имя пользователя – пароль");
    }
    else die("Неверная комбинация имя пользователя – пароль");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
  }

  $connection->close();

  function mysql_entities_fix_string($connection, $string)
```

```
  {
    return htmlentities(mysql_fix_string($connection, $string));
  }

  function mysql_fix_string($connection, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
  }
?>
```

При HTTP-аутентификации применение функции password_verify с паролями, прошедшими хеширование с помощью алгоритма BCRYPT при используемом по умолчанию значении затрат, равном 10, приведет к замедлению примерно в 80 мс. Это замедление послужит барьером для атакующих, не позволяющим им предпринять попытку взлома пароля на максимальной скорости. Из-за этого замедления HTTP-аутентификация не считается приемлемым решением для слишком нагруженных сайтов, где вы, наверное, отдадите предпочтение использованию сессий (которые рассматриваются в следующем разделе).

Рост объема кода в некоторых приводимых в книге примерах, наверное, вполне соответствует вашим ожиданиям. Но переживать по этому поводу не стоит. Последние 10 строк — не что иное, как пример 10.21, который уже встречался в главе 10. Эти строки присутствуют здесь для выполнения очень важной задачи — обезвреживания введенных пользователем данных.

В настоящий момент практический интерес для вас должны представлять только те строки, которые выделены полужирным шрифтом. Они начинаются с присваивания значений двум переменным — `$un_temp` и `$pw_temp` — с использованием отправленных имени пользователя и пароля. Затем выдается запрос к MySQL на поиск пользователя с именем `$un_temp` и, если будет возвращен результат, значение его первой строки присваивается переменной `$row`. Поскольку имя пользователя уникально, возвращена будет только одна строка.

Теперь остается лишь сравнить значение хеша, хранящееся в базе данных в четвертом столбце, который при начале отсчета с нуля соответствует столбцу 3. В элементе `$row[3]` содержится предыдущее хеш-значение, вычисленное функцией `password_hash` при создании пользователем пароля.

Если хеш и только что предоставленный пользователем пароль проходят проверку, функция `password_verify` возвращает `TRUE`, и выдается строка приветствия, в которой содержится обращение к пользователю по его настоящему имени (рис. 12.4). В противном случае выдается сообщение об ошибке.

Вы можете самостоятельно испытать эту программу в работе, вызвав ее в браузере и набрав имя пользователя `bsmith` и пароль `mysecret` (или набрав пару `pjones`

и `acrobat`), то есть те значения, которые были сохранены в базе данных программой из примера 12.3.

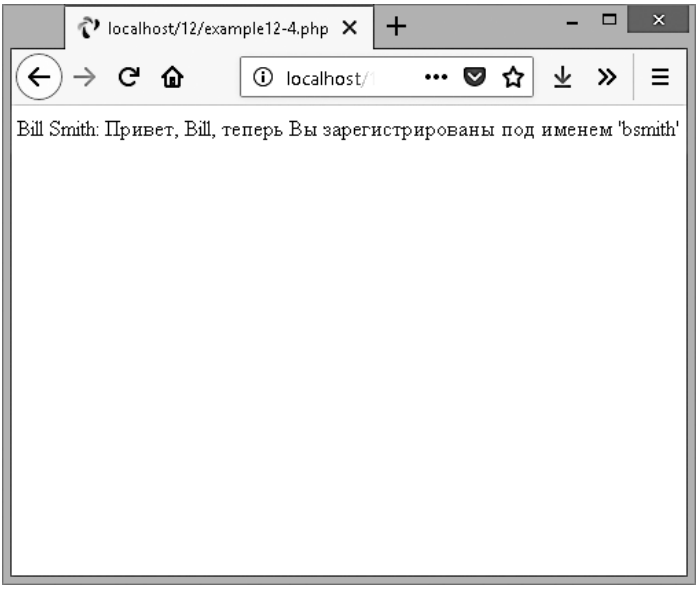

Рис. 12.4. Аутентификация пользователя Bill Smith прошла успешно

Обрабатывая входные данные сразу же после их появления, вы заблокируете любые атаки, которые проводятся путем внедрения вредоносного HTML-, JavaScript- или MySQL-кода, еще до того, как они будут предприняты, и тогда не придется обезвреживать эти данные еще раз. Если пользователь, к примеру, задает в пароле такие символы, как < или &, функция htmlemtities превратит их в последовательности символов < или &. Но пока ваш код будет разрешать получаемым в конечном итоге строкам быть длиннее предоставленной ширины ввода и пока вы неизменно будете пропускать пароли через эту обработку, все будет в порядке.

Использование сессий

Поскольку ваша программа не может сообщить о том, какие значения были присвоены переменным в других программах, или даже о том, какие значения были присвоены переменным при ее предыдущем запуске, иногда приходится отслеживать действия пользователя, переходящего со страницы на страницу. Это можно сделать за счет установки в форме скрытых полей, как было показано в главе 10, и проверки значений этих полей после отправки формы, но PHP предоставляет

более простое, безопасное и действенное решение — *сессии*. Сессии — это группы переменных, которые хранятся на сервере, но относятся только к текущему пользователю. Чтобы обеспечить обращение нужных пользователей к нужным переменным, для уникальной идентификации этих пользователей PHP сохраняет файлы cookie в пользовательских браузерах.

Эти cookie имеют значение только для веб-сервера и не могут быть использованы для извлечения какой-либо информации о пользователе. Вы можете спросить о том, как быть с теми пользователями, которые отключили cookie. В наши дни каждый, кто отключает cookie, не может рассчитывать на получение всей полноты предоставляемой информации, и, если обнаружится, что у пользователя отключены cookie, нужно, наверное, проинформировать его, что при желании в полной мере воспользоваться преимуществами вашего сайта им нужно включить cookie, а не пытаться каким-то образом обойти их применение во избежание возникновения проблем безопасности.

Начало сессии

Чтобы инициировать работу сессии, нужно перед выводом на экран любого кода HTML вызвать PHP-функцию `session_start` точно так же, как это делалось при отправке cookie в процессе обмена заголовками. Затем, чтобы приступить к сохранению переменных сессии, им нужно присвоить значения как элементам массива `$_SESSION`:

```
$_SESSION['имя_переменной'] = $переменная_со_значением;
```

При последующих запусках программы их значения можно будет снова прочитать, воспользовавшись таким кодом:

```
$имя_переменной = $_SESSION['имя_переменной'];
```

Предположим, у вас есть приложение, которому всегда нужен доступ к имени и фамилии каждого пользователя в том виде, в котором они сохранены в базе данных в таблице `users`, созданной совсем недавно. Выполним еще одну модификацию программы `authenticate.php` из примера 12.4, чтобы инициировать работу сессии сразу же после идентификации пользователя.

Все необходимые изменения показаны в примере 12.5. Единственное отличие касается раздела `if (password_verify($pw_temp, $row[3]))`, который теперь начинается с открытия сессии и сохранения в ней четырех переменных. Наберите код этой программы (или измените код примера 12.4) и сохраните его в файле `authenticate2.php`. Но пока не запускайте эту программу в браузере, поскольку нужно будет создать еще и вторую программу.

Пример 12.5. Открытие сессии после успешной аутентификации

```
<?php // authenticate2.php
  require_once 'login.php';
```

```
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    $un_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_PW']);
    $query   = "SELECT * FROM users WHERE username='$un_temp'";
    $result  = $connection->query($query);

    if (!$result) die("Пользователь не найден");
    elseif ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_NUM);
      $result->close();
      if (password_verify($pw_temp, $row[3]))
      {
        session_start();
        $_SESSION['forename'] = $row[0];
        $_SESSION['surname']  = $row[1];
        echo htmlspecialchars("$row[0] $row[1] : Привет, $row[0],
        теперь вы зарегистрированы под именем '$row[2]'";
        die ("<p><a href='continue.php'>Щелкните здесь
              для продолжения</a></p>");
      }
      else die("Неверная комбинация имя пользователя – пароль");
    }
    else die("Неверная комбинация имя пользователя – пароль");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
  }
  $connection->close();
  function mysql_entities_fix_string($connection, $string)
  {
    return htmlentities(mysql_fix_string($connection, $string));
  }

  function mysql_fix_string($connection, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
  }
?>
```

К программе также добавлена ссылка Щелкните здесь для продолжения с URL-адресом `continue.php`. Она будет использована для иллюстрации того, как сессия будет перенесена на другую программу или веб-страницу PHP-программы. Поэтому создайте файл `continue.php`, набрав и сохранив в нем программу из примера 12.6.

Пример 12.6. Извлечение переменных сессии

```
<?php // continue.php
  session_start();

  if (isset($_SESSION['forename']))
  {
    $forename = $_SESSION['forename'];
    $surname = $_SESSION['surname'];

    echo "С возвращением, $forename.<br>
    Ваше полное имя $forename $surname.<br>";
  }
  else echo "Пожалуйста, для входа <a href='authenticate2.php'>щелкните
    здесь</a>.";
?>
```

Теперь можно вызвать в браузере `authenticate2.php`, после появления приглашения ввести имя пользователя `bsmith` и пароль `mysecret` (или `pjones` и `acrobat`) и щелкнуть на ссылке для загрузки программы `continue.php`. Когда браузер вызовет эту программу, появится результат, аналогичный показанному на рис. 12.5.

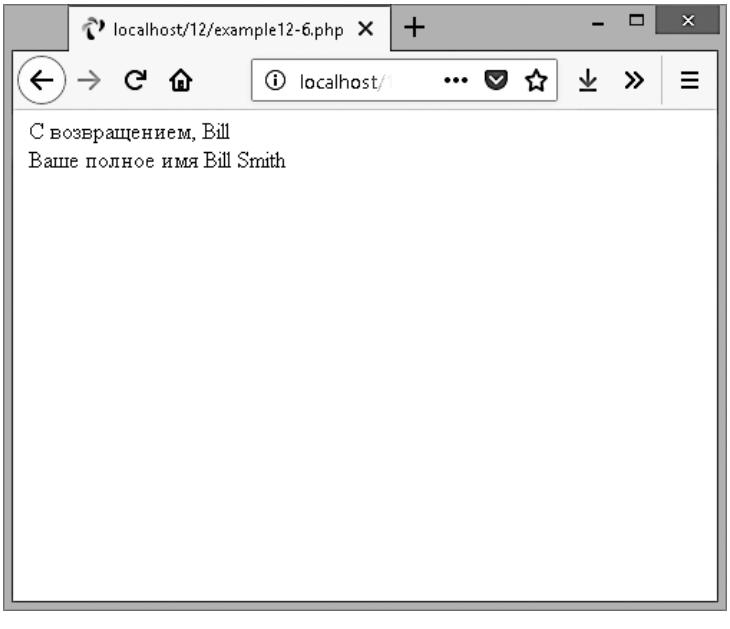

Рис. 12.5. Поддержка пользовательских данных с помощью сессий

Сессии искусно ограничивают одной программой весь объемный код, необходимый для аутентификации и регистрации пользователя. После аутентификации пользователя и создания сессии весь остальной программный код действительно

упрощается. Нужно лишь вызвать функцию `session_start` и найти в массиве `$_SESSION` любые переменные, к которым нужен доступ.

В примере 12.6 быстрой проверки наличия значения у элемента `$_SESSION['forename']` вполне достаточно для того, чтобы узнать об аутентификации текущего пользователя, потому что переменные сессии хранятся на сервере (в отличие от cookie, которые хранятся на машине браузера) и им можно доверять.

Если элементу `$_SESSION['forename']` значение присвоено не было, значит активная сессия отсутствует, и поэтому последняя строка кода в примере 12.6 перенаправляет пользователей на страницу регистрации на сайте, которая находится в программе `authenticate2.php`.

Завершение сессии

Обычно, когда пользователю нужно уйти с вашего сайта, наступает момент завершения работы сессии, для чего, как показано в примере 12.7, можно воспользоваться функцией `session_destroy`. В этом примере предоставляется полезная функция для полного уничтожения сессии, выхода пользователя и очистки всех переменных сессии.

Пример 12.7. Полезная функция уничтожения сессии и ее данных

```
<?php
  function destroy_session_and_data()
  {
    session_start();
    $_SESSION = array();
    setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
  }
?>
```

Для того чтобы увидеть этот код в действии, можно модифицировать программу `continue.php`, как показано в примере 12.8.

Пример 12.8. Извлечение переменных сессии перед ее уничтожением

```
<?php
  session_start();

  if (isset($_SESSION['username']))
  {
    $forename = $_SESSION['forename'];
    $surname = $_SESSION['surname'];

    destroy_session_and_data();

    echo htmlspecialchars("С возвращением, $forename.<br>
```

```
      Ваше полное имя $forename $surname.");

  }
  else echo "Пожалуйста, для входа <a href='authenticate2.php'>щелкните
    здесь.</a> ";

  function destroy_session_and_data()
  {
    $_SESSION = array();
    setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
  }
?>
```

При первом переходе из `authenticate2.php` в `continue.php` будут выведены все переменные сессии. Но если после этого нажать в браузере кнопку обновления страницы, в результате предшествующего этому вызова функции `destroy_session_and_data` сессия уже будет уничтожена и появится приглашение вернуться на страницу регистрации.

Установка времени ожидания

Есть и другие причины, по которым может потребоваться самостоятельное закрытие пользовательской сессии, например, если пользователь забыл зарегистрироваться или проигнорировал этот процесс и нужно, чтобы программа закрыла его сессию ради собственной безопасности. Это можно сделать, установив время ожидания, по истечении которого, если не предпринять активных действий, произойдет автоматическое завершение работы.

Для этого используется функция `ini_set`. В данном примере время ожидания устанавливается ровно на сутки:

```
ini_set('session.gc_maxlifetime', 60 * 60 * 24);
```

Если нужно узнать текущее время ожидания, его можно отобразить, воспользовавшись следующим кодом:

```
echo ini_get('session.gc_maxlifetime');
```

Безопасность сессии

Все мои прежние заверения в том, что после аутентификации пользователя и начала сессии можно спокойно предположить, что переменные сессии заслуживают доверия, не вполне соответствуют действительности. Дело в том, что для вскрытия идентификаторов сессий, передаваемых по сети, можно организовать *анализ пакетов* — packet sniffing (перехват набора данных). Кроме того, если идентификатор (ID) сессии передается в области GET-запроса URL-адреса, он может появиться в файлах регистрации внешних сайтов.

Единственный по-настоящему безопасный способ предотвращения вскрытия заключается в применении *протокола защищенных сокетов* — Secure Socket Layer (SSL) — и запуске веб-страниц, использующих вместо протокола HTTP протокол HTTPS. Эта тема выходит за рамки данной книги, но за подробностями настроек безопасности веб-сервера можно обратиться к документации Apache по адресу http://tinyurl.com/apachetls.

Предупреждение хищения сессии

Когда применение SSL не представляется возможным, можно продолжить аутентификацию пользователей за счет хранения наряду с остальными сведениями их IP-адресов. Для этого нужно при сохранении их сессии добавить следующую строку кода:

```
$_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
```

Затем в качестве дополнительной меры контроля при любой загрузке страницы и доступности сессии проводится следующая проверка, которая при несоответствии текущего IP-адреса сохраненному вызывает функцию `different_user`:

```
if ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR']) different_user();
```

Какой код будет у функции `different_user` — решать вам, но я рекомендую либо удалить текущую сессию и попросить пользователя пройти повторную регистрацию вследствие технической ошибки, либо, если у вас есть электронный адрес пользователя, отправить ему ссылку на подтверждение его личности, что позволит ему сохранить все данные в сессии. Больше ни о чем сообщать не нужно, иначе произойдет утечка потенциально ценной информации.

Разумеется, нужно принимать в расчет, что пользователи, работающие через один и тот же прокси-сервер или использующие одинаковые общие IP-адреса в домашней или офисной сети, будут иметь один и тот же IP-адрес. Если это вызовет проблему, нужно опять обратиться к протоколу HTTPS. Можно также сохранить копию браузерной *строки агента пользователя* (той самой строки, которую разработчики помещают в свои браузеры, для того чтобы идентифицировать их по типу и версии), с помощью которой также возможно отличить пользователей друг от друга благодаря существованию широкого выбора типов, версий и компьютерных платформ (хотя это далеко не идеальное решение, и при автообновлении браузера строка претерпит изменения). Для сохранения агента пользователя можно ввести следующий код:

```
$_SESSION['ua'] = $_SERVER['HTTP_USER_AGENT'];
```

А для сравнения текущей строки агента-пользователя с сохраненной можно воспользоваться таким кодом:

```
if ($_SESSION['ua'] != $_SERVER['HTTP_USER_AGENT']) different_user();
```

Или, что еще лучше, можно объединить эти две проверки и сохранить их комбинацию в виде шестнадцатеричной строки, получаемой от функции hash:

```
$_SESSION['check'] = hash('ripemd128',$_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT']);
```

Затем для сравнения текущей и сохраненной строк можно применить такой код:

```
if ($_SESSION['check'] != hash('ripemd128',$_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT'])) different_user();
```

Предотвращение фиксации сессии

Фиксация сессии возникает, когда сторонний злоумышленник получает верный идентификатор сессии (который может быть сгенерирован сервером) и заставляет пользователя аутентифицироваться с этим идентификатором сессии. Это происходит в том случае, если атакующий злоумышленник пользуется возможностью передачи идентификатора сессии в области GET-запроса URL-адреса, например таким образом:

```
http://yourserver.com/authenticate.php? PHPSESSID=123456789
```

В данном случае серверу будет передан вымышленный ID сессии 123456789. А теперь рассмотрим пример 12.9, код которого восприимчив к фиксации сессии. Чтобы увидеть его в действии, наберите текст примера и сохраните его в файле sessiontest.php.

Пример 12.9. Сессия, восприимчивая к фиксации сессии

```
<?php // sessiontest.php
  session_start();

  if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
  else ++$_SESSION['count'];

  echo $_SESSION['count'];
?>
```

После того как код будет сохранен, вызовите программу в вашем браузере, используя следующий URL (предваряя его правильным путевым именем, например http://localhost):

```
sessiontest.php?PHPSESSID=1234
```

Через некоторое время, нажав кнопку обновления страницы, вы увидите увеличение значения счетчика. Теперь попробуйте ввести в браузер следующий URL-адрес:

```
sessiontest.php?PHPSESSID=5678
```

Несколько раз нажмите кнопку обновления страницы, и вы увидите, что счет опять начался с нуля. Оставьте показания счетчика на номере, отличающемся от его показаний при использовании первого URL-адреса, вернитесь на первый URL-адрес и посмотрите на то, как показания счетчика вернулись к первоначальному значению. Вы по собственному усмотрению создали две разные сессии и без особого труда можете создать их в любом количестве.

Особая опасность этого подхода состоит в том, что затеявший атаку злоумышленник может попытаться распространить такие URL-адреса среди ничего не подозревающих пользователей, и если кто-нибудь из них перейдет по ссылкам с этими адресами, атакующий сможет вернуться и перехватить любую сессию, которая не была удалена или срок действия которой еще не истек!

С целью предотвращения фиксации сессии нужно как можно быстрее воспользоваться для изменения ID сессии функцией `session_regenerate_id`. Она сохраняет значения всех переменных текущей сессии, но заменяет ID сессии новым, о котором не может знать атакующий.

Для этого при получении запроса проверьте факт существования специальной переменной сессии, которую вы выдумали произвольным образом. Если ее не существует, вы будете знать, что создана новая сессия, поэтому вы просто меняете ID сессии и устанавливаете значение ее специальной переменной, позволяющей заметить изменение.

В примере 12.10 показано, как может выглядеть код, использующий переменную сессии `initiated`.

Пример 12.10. Регенерация сессии

```
<?php
  session_start();

  if (!isset($_SESSION['initiated']))
  {
    session_regenerate_id();
    $_SESSION['initiated'] = 1;
  }

  if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
  else ++$_SESSION['count'];

  echo $_SESSION['count'];
?>
```

Атакующий может вернуться на ваш сайт, используя любые сгенерированные им ID-номера сессии, но ни один из них не приведет к вызову другой пользовательской сессии, поскольку все они будут заменены регенерированными ID-номерами.

Преднамеренная настройка на сессии, использующие исключительно cookie

Если вы собираетесь потребовать от своих пользователей включить cookie при просмотре ваших сайтов, то можете обратиться к функции `ini_set`:

```
ini_set('session.use_only_cookies', 1);
```

С такой настройкой трюк с `?PHPSESSID=` будет полностью проигнорирован. Если вы воспользуетесь этой мерой безопасности, я также рекомендую проинформировать ваших пользователей о том, что для работы с вашим сайтом требуются cookie (но только если они отключили cookie), чтобы те знали причину, по которой им не удается получить требуемые результаты.

Использование общего сервера

Если вы делите общий сервер с владельцами других учетных записей, то вам наверняка не захочется, чтобы все данные ваших сессий хранились в том же каталоге, где хранятся данные других пользователей. Вместо этого нужно выбрать для хранения ваших сессий каталог, доступный только пользователю с вашей учетной записью (и поэтому невидимый в сетевом окружении). Для этого придется ближе к началу программы поместить вызов функции `ini_set`:

```
ini_set('session.save_path', '/home/user/myaccount/sessions');
```

В результате такой настройки конфигурации новое значение будет сохраняться лишь на время выполнения программы, а как только она завершит свою работу, будут возвращены исходные настройки.

Папка с данными *сессий* будет быстро заполняться, и в зависимости от занятости вашего сервера может потребоваться ее периодическая очистка от данных устаревших сессий. Чем чаще она будет использоваться, тем реже будет возникать желание хранить данные сессий.

Следует помнить о том, что ваши сайты могут и будут подвергаться попыткам взлома. Существуют так называемые боты (bots), или сетевые автоматические программы, будоражащие Интернет попытками отыскать уязвимые для атак сайты. Поэтому, что бы вы ни делали, если в своей программе вы проводите какую-либо обработку данных, не имеющих стопроцентной гарантии безопасности, к ним нужно относиться с предельной осторожностью.

Теперь, когда вы в достаточной степени усвоили и PHP, и MySQL, настало время предложить вам в следующей главе введение в язык JavaScript — третью основную технологическую составляющую, рассматриваемую в данной книге.

Вопросы

1. Почему cookie должны быть переданы в начале работы программы?
2. Какая PHP-функция сохраняет cookie на машине браузера?
3. Как можно удалить cookie?
4. Где в PHP-программе сохраняются имя пользователя и пароль при использовании HTTP-аутентификации?
5. Почему функция `password_hash` считается мощным средством защиты?
6. Что подразумевается под *«посыпанием солью»* (salting) строки?
7. Что такое PHP-сессия?
8. Как инициируется PHP-сессия?
9. Что такое хищение сессии?
10. Что такое фиксация сессии?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

13 Изучение JavaScript

JavaScript придает вашим сайтам динамическую функциональность. Когда вы видите, как при прохождении указателя мыши над каким-нибудь элементом браузера что-нибудь выделяется, появляется новый текст, изменяется цветовое оформление или изображение, вы должны понимать, что все это делается с помощью JavaScript. Этот язык предлагает такие эффекты, которых нельзя достичь никакими другими средствами, поскольку он запускается внутри браузера и имеет непосредственный доступ ко всем элементам веб-документа.

Впервые JavaScript появился в браузере Netscape Navigator в 1995 году наряду с добавлением поддержки Java-технологии. Поскольку изначально сложилось неверное представление о том, что JavaScript был побочным продуктом Java, возникло устойчивое заблуждение об их взаимосвязанности. Но такое название было всего лишь удачным маркетинговым ходом, призванным помочь новому языку сценариев получить преимущества за счет той популярности, которой пользовался язык программирования Java.

Когда HTML-элементы веб-страницы обрели более четкое, структурированное определение в так называемой *объектной модели документа* — DOM (Document Object Model), язык JavaScript получил еще большие возможности. Объектная модель документа позволила относительно просто добавлять новый абзац или сфокусироваться на какой-нибудь части текста и внести в нее изменения.

Поскольку как в JavaScript, так и в PHP поддерживаются многие элементы синтаксиса структурного программирования, используемые в языке программирования C, эти два языка очень похожи друг на друга. Оба относятся к языкам высокого уровня. К примеру, у них весьма слабая типизация, позволяющая легко приводить переменную к новому типу данных лишь за счет применения ее в новом контексте.

После знакомства с PHP язык JavaScript должен восприниматься еще проще. И его изучение принесет вам несомненное удовольствие, поскольку этот язык является основой технологии асинхронного обмена данными, которая (наряду со свойствами HTML5) предоставляет гибко подстраивающийся пользовательский интерфейс, востребованный в наши дни опытными веб-пользователями.

JavaScript и текст HTML

JavaScript является языком сценариев, который работает исключительно на стороне клиента внутри браузера. Для вызова этого языка код помещается между открывающим и закрывающим HTML-тегами `<script>` и `</script>`. Типовой документ Hello World, созданный на HTML 4.01 с применением JavaScript, может иметь вид, показанный в примере 13.1.

Пример 13.1. Фраза Hello World, отображаемая с помощью JavaScript

```
<html>
  <head><title>Hello World</title></head>
  <body>
    <script type="text/javascript">
      document.write("Hello World")
    </script>
    <noscript>
      Ваш браузер не поддерживает JavaScript, или его поддержка отключена
    </noscript>
  </body>
</html>
```

Вам могут встретиться веб-страницы, в которых используется не рекомендуемый в наши дни HTML-тег <script language="javascript">. В данном примере применяется более современный и предпочтительный тег <script type="text/javascript"> или же, если хотите, просто сам тег <script>.

Внутри тегов `<script>` находится всего одна строка кода JavaScript, в которой используется команда `document.write`, являющаяся эквивалентом PHP-команды `echo` или команды `print`. Как и следовало ожидать, она просто выводит предоставленную ей строку в текущий документ при его отображении на экране.

Можно было также заметить, что, в отличие от PHP, в этой команде отсутствует замыкающая точка с запятой (`;`). Причина в том, что в JavaScript действием, эквивалентным действию точки с запятой, обладает символ новой строки. Тем не менее если потребуется разместить в одной строке более одной инструкции, то после каждой инструкции, кроме последней, нужно ставить точку с запятой. Разумеется, при желании можете ставить точку с запятой после каждой инструкции. Это нисколько не помешает работе JavaScript.

В этом примере следует также обратить внимание на пару тегов `<noscript>` и `</noscript>`. Они используются в том случае, когда требуется предоставить альтернативный HTML пользователям, на чьих браузерах JavaScript не поддерживается или отключен. Эти теги применяются по вашему усмотрению и не являются обязательными, но будет лучше, если вы ими воспользуетесь, поскольку предоставить альтернативу в виде статичного HTML тем операциям, для которых применяется JavaScript, обычно не составляет большого труда. Но в последующих примерах, приводимых в книге, теги `<noscript>` будут опускаться, так как основное

внимание будет уделяться тому, что можно сделать с использованием JavaScript, а не тому, что можно сделать без него.

Когда загрузится код примера 13.1, на экран браузера с включенным JavaScript будет выведен следующий текст (рис. 13.1):

```
Hello World
```

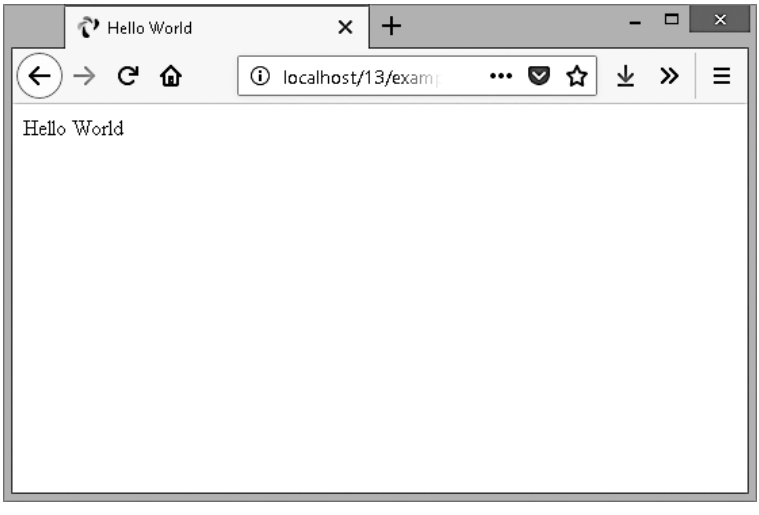

Рис. 13.1. Включенный и работающий JavaScript

А те браузеры, на которых JavaScript отключен, выведут сообщение, как на рис. 13.2.

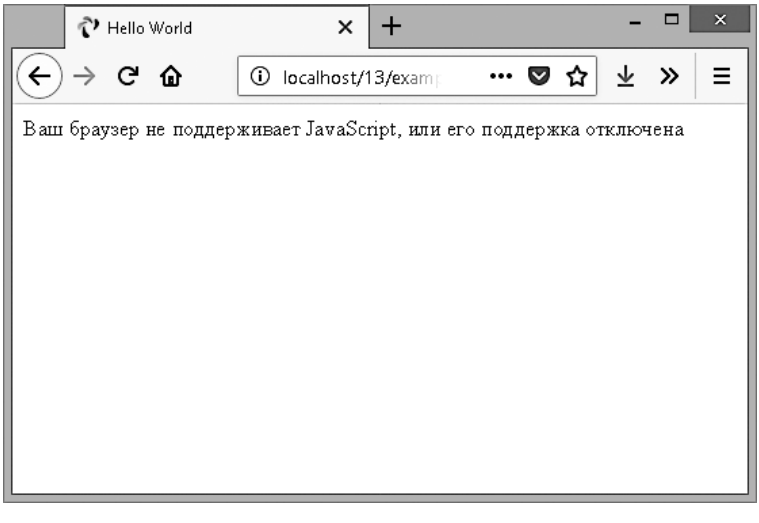

Рис. 13.2. JavaScript отключен

Использование сценариев в заголовке документа

Сценарий можно вставить не только в тело документа, но и в его раздел `<head>`, являющийся идеальным местом, если нужно выполнить сценарий при загрузке страницы. Присутствие в этом разделе какого-нибудь очень важного кода и функций обеспечивает также их немедленную готовность к использованию в других разделах, имеющих сценарии, в том документе, который зависит от их применения.

Другой причиной для вставки сценария в заголовок документа может быть способность JavaScript записывать в раздел `<head>` такие элементы, как метатеги, поскольку то место, куда вставляется сценарий, становится по умолчанию частью документа, куда он осуществляет вывод информации.

Устаревшие и нестандартные браузеры

Если нужно поддерживать браузеры, не допускающие выполнения сценариев (что в наши дни маловероятно), следует воспользоваться HTML-тегами комментариев (`<!--` и `-->`), препятствующими встрече браузеров с кодом сценария, который они не должны видеть. В примере 13.2 показано, как эти теги добавляются к коду сценария.

Пример 13.2. Пример Hello World, измененный в расчете на использование браузеров, не поддерживающих JavaScript

```
<html>
  <head><title>Hello World</title></head>
  <body>
    <script type="text/javascript"><!--
      document.write("Hello World")
    // -->
    </script>
  </body>
</html>
```

В данном примере открывающий HTML-тег комментария (`<!--`) был добавлен сразу же после открывающего тега `<script>`, а тег (`// -->`), закрывающий комментарий, — непосредственно перед тегом `</script>`, который закрывает сценарий.

Двойной прямой слеш (`//`) используется в JavaScript, чтобы показать, что вся остальная строка является комментарием. Он присутствует здесь для того, чтобы браузеры, поддерживающие JavaScript, проигнорировали следующий за ним тег `-->`, а браузеры, не поддерживающие JavaScript, проигнорировали идущие в начале символы `//` и обработали тег `-->`, закрывая тем самым HTML-комментарий.

Хотя такое решение имеет не самый изящный вид, при желании поддерживать устаревшие или нестандартные браузеры вам нужно лишь запомнить используемые в нем две строки, в которые помещается код JavaScript:

```
<script type="text/javascript"><!--
  (Здесь должен быть ваш код JavaScript...)
// -->
</script>
```

Разумеется, пройдет еще несколько лет, и эти комментарии станут не нужны для любых выпускаемых браузеров.

Есть еще два языка сценариев: VBScript, разработанный корпорацией Microsoft и основанный на языке программирования Visual Basic, и Tcl, представляющий собой язык для быстрой разработки прототипов. Их можно вызвать тем же способом, что и JavaScript, за исключением того, что при объявлении типа нужно указывать text/vbscript и text/tcl соответственно. Язык VBScript работает только в Internet Explorer вплоть до версии 10, а в последующих версиях применять его не рекомендуется; его использование в других браузерах требует загрузки дополнительного модуля. Для применения Tcl дополнительный модуль нужен всегда. Поэтому оба этих языка считаются нестандартными и в данной книге не рассматриваются.

Включение файлов JavaScript

В дополнение к внесению кода JavaScript непосредственно в HTML-документы вы можете включать в них файлы с кодом JavaScript или со своего сайта, или из любого места в Интернете. Для этого используется следующий синтаксис:

```
<script type="text/javascript" src="script.js"></script>
```

А для извлечения файла из Интернета применяется этот синтаксис:

```
<script type="text/javascript" src="http://someserver.com/script.js">
</script>
```

В самих файлах сценариев *не должно* быть никаких тегов `<script>` или `</script>`, поскольку они там не нужны: браузеру и так известно, что будет загружаться файл JavaScript. Применение этих тегов в файлах JavaScript приводит к возникновению ошибки.

Включение файлов сценариев — предпочтительный способ использования на вашем сайте файлов JavaScript, принадлежащих сторонним производителям.

Параметры type="text/javascript" можно не указывать, поскольку все современные браузеры по умолчанию предполагают, что сценарий содержит код JavaScript.

Отладка кода JavaScript

При изучении JavaScript очень важно иметь возможность отслеживать набранный код или выявлять не связанные с ним ошибки программирования. В отличие от PHP, который отображает сообщения об ошибках в браузере, JavaScript обрабатывает сообщения об ошибках по-другому, и способ их обработки имеет свои

особенности, зависящие от используемого браузера. В табл. 13.1 перечислены способы доступа к сообщениям об ошибках JavaScript в самых распространенных браузерах.

Таблица 13.1. Доступ к сообщениям об ошибках JavaScript в различных браузерах

Браузер	Способ доступа к сообщениям об ошибках JavaScript
Apple Safari	В Safari нет консоли ошибок, включенной по умолчанию, но вы можете включить эту функцию, выбрав в меню пункты Safari ▸ Настройки ▸ Дополнения и установив флажок Показывать меню «Разработка» в строке меню. Кроме того, предпочтение можно отдать JavaScript-модулю Firebug Lite (https://getfirebug.com/firebuglite), который многие считают более простым в использовании
Google Chrome	Нажмите сочетание клавиш Ctrl+Shift+J на PC или Command+Shift+J на Mac
Microsoft Internet Explorer и Edge	Нажмите клавишу F12, чтобы вызвать консоль инструментария разработчика DevTools Console
Mozilla Firefox	Нажмите сочетание клавиш Ctrl+Shift+J на PC или Command+Shift+J на Mac
Opera	Выберите команду Инструменты ▸ Дополнительно ▸ Консоль ошибок

Все консоли отличаются друг от друга, поэтому по вопросам особенностей их использования обращайтесь, пожалуйста, к документации разработчиков на вебсайтах соответствующих браузеров.

Комментарии

В силу общих наследственных черт, приобретенных у языка программирования C, языки PHP и JavaScript имеют много общего и между собой, в частности в приемах комментирования кода. В первую очередь это касается однострочных комментариев:

```
// Это комментарий
```

В этой технологии используется пара прямых слешей (//), информирующая JavaScript о том, что все остальные символы должны быть проигнорированы. А затем наступает черед многострочных комментариев:

```
/* Это раздел
   многострочного
   комментария,
   не подвергаемого
   интерпретации */
```

Многострочный комментарий начинается с последовательности символов /* и заканчивается символами */. Нужно лишь запомнить, что использовать вложенные многострочные комментарии не допускается, поэтому важно убедиться в отсутствии большого закомментированного участка кода, в котором уже имеются многострочные комментарии.

Точка с запятой

В отличие от PHP, точка с запятой коду JavaScript, имеющему в строке одну инструкцию, не требуется. Поэтому следующая строка вполне имеет право на существование:

```
x += 10
```

Но при необходимости иметь в строке более одной инструкции их нужно разделить точками с запятыми:

```
x += 10; y -= 5; z = 0
```

Последнюю точку с запятой можно опустить, поскольку последняя инструкция будет завершена символом новой строки.

В правилах использования точки с запятой есть исключения. Если пишутся URL-закладки (bookmarklet) JavaScript или инструкция завершается ссылкой на переменную или функцию и первый символ расположенной ниже строки является левой круглой или фигурной скобкой, нужно обязательно поставить точку с запятой, иначе сценарий JavaScript даст сбой. Поэтому при любых сомнениях нужно ставить точку с запятой.

Переменные

В JavaScript нет никаких идентификационных символов переменных, таких как знак доллара ($) в PHP. Вместо этого в отношении имен переменных действуют следующие правила.

- Имена переменных могут включать только буквы a–z, A–Z, цифры 0–9, символ $ и символ подчеркивания (_).
- Никакие другие символы, включая пробелы или знаки пунктуации, использовать в именах переменных не допускается.
- Первым в имени переменной может быть символ из диапазонов a–z, A–Z, символ $ или символ подчеркивания _ (и никаких цифр).

- Имена чувствительны к регистру. Имена `Count`, `count` и `COUNT` принадлежат трем разным переменным.
- Ограничений на длину имени переменной не существует.

Вы наверняка обратили внимание на присутствие в этом перечне символа `$`. JavaScript *допускает* его использование, и он *может быть* первым символом в имени переменной или функции. Такая возможность означает, что благодаря этому можно переносить на JavaScript большие объемы кода PHP значительно быстрее, хотя я не рекомендую применять его в данном качестве.

Строковые переменные

В JavaScript значения строковых переменных должны быть заключены либо в одиночные, либо в двойные кавычки:

```
greeting = "Привет!"
warning  = 'Осторожно!'
```

В строку в двойных кавычках можно включить одиночную кавычку или же в строку в одинарных кавычках можно включить двойную кавычку. Но кавычка того же типа должна быть отключена с помощью символа обратного слеша:

```
greeting = "\"Привет!\" является приветствием"
warning  = '\'Осторожно!\' является предупреждением'
```

Чтобы прочитать значение из строковой переменной, его можно присвоить другой переменной:

```
newstring = oldstring
```

Или использовать его в функции:

```
status = "Все системы работают успешно"
document.write(status)
```

Числовые переменные

Создание числовой переменной сводится к простому присваиванию значения, как в следующих примерах:

```
count       = 42
temperature = 98.4
```

Значения числовых переменных точно так же, как и значения строковых переменных, могут быть прочитаны и использованы в выражениях и функциях.

Массивы

Массивы JavaScript очень похожи на массивы в PHP тем, что они могут содержать строковые или числовые данные, а также другие массивы.

Чтобы присвоить массиву значения, используется следующий синтаксис (с помощью которого в данном случае создается строковый массив):

```
toys = ['bat', 'ball', 'whistle', 'puzzle', 'doll']
```

Для создания многомерного массива более мелкие массивы вкладываются в более крупный. Для создания двумерного массива цветных квадратов, расположенных на одной из сторон кубика Рубика (в котором есть следующие цвета: красный (R), зеленый (G), оранжевый (O), желтый (Y), синий (B) и белый (W) — представленные прописными буквами, указанными в скобках), можно воспользоваться таким кодом:

```
face =
[
    ['R', 'G', 'Y'],
    ['W', 'R', 'O'],
    ['Y', 'W', 'G']
]
```

Предыдущий пример был отформатирован так, чтобы было понятно, что именно происходит, но его можно переписать и в следующем виде:

```
face = [['R', 'G', 'Y'], ['W', 'R', 'O'], ['Y', 'W', 'G']]
```

или даже так:

```
top  = ['R', 'G', 'Y']
mid  = ['W', 'R', 'O']
bot  = ['Y', 'W', 'G']

face = [top, mid, bot]
```

Для доступа к элементу, расположенному в этой матрице во второй сверху и в третьей слева ячейке, нужно воспользоваться следующим кодом (нужно помнить, что позиционирование элементов массива начинается с нуля):

```
document.write(face[1][2])
```

Эта инструкция выведет букву `O`, означающую оранжевый цвет.

Массивы JavaScript — это мощная структура, предназначенная для хранения данных, поэтому в главе 15 они рассматриваются более подробно.

Операторы

Как и в PHP, операторы в JavaScript могут использоваться в математических операциях, для внесения изменений в строки, а также в операциях сравнения и логических операциях (И, ИЛИ и т. д.). Математические операторы JavaScript во многом похожи на обычные арифметические операторы. Например, следующая инструкция выводит число 15:

```
document.write(13 + 2)
```

Все разнообразие операторов будет рассмотрено в следующих разделах.

Арифметические операторы

Арифметические операторы предназначены для осуществления математических операций. Их можно использовать для выполнения четырех основных операций (сложения, вычитания, умножения и деления), а также для нахождения модулей (остатков от деления) и инкремента (увеличения на единицу) или декремента (уменьшения на единицу) значения (табл. 13.2).

Таблица 13.2. Арифметические операторы

Оператор	Описание	Пример
+	Сложение	j + 12
–	Вычитание	j – 22
*	Умножение	j * 7
/	Деление	j / 3.13
%	Деление по модулю (остаток от деления)	j % 6
++	Инкремент	++j
––	Декремент	––j

Операторы присваивания

Эти операторы используются для *присваивания* значений переменным. Их линейка начинается простым знаком равенства (=) и продолжается сочетаниями +=, -= и т. д. Оператор += добавляет значение, находящееся справа, к переменной, находящейся слева, вместо того чтобы целиком заменить значение переменной в левой части. Поэтому, если изначально значение переменной count было 6, то оператор:

```
count += 1
```

установит для нее значение 7 точно так же, как и более привычный оператор присваивания:

```
count = count + 1
```

В табл. 13.3 перечислены различные операторы присваивания, доступные в JavaScript.

Таблица 13.3. Операторы присваивания

Оператор	Пример	Эквивалентен оператору
=	j = 99	j = 99
+=	j += 2	j = j + 2
+=	j += 'string'	j = j + 'string'
–=	j –= 12	j = j – 12
*=	j *= 2	j = j * 2
/=	j /= 6	j = j / 6
%=	j %= 7	j = j % 7

Операторы сравнения

Операторы *сравнения* обычно используются с такими конструкциями, как инструкция `if`, в которой требуется сравнивать два элемента. Например, может потребоваться узнать, достигло ли значение переменной, подвергаемой автоприращению, определенной величины или меньше ли установленной величины значение другой переменной и т. д. (табл. 13.4).

Таблица 13.4. Операторы сравнения

Оператор	Описание	Пример
==	Равно	j == 42
!=	Не равно	j != 17
>	Больше	j > 0
<	Меньше	j < 100
>=	Больше или равно	j >= 23
<=	Меньше или равно	j <= 13
===	Равно (и того же типа)	j === 56
!==	Не равно (и того же типа)	j !== '1'

Логические операторы

У *логических* операторов JavaScript, в отличие от PHP, нет эквивалентов `and` и `or` для `&&` и `||` и отсутствует оператор `xor` (табл. 13.5).

Таблица 13.5. Логические операторы

Оператор	Описание	Пример
&&	И	j == 1 && k == 2
||	ИЛИ	j < 100 || j > 0
!	НЕ	! (j == k)

Инкремент, декремент переменной и краткая форма присваивания

Следующие используемые в PHP и уже изучавшиеся вами формы инкремента (приращения) и декремента (отрицательного приращения), осуществляемые как после операции сравнения, так и перед ней, поддерживаются также и в JavaScript в качестве краткой формы операторов присваивания:

```
++x
--y
x += 22
y -= 3
```

Объединение строк

Объединение (конкатенация) строк в JavaScript осуществляется немного иначе, чем в PHP. Вместо оператора `.` (точка) используется знак «плюс» (`+`):

```
document.write("у вас " + messages + " сообщения.")
```

Если предположить, что переменной `messages` присвоено значение `3`, эта строка кода выведет следующую информацию:

```
У вас 3 сообщения.
```

Оператор `+=` точно так же, как и при добавлении значения к числовой переменной, позволяет добавить одну строку к другой:

```
name = "James" name += " Dean"
```

Управляющие символы

Управляющие символы, пример использования которых вы видели при вставке в строку кавычек, могут применяться также для вставки различных специальных символов: табуляции, новой строки и возврата каретки. В следующем примере символы табуляции используются для разметки заголовка; они включены в строку

лишь для иллюстрации использования управляющих символов, поскольку в вебстраницах существуют более подходящие способы разметки:

```
heading = "Name\tAge\tLocation"
```

В табл. 13.6 приведены управляющие символы, доступные в JavaScript.

Таблица 13.6. Управляющие символы JavaScript

Символ	Назначение
\b	Забой
\f	Перевод страницы
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\'	Одиночная кавычка
\"	Двойная кавычка
\\	Обратный слеш
XXX	Восьмеричное число в диапазоне от 000 до 377, представляющее эквивалент символа Latin-1 (например, \251 для символа ©)
\x*XX*	Шестнадцатеричное число в диапазоне от 00 до FF, представляющее эквивалент символа Latin-1 (например, \xA9 для символа ©)
\u*XXXX*	Шестнадцатеричное число в диапазоне от 0000 до FFFF, представляющее эквивалент символа Unicode (например, \u00A9 для символа ©)

Типизация переменных

Как и PHP, JavaScript относится к весьма слабо типизированным языкам. *Тип* переменной определяется только при присваивании ей значения и может изменяться при появлении переменной в другом контексте. Как правило, о типе переменной волноваться не приходится: язык JavaScript сам определяет, что именно вам нужно, и просто делает это.

Посмотрите на пример 13.3, в котором выполняются следующие действия.

1. Переменной `n` присваивается строковое значение `"838102050"`, в следующей строке осуществляется вывод ее значения, а чтобы посмотреть на ее тип, используется инструкция `typeof`.
2. Переменной `n` присваивается значение, получаемое при перемножении чисел 12 345 и 67 890. Это значение также равно `838102050`, но оно является числом, а не строкой. Затем определяется и выводится на экран тип переменной.
3. К числу `n` добавляется текст, и результат отображается на экране.

Пример 13.3. Установка типа переменной путем присваивания ей значения

```
<script>
  n = '838102050'    // Присваивание 'n' строкового значения
  document.write('n = ' + n + ', и имеет тип ' + typeof n + '<br>')

  n = 12345 * 67890; // Присваивание 'n' числа
  document.write('n = ' + n + ', и имеет тип ' + typeof n + '<br>')

  n += ' плюс текст' // Изменение типа 'n' с числового на строковое
  document.write('n = ' + n + ', и имеет тип ' + typeof n + '<br>')
</script>
```

Этот сценарий выведет следующую информацию:

```
n = 838102050 и имеет тип string
n = 838102050 и имеет тип number
n = 838102050 плюс текст и имеет тип string
```

Если в отношении типа переменной есть какие-то сомнения или нужно обеспечить, чтобы переменная относилась к определенному типу, вы можете принудительно привести ее к этому типу, используя операторы, показанные в следующем примере (которые превращают строку в число и число в строку соответственно):

```
n = "123"
n *= 1    // Превращение 'n' в число

n = 123
n += ""   // Превращение 'n' в строку
```

Или же тип переменной можно всегда определить с помощью инструкции `typeof`.

Функции

Как и в PHP, в JavaScript функции используются для выделения фрагментов кода, выполняющих конкретную задачу. Для создания функции ее нужно объявить, как показано в примере 13.4.

Пример 13.4. Объявление простой функции

```
<script>
  function product(a, b)
  {
    return a*b
  }
</script>
```

Эта функция принимает два переданных ей параметра, перемножает их и возвращает произведение.

Глобальные переменные

К *глобальным* относятся переменные, определенные за пределами любых функций (или внутри функций, но без использования ключевого слова `var`). Они могут быть определены следующими способами:

```
    a = 123                // Глобальная область видимости
var b = 456                // Глобальная область видимости
if (a == 123) var c = 789  // Глобальная область видимости
```

Независимо от применения ключевого слова `var`, если переменная определена за пределами функции, она приобретает глобальную область видимости. Это означает, что к ней может быть получен доступ из любой части сценария.

Локальные переменные

Параметры, переданные функции, автоматически приобретают *локальную* область видимости, то есть к ним можно обращаться только из тела этой функции. Но есть одно исключение. Массивы передаются функции по ссылке, поэтому, если вы внесете изменения в любые элементы массива, переданного в качестве параметра, то элементы исходного массива также будут изменены.

Для определения локальной переменной, имеющей область видимости только внутри текущей функции и не переданной ей в качестве параметра, используется ключевое слово `var`. В примере 13.5 показана функция, которая создает одну переменную с глобальной и две переменные — с локальными областями видимости.

Пример 13.5. Функция, создающая переменные с глобальной и локальной областями видимости

```
<script>
  function test()
  {
        a = 123                // Глобальная область видимости
    var b = 456                // Локальная область видимости
    if (a == 123) var c = 789  // Локальная область видимости
  }
</script>
```

В PHP, чтобы проверить работоспособность установки области видимости, можно воспользоваться функцией `isset`. Но в JavaScript такой функции нет, поэтому в примере 13.6 применяется инструкция `typeof`, возвращающая строку `undefined`, если переменная не определена.

Пример 13.6. Проверка области видимости переменных, определенных в функции test

```
<script>
  test()

  if (typeof a != 'undefined') document.write('a = "' + a + '"<br>')
```

```
  if (typeof b != 'undefined') document.write('b = "' + b + '"<br>')
  if (typeof c != 'undefined') document.write('c = "' + c + '"<br>')

  function test()
  {
    a     = 123
    var b = 456

    if (a == 123) var c = 789
  }
</script>
```

Этот сценарий выведет только одну строку:

```
a = "123"
```

Это свидетельствует о том, что, как и предполагалось, глобальная область видимости была определена лишь для одной переменной, потому что для переменных `b` и `c` установкой перед ними ключевого слова `var` была задана локальная область видимости.

Если ваш браузер выведет предупреждение о том, что переменная `b` не определена, то при всей своей справедливости оно может быть проигнорировано.

Объектная модель документа

Разработчики JavaScript — очень умные люди. Вместо того чтобы просто создать еще один язык написания сценариев (который имел бы на момент создания весьма неплохие усовершенствования), они проявили дальновидность и построили его вокруг уже существующей объектной модели документа HTML, или DOM (Document Object Model). Эта модель разбивает части HTML-документа на отдельные *объекты*, у каждого из которых есть собственные *свойства* и *методы* и каждым из которых можно управлять с помощью JavaScript.

В JavaScript объекты, свойства и методы разделяются с помощью точек (это одна из причин того, что в качестве оператора объединения строк в JavaScript используется знак `+`, а не точка). Рассмотрим, к примеру, в качестве объекта с именем `card` визитную карточку. Этот объект содержит такие свойства, как имя — `name`, адрес — `address`, номер телефона — `phone` и т. д. В синтаксисе JavaScript эти свойства будут иметь следующий вид:

```
card.name
card.phone
card.address
```

Его методами будут функции, занимающиеся извлечением, изменением и другими действиями со свойствами. Например, для вызова метода, отображающего свойства объекта `card`, можно воспользоваться следующим синтаксисом:

```
card.display()
```

Взгляните на некоторые из представленных ранее в этой главе примеров и обратите внимание на те из них, в которых применяется инструкция `document.write`. Уяснив, что JavaScript основан на работе с объектами, вы поймете, что `write` — это метод объекта `document`.

Внутри JavaScript выстраивается иерархия из родительских и дочерних объектов. Эта иерархия и называется объектной моделью документа — DOM (рис. 13.3).

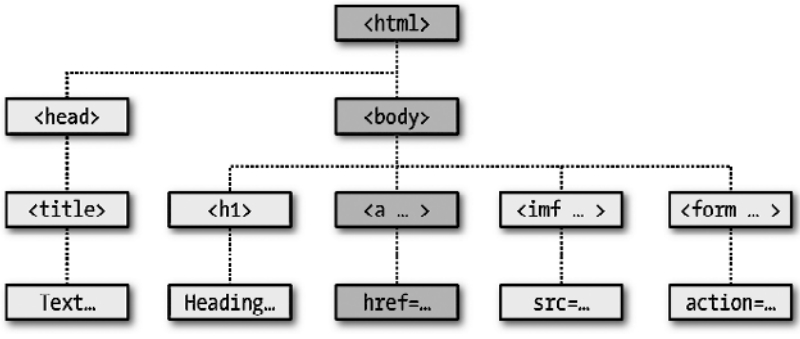

Рис. 13.3. Пример иерархии объектов DOM

На этом рисунке используются уже знакомые вам HTML-теги, иллюстрирующие родительско-дочерние взаимоотношения между различными объектами документа. Например, URL-адрес внутри ссылки является частью тела (body) HTML-документа. В JavaScript на него можно сослаться следующим образом:

```
url = document.links.linkname.href
```

Обратите внимание на то, как эта ссылка идет сверху вниз по центральному столбцу. Первая часть, `document`, ссылается на теги `<html>` и `<body>`, `links.linkname` — на тег `<a>`, а `href` — на атрибут `href`.

Превратим это в некий код HTML и в сценарий для чтения свойств ссылки. Наберите код примера 13.7 и сохраните его в файле с именем `linktest.html`, а затем вызовите в своем браузере.

Пример 13.7. Чтение ссылки на URL-адрес с помощью JavaScript

```
<html>
  <head>
    <title>Тестирование ссылки</title>
  </head>
  <body>
    <a id="mylink" href="http://mysite.com">Щелкни</a><br>
    <script>
      url = document.links.mylink.href
```

```
      document.write('URL-адрес - ' + url)
    </script>
  </body>
</html>
```

Обратите внимание на краткую форму тегов `script`, в которой для экономии времени на набор текста был опущен параметр `type="text/JavaScript"`. При желании ради проверки этого (и других примеров) можете также опустить все, кроме тегов `<script>` и `</script>`. Код этого примера выведет следующую информацию:

```
Щелкни
URL-адрес - http://mysite.com
```

Вторая строка выведенной информации появилась благодаря работе метода `document.write`. Обратите внимание на то, как код следует сверху вниз по дереву документа от `document` к `links`, к `mylink` (идентификатору, присвоенному ссылке) и к `href` (значению, содержащему URL-адрес).

Есть также краткая форма, работающая не менее успешно, которая начинается со значения, присвоенного атрибуту `id`: `mylink.href`. Поэтому следующую строку:

```
url = document.links.mylink.href
```

можно заменить строкой:

```
url = mylink.href
```

Еще одно использование знака $

Как уже упоминалось, символ `$` разрешено использовать в именах переменных и функций JavaScript. По этой причине иногда можно встретить код довольно странного вида:

```
url = $('mylink').href
```

Некоторые изобретательные программисты решили, что метод `getElementByld` слишком часто применяется в JavaScript, и написали взамен него свою функцию, показанную в примере 13.8. Они присвоили ей имя `$`, как в jQuery (хотя в этой библиотеке символ `$` используется более широко, чем здесь, о чем можно узнать, прочитав главу 21).

Пример 13.8. Функция, заменяющая метод getElementByld

```
<script>
  function $(id)
  {
    return document.getElementById(id)
  }
</script>
```

Поэтому, как только функция $ будет вставлена в ваш код, синтаксис:

```
$('mylink').href
```

может заменить следующий код:

```
document.getElementById('mylink').href
```

Использование DOM

На самом деле объект `links` является массивом, состоящим из URL-адресов, поэтому на URL `mylink` в примере 13.7 можно спокойно ссылаться во всех браузерах, используя для этого следующий код (поскольку это первая и единственная ссылка):

```
url = document.links[0].href
```

Если нужно узнать, сколько ссылок содержится во всем документе, можно запросить свойство `length` объекта `links`:

```
numlinks = document. links.length
```

Благодаря этому свойству можно извлечь и отобразить все имеющиеся в документе ссылки:

```
for (j=0 ; j < document.links.length ; ++j)
  document.write(document.links[j].href + '<br>')
```

Длина — `length` является свойством каждого массива, а также многих других объектов. Например, можно запросить количество записей в истории вашего браузера:

```
document.write(history.length)
```

Но, чтобы исключить перехват ваших сайтов с помощью истории браузера, в объекте `history` хранится только количество сайтов в массиве, и вы не можете прочитать или записать значения, относящиеся к этим сайтам. Но вы можете заменить текущую страницу одной из тех, что хранятся в истории, если знаете, на какой позиции она там находится. Это может пригодиться в тех случаях, когда вам известны конкретные страницы, попавшие в историю при переходах с вашей страницы, или вы просто хотите вернуть браузер назад на одну или несколько страниц, что делается с помощью метода `go` объекта `history`. Например, чтобы отправить браузер назад на три страницы, нужно выдать следующую команду:

```
history.go(-3)
```

Для перехода вперед и назад на одну страницу можно воспользоваться также следующими методами:

```
history.back()
history.forward()
```

Подобным способом можно заменить только что загруженный URL-адрес другим по вашему выбору:

```
document.location.href = 'http://google.com'
```

Конечно, DOM можно использовать для решения куда более широкого круга задач, чем чтение и модификация ссылок. По мере изучения следующих глав, посвященных JavaScript, знакомство с DOM и с доступом к этой модели станет еще более тесным.

О функции document.write

При изучении программирования необходимо иметь быстрый и простой способ отображения результатов выражений. В PHP (к примеру) есть инструкции `echo` и `print`, позволяющие просто отправлять текст браузеру, поэтому при работе с ним отображение результатов дается довольно легко. А в JavaScript имеются следующие варианты такого отображения.

Использование console.log

Функция `console.log` выводит результат вычисления значения любой переданной ей переменной или выражения в консоль текущего браузера. Это специальный режим, имеющий фрейм или окно, отдельное от окна браузера, куда можно выводить сообщения об ошибках или другие сообщения. Этот режим больше подходит опытным программистам и не считается идеальным решением для начинающих, поскольку вызов консоли осуществляется во всех браузерах по-разному, то же самое можно сказать и о характере его работы. К тому же его вывод осуществляется отдельно от веб-содержимого браузера.

Использование alert

Функция `alert` выводит значения переданных ей переменных или выражений в появляющемся окне, для закрытия которого требуется нажать кнопку. Конечно, это занятие может быстро надоесть, тем более что есть еще один недостаток, заключающийся в отображении только текущего сообщения и утрате при этом всех предыдущих.

Запись в элементы

У вас есть возможность ввести запись непосредственно в текст HTML-элемента, и она представляется весьма элегантным решением (и лучшим для производства сайтов), за исключением того, что в данной книге для каждого примера понадобится создавать такой элемент, а также добавлять несколько строк кода JavaScript для доступа к этому элементу. Это будет мешать усвоению сути примера и сделает его код излишне громоздким и запутанным.

Использование document.write

Функция `document.write` записывает значение или результат вычисления выражения в текущую позицию в браузере и поэтому является наиболее удачным вариантом для быстрой демонстрации результатов, поскольку примеры остаются довольно краткими и прилично выглядящими за счет того, что вывод помещается прямо в браузере, сразу же за веб-содержимым и кодом.

Возможно, вам приходилось слышать, что некоторые разработчики считают эту функцию небезопасной, поскольку при ее вызове после полностью загруженной страницы она переписывает текущий документ. Хотя это действительно так, к примерам этой книги данная особенность не имеет никакого отношения, поскольку во всех примерах `document.write` используется по первоначальному предназначению — в качестве части процесса создания страницы и вызов функции происходит до полной загрузки и отображения страницы.

Несмотря на то что функция `document.write` используется данным образом для простых примеров, я никогда не пользуюсь ею в своем производственном коде (за исключением весьма редких случаев, когда без этого просто не обойтись). Вместо нее я практически всегда использую предыдущий вариант, заключающийся в записи непосредственно в специально подготовленный элемент, как в показанных далее более сложных примерах в главе 17 (где для программного вывода происходит обращение к свойству элементов `innerHTML`).

Поэтому я прошу запомнить, что вызов функции `document.write`, встречающийся в данной книге, используется исключительно для упрощения примера, и я рекомендую, чтобы вы также использовали эту функцию лишь в подобных целях — для получения быстрых тестовых результатов.

С учетом данного предостережения в следующей главе мы продолжим исследование JavaScript и рассмотрим вопросы управления ходом выполнения программы и написания выражений.

Вопросы

1. Какие теги используются для заключения в них кода JavaScript?
2. К какой части документа будет по умолчанию добавлена информация, выводимая кодом JavaScript?
3. Как в ваши документы может быть включен код JavaScript из другого источника?
4. Какая функция JavaScript является эквивалентом PHP-команд `echo` или `print`?
5. Как можно создать комментарий в JavaScript?
6. Какой оператор используется в JavaScript для объединения строк?

7. Какое ключевое слово можно применять внутри функции JavaScript для определения переменной, имеющей локальную область видимости?
8. Покажите два метода, работающих во всех браузерах и позволяющих отобразить присвоенный ссылке URL-адрес на основе ID этой ссылки.
9. Какие две команды JavaScript заставят браузер загрузить предыдущую страницу, содержащуюся в его массиве `history`?
10. Какой командой JavaScript вы воспользуетесь для замены текущего документа главной страницей сайта oreilly.com?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

14 Выражения и управление процессом выполнения сценариев в JavaScript

В предыдущей главе были изложены основы JavaScript и DOM. А теперь настало время рассмотреть порядок построения в JavaScript сложных выражений и способов управления процессом выполнения ваших сценариев с помощью условных инструкций.

Выражения

Выражения в JavaScript очень похожи на выражения в PHP. В главе 4 мы выяснили, что выражение представляет собой сочетание значений, переменных, операторов и функций, в результате вычисления которого получается значение. Это значение может быть числовым, строковым или логическим (вычисляемым либо как истина — `TRUE`, либо как ложь — `FALSE`).

В примере 14.1 показано несколько простых выражений. Код каждой строки выводит буквы от `a` до `d`, за которыми следуют двоеточие и результат вычисления выражения. Тег `<br>` служит для перехода на новую строку и разделения выводимой информации на четыре строки (в HTML5 допустимо применение как `<br>`, так и `<br />`, но для краткости я выбрал первый стиль).

Пример 14.1. Четыре примера булевых выражений

```
<script>
  document.write("a: " + (42 > 3) + "<br>")
  document.write("b: " + (91 < 4) + "<br>")
  document.write("c: " + (8 == 2) + "<br>")
  document.write("d: " + (4 < 17) + "<br>")
</script>
```

Этот код выведет следующую информацию:

```
a: true
b: false
```

```
c: false
d: true
```

Заметьте, что выражения `a:` и `d:` вычисляются как `true`, а выражения `b:` и `c:` — как `false`. В отличие от языка PHP (который вывел бы число `1` и пустое место соответственно), JavaScript выводит строки `true` и `false`.

В JavaScript при проверке значения на истинность или ложность все выражения вычисляются как `true`, за исключением следующих, которые вычисляются как `false`: самой строки `false`, `0`, `-0`, пустой строки, `null`, неопределенного значения (`undefined`) и `NaN` (Not a Number — «не число», понятие в вычислительной технике для недопустимых операций с числами с плавающей точкой, таких как деление на нуль).

Учтите, что, в отличие от PHP, в JavaScript значения `true` и `false` пишутся в нижнем регистре. Поэтому из следующих двух инструкций информацию будет отображать только первая, выводя на экран слово `true` в нижнем регистре, поскольку вторая инструкция вызовет сообщение об ошибке, где будет сказано, что переменная `TRUE` не определена:

```
if (1 == true) document.write('true') // Истина
if (1 == TRUE) document.write('TRUE') // Приводит к ошибке
```

Следует помнить, что любые фрагменты кода, которые вам захочется набрать и опробовать, запустив их в HTML-файле, нужно заключать в теги <script> и </script>.

Литералы и переменные

Простейшей формой выражения является *литерал*, означающий нечто, вычисляемое само в себя, например число 22 или строка Нажмите клавишу Enter. Выражение может также быть переменной, которая вычисляется в присвоенное ей значение. И литералы, и переменные относятся к типам выражений, поскольку они возвращают значение.

В примере 14.2 показаны три разных литерала и две переменные, и все они возвращают значения, хотя и разных типов.

Пример 14.2. Пять типов литералов

```
<script>
  myname = "Peter"
  myage  = 24
  document.write("a: " + 42     + "<br>")  // Числовой литерал
  document.write("b: " + "Hi"   + "<br>")  // Строковый литерал
  document.write("c: " + true   + "<br>")  // Литерал константы
  document.write("d: " + myname + "<br>")  // Литерал строковой переменной
  document.write("e: " + myage  + "<br>")  // Литерал числовой переменной
</script>
```

Как и можно было ожидать, в выводимой информации будут показаны значения, возвращаемые всеми этими литералами:

```
a: 42
b: Hi
c: true
d: Peter
e: 24
```

Операторы позволяют создавать более сложные выражения, вычисляемые в полезные результаты. При объединении присваивания или управляющей конструкции с выражениями получается *инструкция*.

В примере 14.3 показано по одной инструкции каждого вида. В первой результат выражения `366 – day_number` присваивается переменной `days_to_new_year`, а во второй приятное сообщение выводится только в том случае, если выражение `days_to_new_year < 30` вычисляется как `true`.

Пример 14.3. Две простые инструкции JavaScript

```
<script>
  days_to_new_year = 366 - day_number;
  if (days_to_new_year < 30) document.write("Скоро Новый год!")
</script>
```

Операторы

JavaScript предлагает большое количество мощных операторов, начиная с арифметических, строковых и логических и заканчивая операторами присваивания, сравнения и т. д. (табл. 14.1).

Таблица 14.1. Типы операторов JavaScript

Оператор	Описание	Пример
Арифметический	Основные математические операции	a + b
Для работы с массивами	Работа с массивами	a + b
Присваивания	Присваивание значений	a = b + 23
Поразрядный	Обработка битов в байтах	12 ^ 9
Сравнения	Сравнение двух значений	a < b
Инкремента и декремента	Прибавление или вычитание единицы	a++
Логический	Булевы операции	a && b
Строковый	Объединение	a + 'строка'

Различные типы операторов воспринимают разное количество операндов.

- *Унарные* операторы, к примеру операторы инкремента (a++) или изменения знака числа (–a), воспринимают один операнд.
- *Бинарные* операторы, представленные основной массой операторов JavaScript (включая операторы сложения, вычитания, умножения и деления), воспринимают два операнда.
- И один *трехкомпонентный* (ternary) оператор, имеющий форму `? x : y` и требующий использования трех операндов. Он является краткой однострочной формой инструкции `if`, которая выбирает одно из двух выражений на основе значения третьего выражения.

Приоритетность операторов

Как и в PHP, в JavaScript используется приоритетность операторов, благодаря которой одни операторы в выражении обрабатываются до обработки других операторов и поэтому вычисляются в первую очередь.

В табл. 14.2 перечислены операторы JavaScript, расположенные по уровню их приоритетности.

Таблица 14.2. Операторы JavaScript, расположенные по уровню их приоритетности (сверху вниз)

Оператор (-ы)	**Тип (-ы)**
() [] .	Скобки, вызов и составляющая объекта
++ ––	Инкремент и декремент
+ – ~ !	Унарные, поразрядные и логические
* / %	Арифметические
+ –	Арифметические и строковые
<< >> >>>	Поразрядные
< > <= >=	Сравнения
== != === !==	Сравнения
& ^ |	Поразрядные
&&	Логический
||	Логический
? :	Трехкомпонентный
= += –= *= /= %= <<= >>= >>>= &= ^= |=	Присваивания
,	Разделитель

Взаимосвязанность

Большинство операторов JavaScript обрабатываются в уравнении слева направо. Но для некоторых операторов требуется обработка справа налево. Направление обработки обусловливается *взаимосвязанностью* операторов.

Эта взаимосвязанность вступает в силу при отсутствии явно заданной приоритетности. Рассмотрим, например, следующие операторы присваивания, благодаря которым трем переменным присваивается значение `0`:

```
level = score = time = 0
```

Это множественное присваивание становится возможным только благодаря тому, что самая правая часть выражения вычисляется первой, а затем обработка продолжается справа налево. В табл. 14.3 перечислены операторы JavaScript и их взаимосвязанность.

Таблица 14.3. Операторы и их взаимосвязанность

Оператор (-ы)	Описание	Взаимосвязанность
++ ––	Инкремент и декремент	Отсутствует
new	Создание нового объекта	Правая
+ – ~ !	Унарные и поразрядные операции	Правая
? :	Условный оператор	Правая
= *= /= %= += –=	Присваивание	Правая
<<= >>= >>>= &= ^= |=	Присваивание	Правая
/	Разделитель	Левая
+ – * / %	Арифметические операции	Левая
<< >> >>>	Поразрядные операции	Левая
< <= > >= == != === !==	Операции отношения	Левая

Операторы отношения

Операторы *отношения* проверяют значения двух операндов и возвращают логический результат, равный либо `true`, либо `false`. Существует три типа операторов отношения: операторы *равенства*, *сравнения* и *логические*.

Операторы равенства

Оператор равенства имеет вид `==` (его не следует путать с оператором присваивания `=`). В примере 14.4 первая инструкция присваивает значение, а вторая проверяет это значение на равенство. В данных условиях на экран ничего не будет выведено,

потому что переменной `month` присвоено значение `July` и его сравнение со значением `October` будет неудачным.

Пример 14.4. Присваивание значения и проверка на равенство

```
<script>
  month = "July"
  if (month == "October") document.write("It's the Fall")
</script>
```

Если два операнда выражения равенства принадлежат к разным типам данных, JavaScript приведет их к тому типу, который имеет для него наибольший смысл. Например, любые строки, полностью состоящие из цифр, при сравнении с числом будут преобразованы в числа. В примере 14.5 у переменных `a` и `b` два разных значения (одно из них является числом, а второе — строкой), и поэтому вряд ли стоило ожидать, что какая-нибудь из инструкций `if` выведет результат.

Пример 14.5. Операторы равенства и тождественности

```
<script>
  a = 3.1415927
  b = "3.1415927"
  if (a == b)  document.write("1")
  if (a === b) document.write("2")
</script>
```

Тем не менее, если запустить код этого примера, вы увидите, что он выведет цифру `1`, что будет означать, что первая инструкция `if` была вычислена как `true`. Это произошло благодаря тому, что строковое значение переменной `b` сначала было временно преобразовано в число, и поэтому обе половины уравнения получили числовое значение `3.1415927`.

В отличие от этого, вторая инструкция `if` использует оператор тождественности — три знака равенства подряд, который препятствует автоматическому преобразованию типов в JavaScript. Таким образом, в данном случае значения `a` и `b` считаются разными, поэтому на экран ничего не выводится.

Точно так же, как при принудительном задании приоритета операторов, если у вас появятся сомнения, связанные с преобразованиями типов операндов, производимыми JavaScript, для отключения этого поведения можно воспользоваться оператором тождественности.

Операторы сравнения

Используя операторы сравнения, можно проверить не только равенство или неравенство. JavaScript предоставляет в ваше распоряжение также операторы `>` (больше), `<` (меньше), `>=` (больше или равно) и `<=` (меньше или равно). Код примера 14.6 показывает эти операторы в действии.

Пример 14.6. Четыре оператора сравнения

```
<script>
  a = 7; b = 11
  if (a > b)  document.write("a больше b<br>")
  if (a < b)  document.write("a меньше b<br>")
  if (a >= b) document.write("a больше или равно b<br>")
  if (a <= b) document.write("a меньше или равно b<br>")
</script>
```

Если a равно 7, а b равно 11, то код этого примера выведет следующую информацию (потому что 7 меньше 11, а также меньше или равно 11):

```
a меньше b
a меньше или равно b
```

Логические операторы

Логические операторы выдают истинные или ложные результаты, поэтому их также называют *булевыми*. В JavaScript используются три логических оператора (табл. 14.4).

Таблица 14.4. Логические операторы JavaScript

Логический оператор	Описание
&&	(И) Возвращает истинное значение (true), если оба операнда имеют истинные значения
||	(ИЛИ) Возвращает истинное значение (true), если любой из операторов имеет истинное значение
!	(НЕ) Возвращает истинное значение (true), если операнд имеет ложное значение, или ложное значение (false), если операнд имеет истинное значение

Код примера 14.7 показывает возможности использования этих операторов, при которых возвращаются 0, 1 и true.

Пример 14.7. Использование логических операторов

```
<script>
  a = 1; b = 0
  document.write((a && b) + "<br>")
  document.write((a || b) + "<br>")
  document.write((  !b  ) + "<br>")
</script>
```

Оператору && для возвращения значения true нужно, чтобы оба операнда имели значение true. Оператору || для возвращения значения true необходимо, чтобы любой операнд имел значение true, а третий оператор применяет к значению переменной b операцию НЕ, превращая ее значение из 0 в true.

При использовании оператора || могут возникнуть непредвиденные проблемы, поскольку второй операнд не будет вычисляться, если при вычислении первого будет получен результат true. В примере 14.8 функция getnext никогда не будет вызвана, если переменная finished имеет значение 1.

Пример 14.8. Инструкция, использующая оператор ||

```
<script>
  if (finished == 1 || getnext() == 1) done = 1
</script>
```

Если *нужно*, чтобы getnext была вызвана при каждом выполнении инструкции if, следует переписать код, как показано в примере 14.9.

Пример 14.9. Инструкция if...ИЛИ, измененная для гарантированного вызова getnext

```
<script>
  gn = getnext()
  if (finished == 1 || gn == 1) done = 1
</script>
```

В данном случае код функции getnext будет выполнен и он вернет значение, которое будет сохранено в переменной gn до выполнения инструкции if.

В табл. 14.5 показаны все допустимые варианты использования логических операторов. Следует заметить, что выражение !true эквивалентно false, а выражение !false эквивалентно true.

Таблица 14.5. Все логические выражения, допустимые в JavaScript

Входные данные		**Операторы и результаты**	
a	**b**	**&&**	**II**
true	true	true	true
true	false	false	true
false	true	true	true
false	false	false	false

Инструкция with

Инструкция with в предыдущих главах, посвященных PHP, не встречалась, поскольку она имеется только в JavaScript. Используя эту инструкцию (если вы понимаете, что я под этим подразумеваю), можно упростить некоторые типы инструкций JavaScript, сократив множество ссылок на объект до всего одной ссылки. Предполагается, что ссылки на свойства и методы внутри блока with должны применяться к указанному объекту.

Рассмотрим код примера 14.10, в котором функция document.write нигде не ссылается на переменную string по имени.

Пример 14.10. Использование инструкции with

```
<script>
  string = "Шустрая бурая лисица перепрыгивает через ленивую собаку"

  with (string)
  {
    document.write("В строке " + length + " символов <br>")
    document.write("В верхнем регистре: " + toUpperCase())
  }
</script>
```

Даже притом, что в `document.write` нет непосредственной ссылки на `string`, этот код все равно справляется с выводом следующей информации:

```
В строке 55 символов
В верхнем регистре: ШУСТРАЯ БУРАЯ ЛИСИЦА ПЕРЕПРЫГИВАЕТ ЧЕРЕЗ ЛЕНИВУЮ СОБАКУ
```

Код примера работает следующим образом: интерпретатор JavaScript распознает, что свойство `length` и метод `toUpperCase` должны быть применены к какому-то объекту. Поскольку они указаны только сами по себе, интерпретатор предполагает, что они применяются к объекту `string`, указанному в инструкции `with`.

Использование события onerror

Рассмотрим еще одну конструкцию, недоступную в PHP. Используя либо событие `onerror`, либо сочетание ключевых слов `try` и `catch`, можно перехватить ошибки JavaScript и самостоятельно справиться с ними.

Событиями называются действия, которые могут быть обнаружены JavaScript. Каждый элемент на веб-странице имеет конкретные события, которыми могут быть приведены в действие функции JavaScript. Например, событие щелчка — `onclick`, принадлежащее элементу `button` (кнопка), может быть настроено на вызов функции, которая будет запущена, если пользователь нажмет кнопку. В примере 14.11 показано, как можно воспользоваться событием `onerror`.

Пример 14.11. Сценарий, использующий событие onerror

```
<script>
  onerror = errorHandler
  document.writ("Добро пожаловать на этот сайт!") // Преднамеренная ошибка

  function errorHandler(message, url, line)
  {
    out  = "К сожалению, обнаружена ошибка.\n\n";
    out += "Ошибка: " + message + "\n";
    out += "URL: " + url + "\n";
    out += "Строка: " + line + "\n\n";
    out += "Нажмите кнопку OK для продолжения работы. \n\n ";
    alert(out);
    return true;
  }
</script>
```

В первой строке сценария событию ошибки предписывается впредь использовать новую функцию `errorHandler`. Эта функция принимает три параметра: сообщение, URL-адрес и номер строки, поэтому остается лишь отобразить все это в появляющемся окне метода `alert`.

Затем для проверки работы новой функции в коде сценария преднамеренно допускается ошибка: вызывается `document.writ` вместо `document.write` (не ставится последняя буква «е»). На рис. 14.1 показан результат запуска этого сценария в браузере. Подобное использование события `onerror` может пригодиться при отладке сценария.

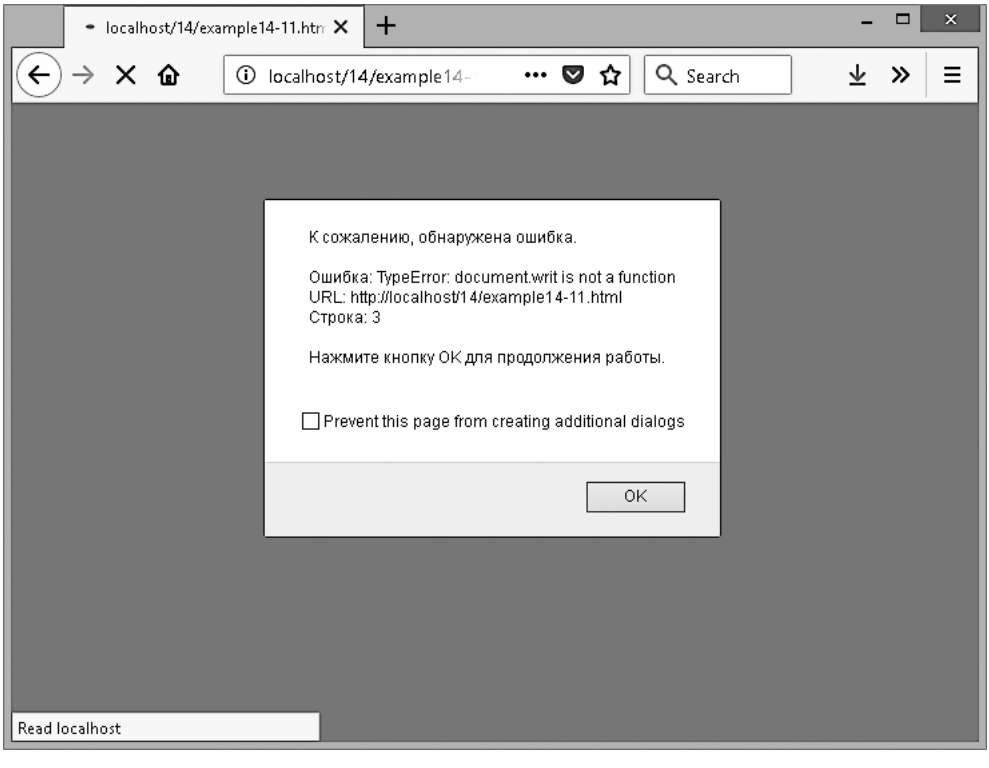

Рис. 14.1. Использование события onerror с методом alert для вывода информации

Конструкция try...catch

Технология, в которой применяются ключевые слова `try` и `catch`, считается более стандартной и гибкой, чем обработка события `onerror`, показанная в предыдущем разделе. Эти ключевые слова позволяют перехватывать ошибки для избранного раздела кода, а не для всех сценариев, имеющихся в документе. Но данная

технология не позволяет перехватывать синтаксические ошибки, для чего приходится применять обработку события `onerror`.

Конструкция `try...catch` поддерживается всеми основными браузерами и задействуется в тех случаях, когда нужно перехватить управление при наступлении конкретных условий, о которых известно то, что они могут сложиться в определенной части вашего кода.

Например, в главе 17 будет рассматриваться технология AJAX, в которой используется объект `XMLHttpRequest`. Поэтому для обеспечения совместимости нужно применять `try` и `catch`, чтобы перехватить управление при отсутствии этого объекта и задействовать какие-нибудь другие технологии. Как это делается, показано в примере 14.12.

Пример 14.12. Перехват ошибки с помощью ключевых слов try и catch

```
<script>
  try
  {
    request = new XMLHTTPRequest()
  }
  catch(err)
  {
    // Использование другого метода для создания объекта XMLHTTPRequest
  }
</script>
```

Я не хочу здесь вникать в подробности реализации отсутствующего в Internet Explorer объекта, но вы можете увидеть, как работает эта система. Со словами `try` и `catch` связано еще одно ключевое слово — `finally`. Блок кода, следующий за этим словом, выполняется всегда, независимо от того, возникла или не возникла ошибка при выполнении блока кода под ключевым словом `try`. Чтобы воспользоваться этим ключевым словом, нужно после инструкции `catch` просто добавить что-нибудь похожее на следующий пример:

```
finally
{
  alert("Был обнаружен блок кода 'try'.")
}
```

Условия

Условия изменяют процесс выполнения программы. Они позволяют задавать конкретные вопросы и по-разному реагировать на полученные ответы. Существуют три типа условий, не связанных с циклами: инструкция `if`, инструкция `switch` и оператор `?`.

Инструкция if

Инструкции `if` уже использовались в нескольких примерах данной главы. Код внутри этой инструкции выполняется только в том случае, если заданное выражение вычисляется как `true`. Многострочные инструкции `if` заключаются в фигурные скобки, но, как и в PHP, для однострочных инструкций скобки можно опустить. Поэтому допустимы все следующие инструкции:

```
if (a > 100)
{
  b=2
  document.write("a больше 100")
}

if (b == 10) document.write("b равно 10")
```

Инструкция else

Если условие не было соблюдено, то с помощью инструкции `else` может быть выполнен альтернативный блок кода:

```
if (a > 100)
{
  document.write("a больше 100")
}
else
{
  document.write("a меньше или равно 100")
}
```

В JavaScript, в отличие от PHP, нет инструкции `elseif`, но ее отсутствие компенсируется возможностью использования инструкции `else`, за которой следует еще одна инструкция `if`, чем создается эквивалент инструкции `elseif`:

```
if (a > 100)
{
  document.write("a больше 100")
}
else if(a < 100)
{
  document.write("a меньше 100")
}
else
{
  document.write("a равно 100")
}
```

За инструкцией `else` после новой инструкции `if` точно так же может следовать еще одна инструкция `if` и т. д. Несмотря на то что в этих инструкциях использованы

фигурные скобки, наличие внутри каждой пары этих скобок всего одной строки кода позволяет переписать весь пример следующим образом:

```
if     (a > 100) document.write("a больше 100")
else if(a < 100) document.write("a меньше 100")
else             document.write("a равно 100")
```

Инструкция switch

Инструкция `switch` применяется в том случае, когда одиночная переменная или результат вычисления выражения может иметь несколько значений, для каждого из которых нужно применить свою функцию. Например, в следующем коде за основу берется система меню PHP, составленная в главе 4, которая переводится в инструкции JavaScript. Эта система работает путем передачи одного строкового значения, соответствующего пожеланию пользователя, коду основного меню. Предположим, пользователю доступны следующие варианты: Home, About, News, Login и Links, — и переменной page присваивается одно из этих значений, соответствующее тому, что ввел пользователь.

Код, созданный для этого с помощью конструкции `if...else if...`, может иметь вид, показанный в примере 14.13.

Пример 14.13. Многострочная конструкция if...else if...

```
<script>
  if      (page == "Home")  document.write("Вы выбрали Home")
  else if (page == "About") document.write("Вы выбрали About")
  else if (page == "News")  document.write("Вы выбрали News")
  else if (page == "Login") document.write("Вы выбрали Login")
  else if (page == "Links") document.write("Вы выбрали Links")
</script>
```

А при использовании конструкции `switch` код может иметь вид, показанный в примере 14.14.

Пример 14.14. Конструкция switch

```
<script>
  switch (page)
  {
    case "Home":
      document.write("Вы выбрали Home")
      break
    case "About":
      document.write("Вы выбрали About")
      break
    case "News":
      document.write("Вы выбрали News")
      break
```

```
    case "Login":
      document.write("Вы выбрали Login")
      break
    case "Links":
      document.write("Вы выбрали Links")
      break
  }
</script>
```

Переменная `page` здесь присутствует только в самом начале инструкции `switch`. После чего совпадения проверяются командой `case`. При совпадении выполняется условная инструкция. Разумеется, в настоящей программе здесь будет код для отображения страницы или для перехода на нее, а не простое сообщение пользователю о том, что он выбрал.

Вы можете также предложить несколько случаев для одного действия. Например:

```
switch (heroName)
{
  case "Superman":
  case "Batman":
  case "Wonder Woman":
    document.write("Justice League")
    break
  case "Iron Man":
  case "Captain America":
  case "Spiderman":
    document.write("The Avengers")
    break
}
```

Прекращение работы инструкции switch

Рассматривая код примера 14.14, можно заметить, что, как и в PHP, команда `break` позволяет сценарию прекратить работу инструкции `switch` при соблюдении условия. Если вы не хотите, чтобы выполнение всех инструкций, начиная со следующей `case`, продолжилось, не забудьте поставить команду `break`.

Действие по умолчанию

С помощью ключевого слова `default` для инструкции `switch` можно определить действие по умолчанию на тот случай, когда не будет выполнено ни одно из условий. В примере 14.15 показан фрагмент кода, который может быть вставлен в код примера 14.14.

Пример 14.15. Инструкция default, предназначенная для кода примера 14.14

```
default:
  document.write("Нераспознанный выбор")
  break
```

Оператор ?

Трехкомпонентный оператор, состоящий из вопросительного знака (`?`), применяемого в сочетании с символом двоеточия (`:`), является упрощенной формой текста `if...else`. Используя этот оператор, можно поставить за вычисляемым выражением знак `?` и код, выполняемый в том случае, если выражение вычисляется как `true`. После этого кода ставится знак `:` и код, который будет выполнен, если выражение будет вычислено как `false`.

В примере 14.16 показан трехкомпонентный оператор, используемый для вывода сообщения о том, что значение переменной `a` меньше или равно 5, или для вывода другого сообщения, если это утверждение не соответствует действительности.

Пример 14.16. Использование трехкомпонентного оператора

```
<script>
  document.write(
    a <= 5 ?
    "a меньше или равно 5" :
    "a больше 5"
  )
</script>
```

Для более понятного представления этот оператор был разбит на несколько строк, но вы, скорее всего, воспользуетесь его однострочной формой:

```
size = a <= 5 ? "короткий" : "длинный"
```

Циклы

При рассмотрении циклов нам опять встретится множество параллелей между JavaScript и PHP. В обоих языках поддерживаются циклы `while`, `do...while` и `for`.

Циклы while

В JavaScript в циклах `while` сначала проверяется значение выражения, а выполнение инструкций внутри цикла начинается лишь в том случае, если выражение вычисляется как `true`. Если выражение вычисляется как `false`, управление переходит к следующей инструкции JavaScript (если таковая имеется).

После завершения итерации цикла выражение опять проверяется на истинность и процесс продолжается до тех пор, пока не наступит момент, когда выражение будет вычислено как `false`, или пока выполнение не будет остановлено по какой-нибудь другой причине. Этот цикл показан в примере 14.17.

Пример 14.17. Цикл while

```
<script>
  counter=0

  while (counter < 5)
  {
    document.write("Счетчик: " + counter + "<br>")
    ++counter
  }
</script>
```

Этот сценарий выведет следующую информацию:

```
Счетчик: 0
Счетчик: 1
Счетчик: 2
Счетчик: 3
Счетчик: 4
```

Если бы переменная counter не увеличивалась на единицу внутри цикла, то вполне возможно, что некоторые браузеры перестали бы откликаться из-за входа в бесконечный цикл и работу со страницей было бы трудно остановить даже нажатием клавиши Esc или кнопки остановки загрузки страницы. Поэтому к циклам в JavaScript нужно относиться с большой осторожностью.

Циклы do...while

Когда нужен цикл, в котором еще до того, как будет проверено выражение, должна пройти хотя бы одна итерация, используется цикл `do...while`, который похож на цикл `while`, за исключением того, что проверка выражения осуществляется только после каждой итерации цикла. Поэтому для вывода первых семи результатов таблицы умножения на 7 можно воспользоваться кодом, показанным в примере 14.18.

Пример 14.18. Цикл do...while

```
<script>
  count = 1

  do
  {
    document.write(count + " умножить на 7 равно " + count * 7 + "<br>")
  } while (++count <= 7)
</script>
```

Как и ожидалось, этот цикл выведет следующую информацию:

```
1 умножить на 7 равно 7
2 умножить на 7 равно 14
3 умножить на 7 равно 21
4 умножить на 7 равно 28
5 умножить на 7 равно 35
6 умножить на 7 равно 42
7 умножить на 7 равно 49
```

Циклы for

Цикл `for` объединяет все лучшие качества организации цикла в одной конструкции, которая позволяет передать каждой инструкции три параметра:

- выражение инициализации;
- выражение условия;
- выражение модификации.

Эти параметры отделяются друг от друга точками с запятыми: `for (`*`выражение1 ; выражение2 ; выражение3`*`)`. С началом первой итерации цикла выполняется выражение инициализации. Для кода вывода таблицы умножения на 7 переменная `count` будет инициализирована значением `1`. Затем после каждого прохождения цикла будет проверено выражение условия (в данном случае `count <= 7`), и новое вхождение в цикл произойдет только в том случае, если выражение условия будет вычислено как `true`. И в завершение в конце каждой итерации будет вычислено выражение модификации. В случае с таблицей умножения на 7 значение переменной `count` увеличивается на единицу. В примере 14.19 показан код такого цикла.

Пример 14.19. Использование цикла for

```
<script>
  for (count = 1 ; count <= 7 ; ++count)
  {
    document.write(count + " умножить на 7 равно " + count * 7 + "<br>");
  }
</script>
```

Как и в PHP, в первом параметре цикла `for` можно присваивать значения сразу нескольким переменным, разделяя выражения запятыми:

```
for (i = 1, j = 1 ; i < 10 ; i++)
```

Точно так же в последнем параметре можно осуществлять сразу несколько модификаций:

```
for (i = 1 ; i < 10 ; i++, --j)
```

Или можно одновременно делать и то и другое:

```
for (i = 1, j = 1 ; i < 10 ; i++, --j)
```

Прекращение работы цикла

Команду `break`, о важности использования которой в инструкции `switch` вы уже знаете, можно применять и внутри циклов `for`. Например, она может пригодиться при поиске совпадений определенного вида. Как только совпадение будет найдено, продолжение поиска станет пустой тратой времени и заставит вашего

посетителя ждать его завершения. Использование команды `break` показано в примере 14.20.

Пример 14.20. Использование команды break в цикле for

```
<script>
  haystack     = new Array()
  haystack[17] = "Иголка"

  for (j = 0 ; j < 20 ; ++j)
  {
    if (haystack[j] == "Иголка")
    {
      document.write("<br>- найдена в элементе " + j)
      break
    }
    else document.write(j + ", ")
  }
</script>
```

Этот сценарий выводит следующую информацию:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
- найдена в элементе 17
```

Инструкция continue

Иногда нужно не выйти из цикла, а пропустить выполнение тех инструкций, которые остались в данной итерации. В таких случаях можно воспользоваться командой `continue`. Ее применение показано в примере 14.21.

Пример 14.21. Использование команды continue в цикле for

```
<script>
  haystack     = new Array()
  haystack[4]  = "Иголка"
  haystack[11] = "Иголка"
  haystack[17] = "Иголка"

  for (j = 0 ; j < 20 ; ++j) {
    if (haystack[j] == "Иголка")
    {
      document.write("<br>- найдена в элементе " + j + "<br>")
      continue
    }

    document.write(j + ", ")
  }
</script>
```

Обратите внимание на то, что второй вызов метода `document.write` не нужно помещать в инструкцию `else` (как было в предыдущем примере), поскольку, если будет

найдено совпадение, в результате выполнения команды `continue` данный вызов будет пропущен. Этот сценарий выводит следующую информацию:

```
0, 1, 2, 3,
- найдена в элементе 4
5, 6, 7, 8, 9, 10,
- найдена в элементе 11
12, 13, 14, 15, 16,
- найдена в элементе 17
18, 19,
```

Явное преобразование типов

В отличие от PHP, в JavaScript нет явного преобразования типов, осуществляемого с помощью операторов `int` или `float`. Когда нужно, чтобы значение имело определенный тип данных, используется одна из встроенных функций JavaScript, показанных в табл. 14.6.

Таблица 14.6. Функции изменения типа, используемые в JavaScript

Преобразование в тип данных	Используемая функция
Int, Integer	parseInt()
Bool, Boolean	Boolean()
Float, Double, Real	parseFloat()
String	String()
Array	spNt()

Например, чтобы преобразовать число с плавающей точкой в целое число, можно использовать следующий код (который выводит значение `3`):

```
n = 3.1415927
i = parseInt(n)
document.write(i)
```

Или можно воспользоваться составной формой:

```
document.write(parseInt(3.1415927))
```

На этом рассмотрение выражений и способов управления процессом выполнения сценариев завершается. В следующей главе мы рассмотрим использование в JavaScript функций, объектов и массивов.

Вопросы

1. Чем отличается обработка логических значений в PHP от их обработки в JavaScript?
2. Какие символы используются для определения имени переменной в JavaScript?

3. В чем разница между унарными, бинарными и трехкомпонентными операторами?
4. Как лучше всего принудительно установить собственный приоритет для оператора?
5. В каком случае следует использовать оператор тождественности (`===`)?
6. Какие две формы выражений считаются самыми простыми?
7. Назовите три типа условных инструкций.
8. Как в инструкциях `if` и `while` интерпретируются условные выражения, в которых используются данные, относящиеся к разным типам?
9. Почему цикл `for` считается мощнее цикла `while`?
10. Для чего предназначена инструкция `with`?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

15 Функции, объекты и массивы JavaScript

В JavaScript предоставляется практически такой же доступ к функциям и объектам, как и в PHP. Язык JavaScript фактически основан на объектах, поскольку, как вы уже поняли, у него есть доступ к DOM — модели, которая реализует доступ к каждому элементу HTML-документа, позволяя работать с ним как с объектом. Способы применения функций и объектов, а также используемый для этого синтаксис очень похожи на все то, что мы видели в PHP, поэтому при изучении этого материала, а также при углубленном рассмотрении массивов JavaScript вы будете чувствовать себя вполне уверенно.

Функции JavaScript

В JavaScript наряду с доступом к десяткам встроенных функций (или методов), среди которых и метод `write`, который, как мы уже видели, использовался в вызовах `document.write`, можно создавать и собственные функции. Как только появляется какой-нибудь непростой фрагмент кода с перспективами на многократное использование, он становится кандидатом на оформление в виде функции.

Определение функции

В общем виде синтаксис функции выглядит следующим образом:

```
function имя_функции([параметр [, ...]])
{
  инструкции
}
```

Из первой строки синтаксиса следует, что:

- определение начинается со слова `function`;
- следующее за этим словом имя должно начинаться с буквы или символа подчеркивания, за которым следует любое количество букв, цифр, символов доллара или подчеркивания;
- необходимо использовать скобки;
- дополнительно могут применяться один или несколько параметров, разделенных запятыми (о чем свидетельствуют квадратные скобки, не являющиеся частью синтаксиса функции).

Имена функций чувствительны к регистру букв, поэтому строки `getInput`, `GETINPUT` и `getinput` ссылаются на разные функции.

В JavaScript для имен функций действует общепринятое соглашение: первая буква каждого слова в имени, за исключением первой буквы всего имени, должна быть прописной. Поэтому в приведенных примерах имен предпочтение следует отдать имени `getinput`, имеющему формат, используемый большинством программистов. Это соглашение часто называют bumpyCaps (неровностями из прописных букв) или camelCase (верблюжий регистр).

Инструкции, которые будут выполняться после вызова функции, начинаются с открывающей фигурной скобки, а составляющая ей пару закрывающая фигурная скобка должна завершать перечень этих инструкций. Среди инструкций обязаны присутствовать одна или несколько инструкций `return`, которые заставляют функцию прекратить выполнение и вернуть управление вызвавшему ее коду. Если к инструкции `return` прилагается какое-нибудь значение, то вызывающий код может его извлечь.

Массив аргументов

Составной частью каждой функции является массив аргументов — `arguments`. Благодаря ему можно определить количество переменных, переданных функции, и понять, что они собой представляют. Рассмотрим, например, функцию `displayItems`. Один из способов ее создания показан в примере 15.1.

Пример 15.1. Определение функции

```
<script>
  displayItems("Собака", "Кошка", "Пони", "Хомяк", "Черепаха")

  function displayItems(v1, v2, v3, v4, v5)
  {
    document.write(v1 + "<br>")
    document.write(v2 + "<br>")
    document.write(v3 + "<br>")
    document.write(v4 + "<br>")
    document.write(v5 + "<br>")
  }
</script>
```

После вызова этого сценария на экране браузера отобразится следующая информация:

Собака
Кошка
Пони
Хомяк
Черепаха

А что делать, если функции нужно будет передать больше пяти аргументов? К тому же применение вместо цикла многократного вызова метода `document.write` считается слишком расточительным приемом программирования. К счастью, массив `arguments` позволяет приспособиться к обработке любого количества аргументов. В примере 15.2 показана возможность использования этого массива для придания предыдущему примеру более рациональной формы.

Пример 15.2. Модификация функции для использования массива аргументов

```
<script>
  function displayItems()
  {
    for (j = 0 ; j < displayItems.arguments.length ; ++j)
      document.write(displayItems.arguments[j] + "<br>")
  }
</script>
```

Обратите внимание на использование свойства `length`, которое уже встречалось в предыдущей главе, а также на то, как с помощью переменной `j`, являющейся индексным смещением внутри массива, осуществляется ссылка на элементы `displayItems.arguments`. Поскольку тело цикла `for` состоит всего из одной инструкции, я решил не заключать его в фигурные скобки, чтобы не загромождать код функции. Не забудьте, что цикл должен остановиться, когда значение переменной `j` будет на единицу меньше значения `length`, а не равно ему.

Теперь благодаря данной технологии у вас есть функция, способная принимать любое количество аргументов и делать с каждым аргументом все, что вам захочется.

Возвращение значения

Функции используются не только для отображения информации. Чаще всего они применяются для выполнения вычислений или обработки данных с возвращением полученного результата. Функция `fixNames`, показанная в примере 15.3, задействует массив `arguments` (рассмотренный в предыдущем пункте) для приема переданной ей последовательности строк и возвращения всех этих строк в виде одной строки. Слово `fix` (исправление) в ее имени означает, что она переводит каждый символ в аргументах в нижний регистр, делая исключение для первой буквы каждого аргумента, которую она превращает в прописную.

Пример 15.3. Приведение в порядок полного названия

```
<script>
  document.write(fixNames("the", "DALLAS", "CowBoys"))

  function fixNames()
  {
    var s = ""

    for (j = 0 ; j < fixNames.arguments.length ; ++j)
      s += fixNames.arguments[j].charAt(0).toUpperCase() +
           fixNames.arguments[j].substr(1).toLowerCase() + " "

    return s.substr(0, s.length-1)
  }
</script>
```

К примеру, если вызвать эту функцию с параметрами `the`, `DALLAS` и `CowBoys`, то она вернет строку `The Dallas Cowboys`. Проанализируем работу этой функции. Сначала она инициализирует временную (и локальную) переменную `s`, присваивая ей значение пустой строки. Затем с помощью цикла `for` осуществляется последовательный перебор каждого переданного параметра с выделением его первой буквы с помощью метода `charAt` и переводом ее в верхний регистр с помощью метода `toUpperCase`. Все методы, показанные в этом примере, встроены в JavaScript и доступны по умолчанию.

После этого для извлечения оставшейся части каждой строки используется метод `substr`, а для перевода букв этой части строки в нижний регистр — метод `toLowerCase`. Если здесь применить полную версию вызова метода `substr`, то в ней вторым аргументом нужно указать, из какого количества символов будет состоять извлекаемая подстрока:

```
substr(1, (arguments[j].length) – 1 )
```

Иными словами, в этом вызове метода `substr` говорится следующее: «Начни с символа, который находится в позиции 1 (это второй символ), и верни оставшуюся часть строки (равную длине строки — `length` за вычетом одного символа)». Но, что особенно приятно, метод `substr` заранее предполагает, что если второй аргумент опущен, то вам нужна вся оставшаяся часть строки.

После того как весь аргумент будет переделан для получения нужного результата, к нему добавляется пробел и результат присоединяется к значению временной переменной `s`.

На завершающей стадии к содержимому переменной `s` опять применяется метод `substr`. Поскольку последний пробел нам не нужен, мы используем метод `substr`, чтобы вернуть всю строку, за исключением ее последнего символа.

Этот пример особенно интересен тем, что в нем показано использование в одном выражении сразу нескольких свойств и методов, например:

```
fixNames.arguments[j].substr(1).toLowerCase()
```

Чтобы понять суть этой инструкции, ее нужно мысленно разделить на части, используя в качестве разделителей точки. JavaScript вычисляет эти элементы инструкции слева направо.

1. Берется имя функции: `fixNames`.
2. Из массива аргументов этой функции извлекается элемент `j`.
3. К извлеченному элементу применяется метод `substr` с параметром `1`. Благодаря этому следующей части выражения будет передан весь извлеченный элемент, за исключением первого символа.
4. К той строке, которая только что была передана, применяется метод `toLowerCase`.

Такие построения часто называют *составлением цепочки методов*. Если, к примеру, приведенному здесь выражению передать строку `mixedCASE`, то она пройдет через следующие преобразования:

```
mixedCase
ixedCase
ixedcase
```

Иными словами, `fixNames.arguments[j]` выдает `mixedCASE`, затем `substr(1)` получает `mixedCASE` и выдает `ixedCASE`, и наконец `toLowerCase()` получает `ixedCASE` и выдает `ixedcase`.

И последнее напоминание: созданная внутри функции переменная `s` имеет локальную область видимости и не может быть доступна за ее пределами. Возвращая `s` с помощью инструкции `return`, мы делаем ее значение доступным вызывавшему функцию коду, который может его сохранить или использовать любым другим способом.

Но с завершением работы функции сама переменная `s` исчезает. Хотя мы можем заставить функцию работать и с глобальными переменными (что иногда просто необходимо), все-таки лучше просто вернуть те значения, которые нужно сохранить, и дать возможность JavaScript избавиться от всех остальных переменных, которые использовались функцией.

Возвращение массива

В примере 15.3 функция возвращает только один параметр. А что делать, если нужно вернуть сразу несколько параметров? Решить задачу поможет возвращение массива, показанное в примере 15.4.

Пример 15.4. Возвращение массива значений

```
<script>
  words = fixNames("the", "DALLAS", "CowBoys")

  for (j = 0 ; j < words.length ; ++j)
```

```
    document.write(words[j] + "<br>")

  function fixNames()
  {
    var s = new Array()

    for (j = 0 ; j < fixNames.arguments.length ; ++j)
      s[j] = fixNames.arguments[j].charAt(0).toUpperCase() +
             fixNames.arguments[j].substr(1).toLowerCase()

  return s
}
</script>
```

Здесь переменная `words` автоматически определяется в виде массива и заполняется результатами, возвращенными вызовом функции `fixNames`. Затем цикл `for` осуществляет последовательный перебор элементов массива, отображая каждый элемент.

Что касается функции `fixNames`, то она практически идентична использованной в примере 15.3, за исключением того, что переменная `s` теперь является массивом, возвращаемым с помощью инструкции `return`.

Эта функция позволяет извлекать отдельные параметры из возвращенных ею элементов, например, как в следующем коде (который выводит строку `The Cowboys`):

```
words = fixNames("the", "DALLAS", "CowBoys")
document.write(words[0] + " " + words[2])
```

Объекты JavaScript

По сравнению с переменными, которые в каждый конкретный момент могут содержать только одно значение, объекты JavaScript — это значительный шаг вперед, поскольку в них может содержаться несколько значений и даже функций. Объект группирует вместе данные и функции для работы с ними.

Объявление класса

При создании сценария, в котором используются объекты, необходимо спроектировать структуру из данных и кода, называемую *классом*. Каждый новый объект, основанный на конкретном классе, называется *экземпляром* класса. Вам уже известно, что данные, связанные с объектом, называются его *свойствами*, а функции, которые им используются, называются *методами*.

Посмотрим, как объявляется класс для объекта по имени `User`, который будет содержать сведения о текущем пользователе. Для создания класса нужно просто создать функцию с именем этого класса.

Эта функция может воспринимать аргументы (позже будет показано, как она вызывается) и создавать свойства и методы для объектов класса. Такая функция называется *конструктором*.

В примере 15.5 показан конструктор для класса `User`, имеющий три свойства: имя — `forename`, пользовательское имя — `username` и пароль — `password`. В классе также определяется метод демонстрации сведений о пользователе — `showUser`.

Пример 15.5. Объявление класса User и его метода

```
<script>
  function User(forename, username, password)
  {
    this.forename = forename this.username = username
    this.password = password

    this.showUser = function()
    {
      document.write("Имя: " + this.forename + "<br>")
      document.write("Пользовательское имя: " + this.username + "<br>")
      document.write("Пароль: " + this.password + "<br>")
    }
  }
</script>
```

От ранее рассмотренных эту функцию отличают несколько особенностей.

- При каждом вызове функции ею создается новый объект. То есть, к примеру, можно вызвать одну и ту же функцию многократно с разными аргументами, чтобы создать объекты для пользователей с разными значениями свойства имени `forename`.
- Она ссылается на объект по имени `this`, который является ссылкой на создаваемый экземпляр. В примере показано, что имя `this` используется объектом для установки своих собственных свойств, которые для разных экземпляров `User` будут иметь разные значения.
- Внутри этой функции создается новая функция по имени `showUser`. Здесь показан новый, усложненный синтаксис. Его задача — привязать `showUser` к классу `User`. Таким образом, `showUser` становится методом класса `User`.

Здесь используется соглашение о выборе имен, согласно которому все свойства получают имена, состоящие из букв в нижнем регистре, а в имени метода в соответствии с упомянутым в данной главе соглашением bumpyCaps есть по крайней мере одна прописная буква.

В примере 15.5 соблюдается рекомендуемый способ создания конструктора класса, который состоит в том, что методы включаются в функции конструктора. Но в примере 15.6 показано, что можно также ссылаться на те функции, которые определены за границами конструктора.

Пример 15.6. Раздельное объявление класса и метода

```
<script>
  function User(forename, username, password)
  {
    this.forename = forename
    this.username = username
    this.password = password
    this.showUser = showUser
  }

  function showUser()
  {
    document.write("Имя: " + this.forename + "<br>")
    document.write("Пользовательское имя: " + this.username + "<br>")
    document.write("Пароль: " + this.password + "<br>")
  }
</script>
```

Эта форма объявления класса показана с расчетом на то, что вам наверняка придется сталкиваться с использованием кода, созданного другими программистами.

Создание объекта

Для создания экземпляра класса `User` можно воспользоваться следующей инструкцией:

```
details = new User("Wolfgang", "w.a.mozart", "composer")
```

Или же можно создать пустой объект:

```
details = new User()
```

а затем наполнить его содержимым:

```
details.forename = "Wolfgang"
details.username = "w.a.mozart"
details.password = "composer"
```

К объекту также можно добавлять новые свойства:

```
details.greeting = "Привет"
```

Проверить работу только что добавленного свойства можно с помощью следующей инструкции:

```
document.write(details.greeting)
```

Доступ к объектам

Для доступа к объекту можно сослаться на его свойства, как показано в следующих не связанных друг с другом примерах инструкций:

```
name = details.forename
if (details.username == "Admin") loginAsAdmin()
```

А для доступа к методу `showUser`, принадлежащему объекту класса `User`, нужно воспользоваться следующим синтаксисом, в котором применяется уже созданный и заполненный данными объект `details`:

```
details.showUser()
```

В соответствии с ранее предоставленными объекту данными этот код отобразит следующую информацию:

```
Имя: Wolfgang
Пользовательское имя: w.a.mozart
Пароль: composer
```

Ключевое слово prototype

Использование ключевого слова `prototype` позволяет добиться существенной экономии оперативной памяти. Каждый экземпляр класса `User` будет содержать три свойства и один метод. Поэтому, если в памяти содержится тысяча таких объектов, метод `showUser` также будет растиражирован тысячу раз. Но, поскольку в каждом экземпляре присутствует один и тот же метод, можно предписать новому объекту ссылаться на единственный экземпляр этого метода и не создавать его копию. Итак, вместо использования в конструкторе класса строки кода:

```
this.showUser = function()
```

можно воспользоваться следующей строкой:

```
User.prototype.showUser = function()
```

Код обновленного конструктора показан в примере 15.7.

Пример 15.7. Объявление класса с использованием для метода ключевого слова prototype

```
<script>
  function User(forename, username, password)
  {
    this.forename = forename
    this.username = username
    this.password = password

    User.prototype.showUser = function()
    {
      document.write("Имя: " + this.forename + "<br>")
      document.write("Пользовательское имя: " + this.username + "<br>")
      document.write("Пароль: " + this.password + "<br>")
    }
  }
</script>
```

Этот код работает благодаря тому, что у всех функций имеется свойство по имени `prototype`, разработанное для хранения свойств и методов, не тиражируемых в каж-

дом объекте, создаваемом на основе класса. Вместо этого они передаются объектам данного класса по ссылке.

Это означает, что свойства или методы `prototype` могут быть добавлены в любое время и будут унаследованы всеми объектами (даже теми, которые уже были созданы), что можно проиллюстрировать следующими инструкциями:

```
User.prototype.greeting = "Привет" document.write(details.greeting)
```

Первая инструкция добавляет к классу `User` прототипное свойство `prototype.greeting`, имеющее значение `Привет`. Во второй строке уже созданный объект `details` вполне корректно отображает это новое свойство.

Можно также добавлять к классу методы или вносить в них изменения, как показано в следующих инструкциях:

```
User.prototype.showUser = function()
{
  document.write("Имя "           + this.forename +
                 " Пользователь " + this.username +
                 " Пароль "       + this.password)
}

details.showUser()
```

Эти строки можно поместить в свой сценарий, в инструкцию условия (например, в `if`), чтобы они запускались только в том случае, когда действия пользователя наталкивают на принятие решения о применении другого метода `showUser`. После запуска этих строк кода даже для уже созданного объекта `details` при всех последующих вызовах метода `details.showUser` будет запускаться новая версия, а старое определение `showUser` будет стерто.

Статические методы и свойства

При изучении объектов PHP вы узнали, что у классов могут быть статические свойства и методы, а также свойства и методы, связанные с конкретным экземпляром класса. JavaScript также поддерживает статические свойства и методы, которые легко и просто могут сохраняться в принадлежащие классу прототипы и извлекаться из них. Следующие инструкции устанавливают в класс `User` и считывают из него статическую строку:

```
User.prototype.greeting = "Привет"
document.write(User.prototype.greeting)
```

Расширение объектов JavaScript

Ключевое слово `prototype` позволяет даже добавлять функциональные возможности встроенным объектам. Предположим, что нужно добавить возможность замены всех пробелов в строке неразрывными пробелами, чтобы избежать переноса ее

части на новую строку. Это можно сделать добавлением к имеющемуся в JavaScript определению исходного объекта `String` прототипного метода:

```
String.prototype.nbsp = function()
{
  return this.replace(/ /g, ' ')
}
```

В коде этого метода для поиска всех одиночных пробелов и замены их строкой ` ` используются метод `replace` и регулярное выражение.

Если вы еще не знакомы с регулярными выражениями, следует пояснить, что они являются очень удобными средствами извлечения информации из строк или манипулирования строками и весьма подробно рассматриваются в главе 16. А пока вы можете скопировать и вставить предыдущие примеры, и они будут работать согласно описаниям, иллюстрируя возможности расширения JavaScript-объектов `String`.

Если после запуска этого кода будет введена следующая команда:

```
document.write("Шустрая бурая лиса".nbsp())
```

то в результате ее работы будет выведена следующая строка: `Шустрая бурая лиса`. Посмотрите также на метод, приведенный далее. Его можно добавить для удаления всех пробелов, с которых начинается и которыми заканчивается строка (в нем опять используется регулярное выражение):

```
String.prototype.trim = function()
{
  return this.replace(/^\s+|\s+$/g, '')
}
```

Если выдать следующую инструкцию, то на выходе будет получена строка `Пожалуйста, избавьте меня от лишних пробелов` (то есть из нее будут удалены все начальные и замыкающие пробелы):

```
document.write(" Пожалуйста, избавьте меня от лишних пробелов ". trim())
```

Если разбить выражение на составные части, то два символа `/` помечают начало и конец выражения, а завершающий символ `g` задает глобальный поиск. Внутри выражения его часть `^\s+` задает поиск одного или нескольких пробельных символов применительно к началу строки, в которой ведется поиск, а его часть `\s+$` задает поиск одного или нескольких пробельных символов применительно к концу строки, в которой ведется поиск. Расположенный в середине символ `|` служит разделителем альтернативных вариантов регулярного выражения.

В результате при соответствии любого из этих выражений соответствующая часть заменяется пустой строкой, возвращая тем самым усеченную версию строки без лидирующих и замыкающих пустых пространств.

Среди программистов возникают споры, насколько хороша или плоха практика расширения объектов. Одни говорят, что при последующем расширении объекта, при котором добавленная вами функциональная возможность предлагается вполне официально, ее реализация может отличаться от вашей, или же совершать нечто абсолютно иное по сравнению с вашим расширением, что в последующем может создать конфликтную ситуацию. Другие же, включая автора JavaScript, говорят, что практика создания расширений вполне приемлема. Я склонен согласиться с последними, но в коде, предназначенном для промышленного применения, лучше выбирать такие имена расширений, использование которых в официальных реализациях будет наименее вероятным. К примеру, расширение trim можно переименовать в mytrim, и поддерживающий его код может с большей долей безопасности быть написан следующим образом:

```
String.prototype.mytrim = function()
{
  return this.replace(/^\s+|\s+$/g, '')
}
```

Массивы в JavaScript

Работа с массивами в JavaScript очень напоминает работу с ними в PHP, хотя синтаксис имеет несколько иной вид. Тем не менее с учетом уже приобретенных знаний о массивах освоить материал этого раздела будет относительно несложно.

Числовые массивы

Чтобы создать новый массив, нужно воспользоваться следующим синтаксисом:

```
arrayname = new Array()
```

или же его более краткой формой:

```
arrayname = []
```

Присваивание значений элементам массива

В PHP можно было добавить к массиву новый элемент простым присваиванием ему значения, без указания смещения элемента относительно начала массива:

```
$arrayname[] = "Элемент 1";
$arrayname[] = "Элемент 2";
```

В JavaScript для этих же целей используется метод `push`:

```
arrayname.push("Элемент 1")
arrayname.push("Элемент 2")
```

Он позволяет добавлять к массиву элементы, не отслеживая их количество. Когда потребуется узнать, сколько элементов содержится в массиве, можно будет воспользоваться свойством `length`:

```
document.write(arrayname.length)
```

Если нужно будет проконтролировать размещение элементов, расставляя их по конкретным местам, можно воспользоваться другим синтаксисом:

```
arrayname[0] = "Элемент 1"
arrayname[1] = "Элемент 2"
```

В примере 15.8 показан простой сценарий, в котором создается массив, в него загружается несколько элементов, после чего эти элементы отображаются на экране.

Пример 15.8. Создание, построение и вывод массива на экран

```
<script>
  numbers = []
  numbers.push("Один")
  numbers.push("Два")
  numbers.push("Три")

  for (j = 0 ; j < numbers.length ; ++j)
    document.write("Элемент " + j + " = " + numbers[j] + "<br>")
</script>
```

Этот сценарий выводит следующую информацию:

```
Элемент 0 = Один
Элемент 1 = Два
Элемент 2 = Три
```

Присваивание с использованием ключевого слова Array

С помощью ключевого слова `Array` можно также создать массив с несколькими исходными элементами:

```
numbers = Array("Один", "Два", "Три")
```

После этого ничто не помешает добавить к данному массиву дополнительные элементы.

Теперь в вашем распоряжении есть несколько способов добавления элементов к массиву и один способ ссылки на них, но JavaScript предлагает куда более обширный арсенал способов, к рассмотрению которых мы скоро перейдем. Но сначала рассмотрим еще один тип массива.

Ассоциативные массивы

К *ассоциативным* относятся такие массивы, в которых ссылки на элементы осуществляются по именам, а не по целочисленному смещению. Но в JavaScript подоб-

ные структуры не поддерживаются. Нам придется добиваться нужных результатов путем создания объекта с аналогично работающими свойствами.

Чтобы создать «ассоциативный массив», нужно определить блок элементов, заключенный в фигурные скобки. Для каждого элемента слева от двоеточия (`:`) указывается его ключ, а справа — содержимое. В примере 15.9 показано, как можно создать ассоциативный массив для хранения данных о товаре в разделе мячей интернет-магазина спортивного инвентаря.

Пример 15.9. Создание и отображение ассоциативного массива

```
<script>
  balls = {"гольф":     "Мячи для гольфа, 6",
           "теннис":    "Мячи для тенниса, 3",
           "футбол":    "Футбольный мяч, 1",
           "пинг-понг": "Мячи для пинг-понга. 12 шт."}

  for (ball in balls)
    document.write(ball + " = " + balls[ball] + "<br>")
</script>
```

Для проверки факта создания и заполнения массива я воспользовался еще одной разновидностью цикла `for`, в которой применяется ключевое слово `in`. В этом цикле создается новая переменная, которая задействуется только внутри массива (в данном примере — `ball`), и вызывается последовательный перебор всех элементов массива, указанных справа от ключевого слова `in` (в данном примере — `balls`). Цикл обрабатывает каждый элемент массива `balls`, помещая значение ключа в переменную `ball`.

Используя значение ключа, сохраненное в переменной `ball`, можно также получить значение текущего элемента массива `balls`. Результат вызова сценария этого примера в браузере будет иметь следующий вид:

```
гольф = Мячи для гольфа, 6
теннис = Мячи для тенниса, 3
футбол = Футбольный мяч, 1
пинг-понг = Мячи для пинг-понга, 12 шт.
```

Чтобы получить значение конкретного элемента ассоциативного массива, нужно в явном виде указать его ключ (в данном случае будет выведено значение `Футбольный мяч, 1`):

```
document.write(balls['футбол'])
```

Многомерные массивы

В JavaScript для создания многомерного массива нужно просто поместить массивы внутрь других массивов. Например, чтобы создать массив, содержащий сведения о двумерной шахматной доске (8 × 8 клеток), можно воспользоваться кодом примера 15.10.

Пример 15.10. Создание многомерного числового массива

```
<script>
  checkerboard = Array(
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array('o', ' ', 'o', ' ', 'o', ' ', 'o', ' '),
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array('O', ' ', 'O', ' ', 'O', ' ', 'O', ' '),
    Array(' ', 'O', ' ', 'O', ' ', 'O', ' ', 'O'),
    Array('O', ' ', 'O', ' ', 'O', ' ', 'O', ' '))

  document.write("<pre>")

  for (j = 0 ; j < 8 ; ++j)
  {
    for (k = 0 ; k < 8 ; ++k)
      document.write(checkerboard[j][k] + " ")

    document.write("<br>")
  }

  document.write("</pre>")
</script>
```

В данном примере буквами нижнего регистра обозначены черные, а буквами верхнего регистра — белые фигуры. Два цикла `for`, один из которых является вложенным, осуществляют последовательный перебор элементов массива и отображают его содержимое.

Внешний цикл содержит две инструкции, поэтому они заключены в фигурные скобки. Внутренний цикл обрабатывает каждую клетку в горизонтали, выводя символ, находящийся в позиции `[j][k]`, за которым следует пробел (чтобы придать выводимой информации квадратную конфигурацию). В этом цикле содержится всего одна инструкция, поэтому заключать ее в фигурные скобки не имеет смысла. Теги `<pre>` и `</pre>` обеспечивают корректный вывод информации:

```
  o   o   o   o
o   o   o   o
  o   o   o   o

O   O   O   O
  O   O   O   O
O   O   O   O
```

Можно также получить непосредственный доступ к любому элементу данного массива, применив для этого квадратные скобки:

```
document.write(checkerboard[7][2])
```

Эта инструкция выводит букву `O` верхнего регистра, то есть содержимое восьмой сверху и третьей справа клетки — напоминаю, что индексация элементов в массиве начинается с нуля, а не с единицы.

Методы массивов

Реализовать возможности, предоставленные массивами, помогают имеющиеся в JavaScript готовые к использованию методы для работы с ними и с содержащимися в них данными. Рассмотрим подборку, состоящую из наиболее востребованных методов.

Метод some

Когда нужно узнать, соответствует ли хотя бы один элемент определенному критерию, можно воспользоваться методом `some`, который позволит протестировать все элементы и автоматически остановиться, возвратив требуемое значение при первом же найденном соответствии. Это избавит вас от необходимости написания собственного кода для выполнения подобного поиска:

```
function isBiggerThan10(element, index, array)
{
  return element > 10
}

result = [2, 5, 8, 1, 4].some(isBiggerThan10);  // результатом будет false
result = [12, 5, 8, 1, 4].some(isBiggerThan10); // результатом будет true
```

Метод indexOf

Для определения факта нахождения элемента в массиве можно воспользоваться в отношении массива методом `indexOf`, которым будет возвращено смещение обнаруженного элемента (начиная с нуля) или значение `-1`, если элемент не будет найден. Например, при выполнении следующего кода переменной `offset` будет присвоено значение 2:

```
animals = ['кот', собака', 'корова', 'лошадь', 'слон']
offset = animals.indexOf('корова')
```

Метод concat

Метод `concat` объединяет два массива или ряд значений в массив. Например, следующий код выведет `Банан,Виноград,Морковь,Капуста`:

```
fruit = ["Банан", "Виноград"]
veg = ["Морковь", "Капуста"]

document.write(fruit.concat(veg))
```

В качестве аргументов можно указать несколько массивов, тогда метод `concat` добавит все их элементы в порядке указания массивов.

А вот еще один способ использования метода concat, где с массивом pets объединяются простые значения и на экран выводится строка Кошка,Собака,Рыба,Кролик,Хомяк:

```
pets = ["Кошка", "Собака", "Рыба"]
more_pets = pets.concat("Кролик", "Хомяк")

document.write(more_pets)
```

Метод forEach (для браузеров не из семейства IE)

Используемый в JavaScript метод forEach является еще одним способом получения функциональных возможностей, аналогичных тем, которые предоставляются ключевым словом PHP forEach. Воспользоваться этим методом можно, передав ему имя функции, которая будет вызвана для каждого элемента массива. Как это делается, показано в примере 15.11.

Пример 15.11. Использование метода forEach

```
<script>
  pets = ["Кошка", "Собака", "Кролик", "Хомяк"]
  pets.forEach(output)

  function output(element, index, array)
  {
    document.write("Элемент с индексом " + index + " содержит значение " +
      element + "<br>")
  }
</script>
```

В данном случае функция, передаваемая методу forEach, называется output. Она воспринимает три параметра: элемент, его индекс и массив. Как они используются, зависит от потребностей вашей функции. В данном примере они просто отображают значения индекса и элемента с помощью метода document.write.

После того как массив будет заполнен, можно вызвать рассматриваемый метод:

```
pets.forEach(output)
```

На выходе будет получена следующая информация:

```
Элемент с индексом 0 содержит значение Кошка
Элемент с индексом 1 содержит значение Собака
Элемент с индексом 2 содержит значение Кролик
Элемент с индексом 3 содержит значение Хомяк
```

Метод join

Метод join позволяет превратить все значения массива в строки, а затем объединить их в одну большую строку, расставляя между значениями необязательные разделители. В примере 15.12 показаны три способа использования этого метода.

Пример 15.12. Использование метода join

```
<script>
  pets = ["Кошка", "Собака", "Кролик", "Хомяк"]

  document.write(pets.join()       + "<br>")
  document.write(pets.join(' ')   + "<br>")
  document.write(pets.join(' : ') + "<br>")
</script>
```

Если не указывать параметр, метод `join` использует в качестве разделителя элементов запятую, в противном случае между элементами вставляется переданная методу `join` строка. Код примера 15.12 выводит следующую информацию:

```
Кошка,Собака,Кролик,Хомяк
Кошка Собака Кролик Хомяк
Кошка : Собака : Кролик : Хомяк
```

Методы push и pop

Применение метода `push` для вставки значения в массив уже было рассмотрено. Противоположным ему по действию является метод `pop`. Он удаляет последний вставленный элемент из массива и возвращает значение этого элемента. Порядок его использования показан в примере 15.13.

Пример 15.13. Использование методов push и pop

```
<script>
  sports = ["Футбол", "Теннис", "Бейсбол"]
  document.write("Изначально = "         + sports + "<br>")

  sports.push("Hockey")
  document.write("После вставки = "      + sports + "<br>")

  removed = sports.pop()
  document.write("После удаления = "     + sports + "<br>")
  document.write("Удаленный элемент = " + removed + "<br>")
</script>
```

Три основные инструкции этого сценария выделены полужирным шрифтом. Сначала в сценарии создается массив по имени `sports`, содержащий три элемента, затем в него вставляется четвертый элемент, после чего сценарий удаляет этот элемент. В процессе этих действий с помощью метода `document.write` отображаются разные значения массива. Сценарий выводит следующую информацию:

```
Изначально = Футбол,Теннис,Бейсбол
После вставки = Футбол,Теннис,Бейсбол,Хоккей
После удаления = Футбол,Теннис,Бейсбол
Удаленный элемент = Хоккей
```

Методы `push` и `pop` применяются в тех случаях, когда нужно отвлечься от каких-нибудь действий на другие, а затем вернуться к прежним действиям. Предположим,

к примеру, что вам нужно отложить некоторые действия на потом и заняться чем-нибудь более важным на данный момент. Такое часто случается в реальной жизни, когда просматриваются списки предстоящих дел, поэтому давайте воспроизведем это в программном коде в списке из пяти дел применительно к задачам 2 и 5, которым отдается приоритет.

Пример 15.14. Использование методов push и pop внутри цикла и за его пределами

```
<script>
  numbers = []

  for (j=1 ; j<6 ; ++j)
  {
    if (j == 2 || j == 5)
    {
      document.write("Выполнение намеченной задачи #" + j + "<br>")
    }
    else
    {
      document.write("Откладывание намеченной задачи #" + j + "
                     на потом<br>")
      numbers.push(j)
    }
  }

  document.write("<br>Завершено выполнение приоритетных задач.")
  document.write("<br>Начало выполнения отложенных задач в порядке,
                  обратном их следованию.<br><br>")

  document.write("Выполнение намеченной задачи #" + numbers.pop() + "<br>")
  document.write("Выполнение намеченной задачи #" + numbers.pop() + "<br>")
  document.write("Выполнение намеченной задачи #" + numbers.pop() + "<br>")
</script>
```

Код этого примера выведет следующую информацию:

```
Откладывание намеченной задачи #1 на потом
Выполнение намеченной задачи # 2
Откладывание намеченной задачи #3 на потом
Откладывание намеченной задачи #4 на потом
Выполнение намеченной задачи # 5

Завершено выполнение приоритетных задач.
Начало выполнения отложенных задач в порядке, обратном их следованию.

Выполнение намеченной задачи #4
Выполнение намеченной задачи #3
Выполнение намеченной задачи #1
```

Использование метода reverse

Метод `reverse` осуществляет простую перестановку элементов массива в обратном порядке. Его действие показано в примере 15.15.

Пример 15.15. Использование метода reverse

```
<script>
  sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
  sports.reverse()
  document.write(sports)
</script>
```

Исходный массив подвергается изменению, и сценарий выводит следующую информацию:

```
Хоккей,Бейсбол,Теннис,Футбол
```

Метод sort

Метод `sort` позволяет расставить все элементы массива в алфавитном или в каком-нибудь другом порядке в зависимости от применяемых параметров. В примере 15.16 показаны четыре типа сортировки.

Пример 15.16. Использование метода sort

```
<script>
  // Сортировка по алфавиту
  sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
  sports.sort()
  document.write(sports + "<br>")

  // Сортировка по алфавиту в обратном порядке
  sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
  sports.sort().reverse()
  document.write(sports + "<br>")

  // Сортировка чисел по возрастанию
  numbers = [7, 23, 6, 74]
  numbers.sort(function(a,b){return a - b})
  document.write(numbers + "<br>")

  // Сортировка чисел по убыванию
  numbers = [7, 23, 6, 74]
  numbers.sort(function(a,b){return b - a})
  document.write(numbers + "<br>")
</script>
```

В первом из четырех блоков этого примера применяется *сортировка по алфавиту*, во втором — возвращение к исходному виду, а затем метод `reverse`, чтобы получить *сортировку по алфавиту в обратном порядке*.

Третий и четвертый блоки усложнены использованием функции для сравнения взаимоотношений между `a` и `b`. У нее отсутствует имя, поскольку она используется только при сортировке. Функция по имени `function`, которая применяется для создания безымянных функций, уже встречалась при определении метода класса (метода `showUser`).

Здесь `function` создает безымянную функцию, отвечающую запросам метода `sort`. Если функция возвращает значение больше нуля, сортировка предполагает, что `b` ставится перед `a`. Если функция возвращает значение меньше нуля, сортировка предполагает, что `a` ставится перед `b`. Сортировка запускает эту функцию применительно ко всем значениям массива для определения порядка их следования. (Разумеется, если `a` и `b` имеют одинаковое значение, функция возвращает нуль, и порядок их следования совершенно не важен.)

За счет манипуляции возвращаемыми значениями (`a – b` или `b – a`) в третьем и четвертом блоках примера 15.16 осуществляется выбор между сортировкой чисел по возрастанию и по убыванию.

На этом я заканчиваю введение в JavaScript. Теперь у вас должно сложиться представление о трех основных технологиях, рассматриваемых в данной книге. В следующей главе будут описаны некоторые современные технические приемы, основанные на применении всех этих технологий, в частности проверка соответствия шаблонам и проверка допустимости введенных значений.

Вопросы

1. Обладают ли имена функций и переменных в JavaScript чувствительностью к регистру используемых в них букв?
2. Как создать функцию, которая воспринимает и обрабатывает неограниченное количество параметров?
3. Назовите способ возвращения из функции сразу нескольких значений.
4. Какое ключевое слово для ссылки на текущий объект используется при определении класса?
5. Должны ли все методы класса определяться внутри определения самого класса?
6. Какое ключевое слово применяется для создания объекта?
7. Как обеспечить доступность свойства или метода всем объектам класса без его тиражирования внутри объекта?
8. Как создать многомерный массив?
9. Какой синтаксис используется для создания ассоциативного массива?
10. Создайте инструкцию для сортировки массива чисел в убывающем порядке.

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

16 Проверка данных и обработка ошибок в JavaScript и PHP

После приобретения основательных знаний по программированию на PHP и JavaScript настало время объединить эти две технологии воедино для создания максимально удобных для пользователей веб-форм.

PHP будет использоваться для создания форм, а JavaScript — для проверки приемлемости и полноты данных на стороне клиента, насколько это возможно до их отправки на сервер. После чего окончательная проверка приемлемости введенных данных будет выполняться программой PHP, которая при необходимости снова выведет форму, чтобы пользователь мог внести в нее изменения.

В процессе изложения данной главы будут рассмотрены проверка данных и применение регулярных выражений как в JavaScript, так и в PHP.

Проверка данных, введенных пользователем, средствами JavaScript

Проверка данных средствами JavaScript должна рассматриваться в качестве помощи пользователям, а не сайтам, поскольку, как я уже неоднократно подчеркивал, нельзя доверять абсолютно ничему, что отправлено на ваш сервер, даже если предположить, что полученные данные проверены с помощью JavaScript. Дело в том, что взломщики могут без особых усилий создать имитацию ваших веб-форм и отправить любые нужные им данные.

Еще одна причина, не позволяющая полагаться на JavaScript при проверке введенных данных, заключается в том, что некоторые пользователи отключают JavaScript или используют браузеры, не поддерживающие этот язык.

Поэтому лучшее, что можно сделать при проверке данных средствами JavaScript, — выполнить проверку информационного наполнения тех полей, которые не должны оставаться пустыми, обеспечить приведение адресов электронной почты в надлежащий формат и гарантировать то, что введенные значения находятся в пределах ожидаемых границ.

Документ validate.html (часть первая)

Рассмотрим стандартную регистрационную форму, используемую большинством сайтов, на которых работать можно только зарегистрированным пользователям. В форме будут запрашиваться *имя*, *фамилия*, *пользовательское имя*, *пароль*, *возраст* и *адрес электронной почты*. В примере 16.1 показан шаблон, который может применяться для этой формы.

Пример 16.1. Форма с проверкой данных средствами JavaScript (часть первая)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Пример формы</title>
    <style>
      .signup {
         border: 1px solid #999999;
         font: normal 14px helvetica;
         color:#444444;
      }
    </style>
    <script>
      function validate(form) {
        fail = validateForename(form.forename.value)
        fail += validateSurname(form.surname.value)
        fail += validateUsername(form.username.value)
        fail += validatePassword(form.password.value)
        fail += validateAge(form.age.value)
        fail += validateEmail(form.email.value)

        if (fail == "") return true
        else { alert(fail); return false }
      }
    </script>
  </head>
  <body>
    <table class="signup" border="0" cellpadding="2"
           cellspacing="5" bgcolor="#eeeeee">
      <th colspan="2" align="center">Регистрационная форма</th>
      <form method="post" action="adduser.php"
            onSubmit="return validate(this)">
        <tr><td>Имя</td><td><input type="text" maxlength="32"
           name="forename"></td></tr>
        <tr><td>Фамилия</td><td><input type="text" maxlength="32"
```

```
          name="surname"></td></tr>
        <tr><td>Пользовательское имя</td>
        <td><input type="text" maxlength="16"
          name="username"></td></tr>
        <tr><td>Пароль</td>
        <td><input type="text" maxlength="12"
          name="password"></td></tr>
        <tr><td>Возраст</td>
       <td><input type="text" maxlength="3"
          name="age"></td></tr>
       <tr><td>Электронный адрес</td>
       <td><input type="text" maxlength="64"
          name="email"></td></tr>
       <tr><td colspan="2" align="center">
         <input type="submit" value="Зарегистрироваться"></td></tr>
      </form>
    </table>
  </body>
</html>
```

В данном виде эта форма будет только отображаться, но не сможет заниматься самопроверкой, поскольку к ней еще не добавлены основные проверочные функции. Но несмотря на это, если набрать данный код, сохранить его в файле `validate.html`, а затем вызвать файл в браузере, мы получим результат, показанный на рис. 16.1.

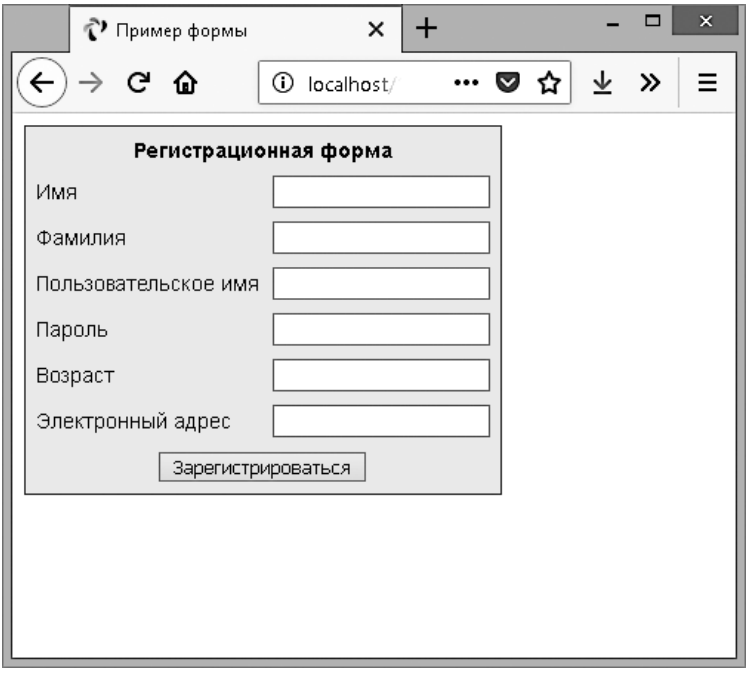

Рис. 16.1. Форма, выведенная кодом из примера 16.1

Рассмотрим, из чего состоит этот документ. В первых нескольких строках осуществляется настройка документа и используется небольшой фрагмент кода CSS, предназначенный для улучшения внешнего вида формы. Затем следует выделенная полужирным шрифтом часть документа, относящаяся к JavaScript.

В теги `<script>` и `</script>` заключена всего одна функция по имени `validate`, которая, в свою очередь, вызывает шесть других функций, проверяющих каждое из имеющихся в форме полей. Все они вскоре будут рассмотрены. А сейчас я ограничусь объяснением того, что они возвращают либо пустую строку, если поле проходит проверку, либо сообщение об ошибке, если оно эту проверку не проходит. При наличии любых ошибок сообщения о них выводятся в окне предупреждения, появляющемся благодаря последней строке сценария.

Если поле проходит проверку, то проводившая ее функция возвращает значение `true`, а если не проходит — `false`. Значения, возвращаемые функцией `validate`, учитываются при отправке данных формы: если она возвращает `false`, данные не отправляются. При этом пользователь получает возможность закрыть появившееся окно предупреждения и внести изменения в данные. Если будет возвращено значение `true`, значит ошибок в полях формы не найдено и форму можно отправлять на сервер.

Во второй части этого примера показан код HTML для формы, где каждое поле и его имя помещены в отдельную строку таблицы. В этом HTML нет ничего сложного, за исключением инструкции `onSubmit="return validate(this)"`, помещенной в открывающий тег `<form>`. Использование атрибута `onSubmit` позволяет при отправке формы вызвать избранную вами функцию. Эта функция может выполнить проверку и вернуть значение либо `true`, либо `false`, для того чтобы известить о том, разрешена или нет отправка формы.

Параметр `this` указывает на текущий объект (то есть на данную форму). Он передается только что рассмотренной функции `validate`, которая получает этот параметр в виде объекта `form`.

Как видите, внутри HTML-формы JavaScript используется только для того, чтобы вызвать инструкцию `return`, помещенную в атрибут `onSubmit`. Браузеры, у которых JavaScript отключен или не поддерживается, просто проигнорируют атрибут `onSubmit` и беспрепятственно отобразят HTML.

Документ validate.html (часть вторая)

Теперь обратимся к коду примера 16.2, содержащему набор из шести функций, осуществляющих проверку полей формы. Я предлагаю набрать весь код этой второй части и сохранить его в разделе `<script>...</script>` примера 16.1, который уже сохранен в файле `validate.html`.

Пример 16.2. Форма, проверяемая средствами JavaScript (вторая часть)

```
function validateForename(field)
{
  return (field == "") ? "Не введено имя.\n" : ""
}

function validateSurname(field)
{
  return (field == "") ? "Не введена фамилия.\n" : ""
}

function validateUsername(field)
{
  if (field == "") return "Не введено имя пользователя.\n"
  else if (field.length < 5)
    return "В имени пользователя должно быть не менее 5 символов.\n"
  else if (/[^a-zA-Z0-9_-]/.test(field))
    return "В имени пользователя разрешены только a-z, A-Z, 0-9, - и _.\n"
  return ""
}

function validatePassword(field)
{
  if (field == "") return "Не введен пароль.\n"
  else if (field.length < 6)
    return "В пароле должно быть не менее 6 символов.\n"
  else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) ||
           !/[0-9]/.test(field))
    return "Пароль требует 1 символ из каждого набора a-z, A-Z и 0-9.\n"
  return ""
}

function validateAge(field)
{
  if (field == "" || isNaN(field)) return "Не введен возраст.\n"
  else if (field < 18 || field > 110)
    return "Возраст должен быть между 18 и 110.\n"
  return ""
}

function validateEmail(field)
{
  if (field == "") return "Не введен адрес электронной почты.\n"
  else if (!((field.indexOf(".") > 0) &&
             (field.indexOf("@") > 0)) ||
            /[^a-zA-Z0-9.@_-]/.test(field))
    return "Электронный адрес имеет неверный формат.\n"
  return ""
}
</script>
</body>
</html>
```

Чтобы понять, как работает проверка, рассмотрим по очереди все эти функции, начиная с `validateForename`.

Проверка имени

Предельно лаконичная функция `validateForename` воспринимает параметр `field`, являющийся значением имени (`forename`), переданным ей функцией `validate`. Если это значение является пустой строкой, возвращается сообщение об ошибке, если нет, то возвращается пустая строка, свидетельствующая о том, что ошибка не обнаружена.

Если пользователь введет в это поле пробелы, то они будут приняты функцией `validateForename`, хотя в качестве имени они не годятся. Этот просчет можно исправить, добавив еще одну инструкцию, удаляющую из поля пустые пространства перед его проверкой на незаполненность, затем воспользоваться регулярным выражением, чтобы убедиться в том, что в поле находится еще что-нибудь, кроме пробелов, или — как это сделано в данном случае — позволить пользователю допустить эту ошибку и «отловить» ее на сервере.

Проверка фамилии

Код функции `validateSurname` похож на код функции `validateForename`, он также возвращает сообщение об ошибке, если в качестве фамилии (`surname`) была предоставлена пустая строка. Я решил не накладывать ограничений на символы обоих полей, чтобы пользователь мог вводить символы, не входящие в английский алфавит, имеющие дополнительные знаки и т. д.

Проверка имени пользователя

Код функции `validateUsername` немного интереснее, поскольку выполняет более сложную работу. Он должен разрешить использование только тех символов, которые входят в набор a–z, A–Z, 0–9, _ и –, и гарантировать, что имена пользователей состоят не менее чем из пяти символов.

Код структуры `if...else` начинается с возвращения сообщения об ошибке в том случае, если поле не было заполнено. Если значение поля не является пустой строкой, но состоит менее чем из пяти символов, то возвращается другое сообщение об ошибке.

Затем вызывается JavaScript-функция `test`, которая сравнивает регулярное выражение (соответствующее любому символу, *не* входящему в перечень разрешенных) с содержимым поля (см. раздел «Регулярные выражения» данной главы). Встретив хотя бы один недопустимый символ, функция `test` возвращает `true`, в результате чего функция `validateUsername` возвращает сообщение об ошибке.

Проверка пароля

Такая же технология используется и в функции `validatePassword`. Сначала функция проверяет поле на пустоту, возвращая сообщение об ошибке при незаполненном поле. Затем сообщение об ошибке возвращается в том случае, если пароль короче шести символов.

Одно из требований, предъявляемых к паролям, заключается в том, что в них должно быть хотя бы по одному символу в нижнем и в верхнем регистре, а также хотя бы одна цифра, поэтому функция `test` вызывается три раза, по одному разу на каждую из этих проверок. Если при любом из таких вызовов будет возвращено значение `false`, это будет говорить о том, что одно из условий не выполнено, поэтому будет возвращено сообщение об ошибке. В противном случае будет возвращена пустая строка, свидетельствующая о том, что с паролем все в порядке.

Проверка возраста

Функция `validateAge` возвращает сообщение об ошибке, если значение поля не является числом (что определяется вызовом функции `isNaN`) либо введенный возраст меньше 18 или больше 110 лет. У ваших приложений могут быть иные требования к возрастной категории или вообще не быть никаких требований. При успешной проверке также будет возвращена пустая строка.

Проверка адреса электронной почты

И последняя, наиболее сложная проверка — адреса электронной почты — выполняется с помощью функции `validateEmail`. После проверки на существование каких-нибудь введенных данных и возвращения сообщения об ошибке при отсутствии таковых функция дважды вызывает JavaScript-функцию `indexOf`. При первом вызове проверяется наличие точки (`.`) где-нибудь в поле после первого символа, а при втором — присутствие символа `@`, также где-нибудь после первого символа.

Если будут пройдены эти две проверки, вызывается функция `test`, чтобы проверить поле на наличие недопустимых символов. Если любая из этих проверок не будет пройдена, то возвращается сообщение об ошибке. Допустимыми в адресе электронной почты считаются буквы в нижнем и верхнем регистрах, символы подчеркивания, тире, точки и символ `@`. Все они перечислены в регулярном выражении, передаваемом методу `test`. Если не будет найдено ни одной ошибки, возвращается пустая строка, свидетельствующая об успешно пройденной проверке. В последней строке примера сценарий и документ закрываются.

На рис. 16.2 показан результат нажатия кнопки Зарегистрироваться без заполнения полей.

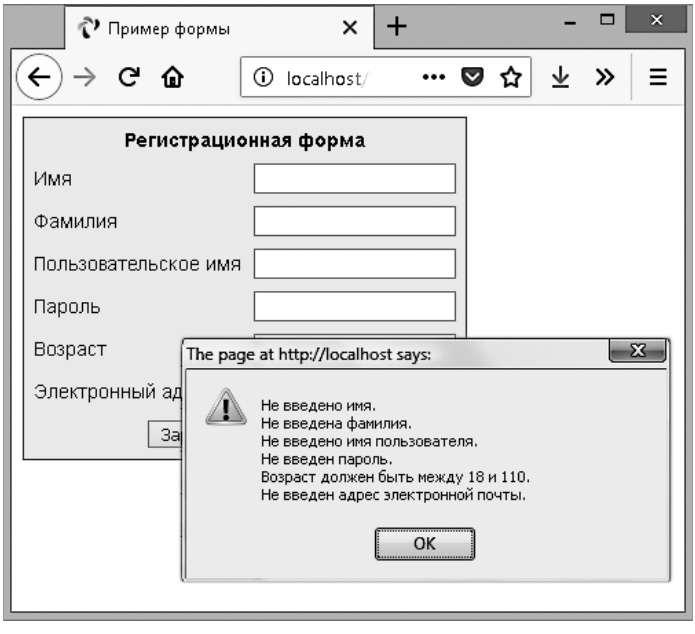

Рис. 16.2. Работа JavaScript-формы с проверкой данных

Использование отдельного файла JavaScript

Конечно, благодаря универсальности своей конструкции и применимости ко многим типам потенциально востребуемых проверок эти шесть функций становятся идеальными кандидатами для выделения в отдельный файл. Этот файл, к примеру, можно назвать `validate_functions.js` и включить его сразу же после начального блока сценария в пример 16.1, используя следующую инструкцию:

```
<script src="validate_functions.js"></script>
```

Регулярные выражения

Более пристально рассмотрим шаблоны соответствия, созданные нами благодаря использованию *регулярных выражений*, которые поддерживаются как в JavaScript, так и в PHP. Они позволяют выстроить внутри одного выражения более мощные алгоритмы соответствия шаблонам.

Соответствие, закладываемое в метасимволы

Любое регулярное выражение должно быть заключено в слеши (/). Конкретные символы, находящиеся внутри этих слешей, называются *метасимволами* и имеют специальное предназначение. Например, звездочка (*) имеет аналогичное

(но не вполне такое же) значение, как и звездочки, уже встречавшиеся вам в оболочке или в командной строке Windows. Звездочка означает следующее: «Текст, подвергаемый сравнению, может содержать любое количество указанного перед ней символа или не содержать его вообще».

К примеру, вы ищете имя Le Guin и знаете, что оно может быть написано как с пробелом, так и без него. Из-за не вполне обычной разметки текста (кто-нибудь, например, мог вставить лишние пробелы, чтобы выровнять строки по правому краю) нужно вести поиск в следующей строке:

```
The difficulty of classifying Le Guin's works
```

Иначе говоря, шаблон должен соответствовать строке `LeGuin`, а также отдельно строкам `Le` и `Guin`, разделенным любым количеством пробелов. Решением может стать установка после пробела звездочки:

```
/Le *Guin/
```

В строке кроме имени `Le Guin` присутствует множество других символов, но этот шаблон все равно будет работать. Поскольку регулярное выражение соответствует какой-то части строки, проверочная функция вернет истинное значение. А если нужно узнать, что в строке не содержится ничего другого, кроме `Le Guin`? Как в этом убедиться, будет показано чуть позже.

Предположим, что известно о непременном наличии хотя бы одного пробела. В таком случае можно воспользоваться знаком «плюс» (`+`), поскольку этот метасимвол требует присутствия хотя бы одного из предшествующих ему символов:

```
/Le +Guin/
```

Нестрогое символьное соответствие

Одним из самых полезных метасимволов является точка (`.`), поскольку она может соответствовать любому символу, за исключением символа новой строки. Предположим, что выполняется поиск HTML-тегов, которые начинаются с символа `<` и заканчиваются символом `>`. Проще всего найти тег с помощью следующего регулярного выражения:

```
/<.*>/
```

Точка соответствует любому символу, а звездочка (`*`) расширяет действие точки до соответствия нулевому или любому другому количеству символов, что означает: «Соответствует всему, что заключено между символами `<` и `>`, даже если там ничего нет».

Этот шаблон будет соответствовать строкам `<>`, `<em>`, `<br>` и т. д. Но если не требуется, чтобы он соответствовал отсутствию символов `<>`, нужно вместо символа * использовать символ `+`:

```
/<.+>/
```

Знак «плюс» расширяет действие точки до соответствия одному или нескольким символам, что означает: «Соответствует всему, что находится между символами < и >, пока между ними есть хотя бы один символ».

Этот шаблон будет соответствовать и , <h1> и </h1> и тегам с атрибутами, например:

```
<a href="www.mozilla.org">
```

К сожалению, знак «плюс» расширит соответствие вплоть до последнего символа > в строке, поэтому соответствовать шаблону будет и такая строка:

```
<h1><b>Введение</b></h1>
```

А в ней содержится больше одного тега! Чуть позже в этом разделе я покажу более подходящее решение.

Если между угловыми скобками использовать только точку и не ставить за ней знаков + или *, то она будет соответствовать любому одиночному символу, а шаблон будет соответствовать таким тегам, как <b> и <i>, но не будет соответствовать тегам или <textarea>.

Если нужно, чтобы соответствие относилось к символу точки (.) как таковому, его действие нужно отключить, поставив перед ним символ обратного слеша (\), поскольку в противном случае точка будет считаться метасимволом, соответствующим любому символу.

К примеру, если нужен шаблон, соответствующий числу с плавающей точкой 5.0, то в нем можно будет использовать следующее регулярное выражение:

```
/5\.0/
```

Символ обратного слеша может отключить действие любого метасимвола, в том числе и еще одного обратного слеша (если в тексте отыскивается соответствие именно обратному слешу). Но слеш может и запутать ситуацию — чуть позже будет показано, как обратные слеши иногда придают следующим за ними символам специальное предназначение.

Только что мы рассмотрели шаблон соответствия числу с плавающей точкой. Но вам наверняка понадобится проверить соответствие не только строке 5.0, но и строке 5., поскольку обе содержат значение одного и то же числа с плавающей точкой. Нужно будет также проверить соответствие строкам 5.00, 5.000 и т. д., ведь разрешено использовать любое количество нулей. Это можно сделать добавлением звездочки:

```
/5\.0*/
```

Группировка с помощью скобок

Предположим, что нужно найти соответствие таким возрастающим степеням, как кило, мега, гига и тера. Иными словами, нужно найти соответствие следующим строкам:

```
1,000
1,000,000
1,000,000,000
1,000,000,000,000
...
```

Здесь мог бы пригодиться знак «плюс», но нужно сгруппировать строку `,000` так, чтобы действие этого знака распространялось на нее целиком. Для этого служит следующее регулярное выражение:

```
/1(,000)+ /
```

Скобки означают: «При применении какого-нибудь метасимвола наподобие знака “плюс” все это нужно рассматривать как группу». Строки `1,00,000` и `1,000,00` не будут соответствовать шаблону, поскольку в тексте должен быть символ 1, за которым следует одна или несколько групп, состоящих из запятой и трех нулей.

Пробел после знака «плюс» показывает, что соответствие должно закончиться, как только встретится пробел. Без этого пробела строка `1,000,00` будет вычислена соответствующей шаблону, поскольку в расчет будет приниматься только ее первая часть `1,000`, а оставшаяся часть `,00` будет проигнорирована. Пробел нужен после остальных символов шаблона, чтобы обеспечить продолжение поиска соответствия шаблону до конца числа.

Символьный класс

Иногда требуется установить нестрогое соответствие, но не настолько пространное, чтобы для этого использовать точку. Нестрогость придает регулярным выражениям огромную мощность: можно регулировать строгость и нестрогость в соответствии с вашими желаниями.

Одним из ключевых элементов поддержки нестрогости соответствия является пара квадратных скобок (`[]`). Эта пара, как и точка, соответствует всего одному символу, но в эти скобки помещается перечень всех возможных соответствий. При появлении любого из символов этого перечня текст будет соответствовать шаблону. Например, если нужно, чтобы шаблону соответствовали оба написания — американское gray и английское grey, можно задать следующее регулярное выражение:

```
/gr[ae]y/
```

В сравниваемой части текста после `gr` может быть либо `a`, либо `e`. Но должна быть только одна из этих букв: все, что помещается внутри квадратных скобок, соответствует лишь одному символу. Группа символов внутри скобок называется *символьным классом*.

Указание диапазона

Для указания диапазона внутри квадратных скобок можно использовать дефис (`-`). Одной из самых распространенных задач является проверка соответствия отдельной цифре, в которой можно использовать диапазон:

```
/[0-9]/
```

Цифры являются настолько распространенным элементом регулярных выражений, что для их представления используется отдельный символ `\d`. Его можно использовать для проверки соответствия цифре вместо регулярного выражения в квадратных скобках:

```
/\d/
```

Инвертирование

Другим важным свойством квадратных скобок является *инвертирование* символьного класса. За счет помещения знака вставки (`^`) после открывающей квадратной скобки можно превратить весь символьный класс в его противоположность. После этого он будет означать: «Соответствует любому символу, *за исключ*ением следующих». Предположим, нужно найти экземпляры строк Yahoo, в которых отсутствует следующий за ними восклицательный знак. (Официальное название компании содержит восклицательный знак!) Для этого можно использовать такое регулярное выражение:

```
/Yahoo[^!]/
```

Символьный класс состоит из одного символа — восклицательного знака, но он инвертируется стоящим перед ним символом `^`. Вообще-то это не самое лучшее решение задачи. Например, это выражение не позволяет найти соответствие, если Yahoo находится в конце строки, поскольку тогда за этим словом не следует *что-нибудь*, а содержимому квадратных скобок должен соответствовать один символ. В более удачном решении используется упреждающее инвертирование (соответствие чему-нибудь, за чем нет ничего другого), но эта тема выходит за рамки данной книги.

Более сложные примеры

После усвоения понятий символьных классов и инвертирования вы уже готовы к изучению более удачных решений задачи поиска соответствия тегу HTML. Рассматриваемое решение позволяет шаблону не пропустить закрывающую

угловую скобку отдельного тега, но по-прежнему соответствовать таким тегам, как `<em>` и `</em>`, а также тегам с атрибутами, таким как:

```
<a href="www.mozilla.org">
```

Один из вариантов такого решения выглядит следующим образом:

```
/<[^>]+>/
```

Это регулярное выражение похоже на результат падения чашки на клавиатуру, после которого она «по-прежнему вполне исправна и работоспособна». Разобьем это выражение на части. На рис. 16.3 показан последовательный анализ всех его элементов.

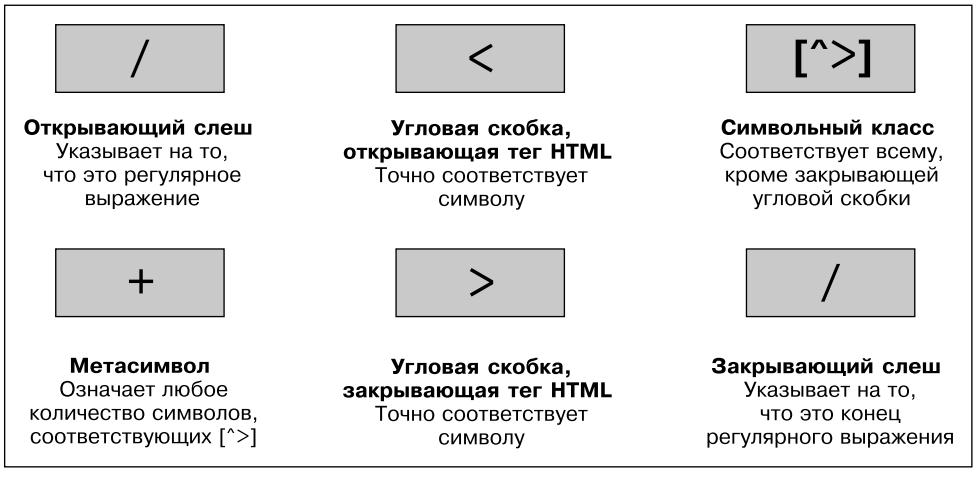

Рис. 16.3. Разбор типичного регулярного выражения

Вот эти элементы:

- `/` — открывающий слеш, указывающий на то, что это регулярное выражение;
- `<` — открывающая угловая скобка тега HTML. Требует точного соответствия, поскольку не является метасимволом;
- `[^>]` — символьный класс. Сочетание знака вставки и закрывающей угловой скобки `^>` означает: «Соответствует всему, кроме закрывающей угловой скобки»;
- `+` — допускает любое количество символов, соответствующих предыдущему регулярному выражению `[^>]`, если есть хотя бы один соответствующий ему символ;
- `>` — закрывающая угловая скобка тега HTML. Требует точного соответствия;
- `/` — закрывающий слеш, указывающий на конец регулярного выражения.

Другое решение задачи поиска соответствия тегам HTML связано с использованием так называемых нежадных инструкций. По умолчанию инструкция поиска соответствия шаблону является жадной, возвращающей наиболее длинное из всех возможных соответствий. При нежадном (или ленивом) поиске соответствия ищется соответствующая строка, наиболее короткая из возможных. Применение нежадного поиска соответствия выходит за рамки данной книги, но более подробную информацию об этом можно найти по https://javascript.info/regexp-greedy-and-lazy.

Теперь рассмотрим одно из выражений из примера 16.1, которое использовалось в функции `validateUsername`:

```
/[^a-zA-Z0-9_]/
```

На рис. 16.4 показан весь набор элементов этого выражения.

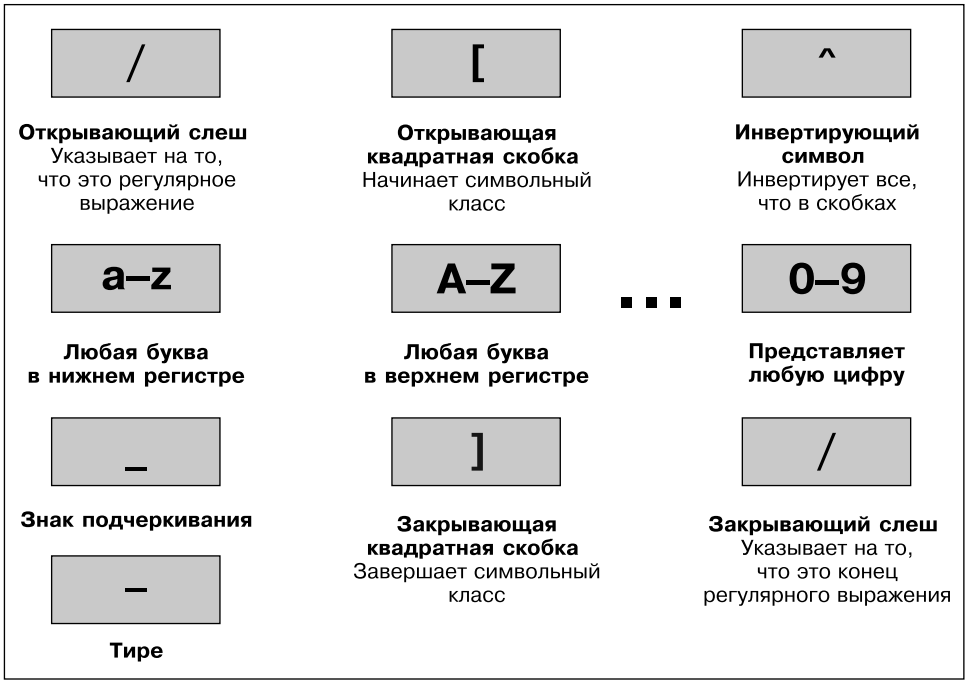

Рис. 16.4. Разбор регулярного выражения, используемого в функции validateUsername

Рассмотрим эти элементы более подробно:

- `/` — открывающий слеш, указывающий на то, что это регулярное выражение;
- `[` — открывающая квадратная скобка, с которой начинается символьный класс;
- `^` — символ инвертирования: инвертирует все, что находится в скобках;
- `a-z` — представляет любую букву в нижнем регистре;

- `A-Z` — представляет любую букву в верхнем регистре;
- `0-9` — представляет любую цифру;
- `_` — символ подчеркивания;
- `-` — тире;
- `]` — квадратная скобка, закрывающая символьный класс;
- `/` — обратный слеш, указывающий на конец регулярного выражения.

Есть еще пара весьма важных метасимволов. Они «закрепляют» регулярное выражение, требуя его применения в определенном месте. Если знак вставки (`^`) присутствует в начале регулярного выражения, то соответствующее выражению строковое значение должно быть в начале строки текста, иначе оно не будет соответствовать шаблону. По аналогии с этим если знак доллара (`$`) ставится в конце регулярного выражения, то соответствующее выражению строковое значение должно находиться в конце строки текста.

Знак вставки (^) может запутать ситуацию, поскольку внутри квадратных скобок он означает «инвертировать символьный класс», а в начале регулярного выражения — «соответствовать началу строки». К сожалению, один и тот же символ служит для достижения совершенно разных целей, поэтому при его использовании следует быть особенно внимательными.

Закончим изучение основ регулярных выражений ответом на ранее заданный вопрос: предположим, вам нужно убедиться в том, что в строке нет больше ничего, кроме того, что соответствует регулярному выражению. Что делать в том случае, если нужна строка текста, в которой нет ничего, кроме Le Guin? Можно усовершенствовать ранее рассмотренное регулярное выражение, закрепив его сразу с двух сторон:

```
/^Le *Guin$/
```

Сводная таблица метасимволов

В табл. 16.1 показаны метасимволы, используемые в регулярных выражениях.

Таблица 16.1. Метасимволы регулярных выражений

Метасимволы	Описание
/	Начало и конец регулярного выражения
.	Соответствует любому одному символу, кроме символа новой строки
*Элемент**	Соответствует появлению *элемента* от нуля и более раз

Продолжение ⇩

Таблица 16.1 (продолжение)

Метасимволы	Описание
Элемент+	Соответствует появлению *элемента* от одного раза и более
Элемент?	Соответствует появлению *элемента* от нуля до одного раза
[*символы*]	Соответствует одному из тех *символов*, которые содержатся в квадратных скобках
[^*символы*]	Соответствует одному из тех *символов*, которые не содержатся в квадратных скобках
(regex)	Рассматривает regex (сокращение, означающее регулярное выражение) как группу для вычисления или для рассмотрения с одним из следующих метасимволов: *, + или ?
Левое|правое	Соответствует либо левому, либо правому
[l–r]	Соответствует диапазону символов между l и r
^	Требует, чтобы соответствие было в начале строки
$	Требует, чтобы соответствие было в конце строки
\b	Соответствует границе слова
\B	Соответствует при отсутствии границы слова
\d	Соответствует одной цифре
\D	Соответствует одному символу, не являющемуся цифрой
\n	Соответствует символу новой строки
\s	Соответствует пробелу
\S	Соответствует символу, не являющемуся пробелом
\t	Соответствует символу табуляции
\w	Соответствует символу, используемому в словах (a–z, A–Z, 0–9 и _)
\W	Соответствует символу, не используемому в словах (все, кроме a–z, A–Z, 0–9 и _)
\x	Соответствует x (применяется, если x является метасимволом, но нужен символ x как таковой)
{n}	Соответствует в точности n появлениям
{n,}	Соответствует n и более появлениям
{min, max}	Соответствует как минимум min и как максимум max появлениям

После изучения этой таблицы и повторного исследования выражения `/[^a-zA-Z0-9_]/` можно понять, что оно легко и просто укорачивается до `/[^\w]/`, так как отдельный метасимвол `\w` (с буквой `w` в нижнем регистре) указывает на символы a–z, A–Z, 0–9 и _.

Можно проявить еще большую наблюдательность и заметить, что метасимвол `\W` (с буквой W в верхнем регистре) указывает на все символы, за исключением a–z,

A–Z, 0–9 и _. Это позволяет избавиться также от метасимвола ^ и использовать для выражения только символы /[\W]/, или пойти еще дальше и не ставить квадратные скобки, поскольку это одиночный символ.

Чтобы дать вам больше пищи для размышлений о том, что и как работает, в табл. 16.2 показан ряд выражений и описаны ситуации, которым они соответствуют.

Таблица 16.2. Примеры регулярных выражений

Пример	Соответствие
r	Первая r в The quick brown
rec[ei][ei]ve	Либо receive, либо recieve (но также и receeve или reciive)
rec[ei]{2}ve	Либо receive, либо recieve (но также и receeve или reciive)
rec(ei|ie)ve	Либо receive, либо recieve (но не receeve или reciive)
cat	Слово cat в I like cats and dogs
cat|dog	Слово cat в I like cats and dogs (соответствует либо слову cat, либо слову dog, в зависимости от того, какое из них попадется первым)
\.	Символ «.» (Знак «\» необходим, так как «.» является метасимволом)
5\.0*	5., 5.0, 5.00, 5.000 и т. д.
[a–f]	Любой из символов a, b, c, d, e или f
cats$	Только последнее слово cats в My cats are friendly cats
^my	Только первое my в my cats are my pets
\d{2,3}	Любое двух- или трехзначное число (от 00 до 999)
7(,000)+	7,000; 7,000,000; 7,000,000,000; 7,000,000,000,000 и т. д.
[\w]+	Любое слово из одного или нескольких символов
[\w]{5}	Любое слово из пяти символов

Общие модификаторы

В регулярных выражениях можно применять следующие модификаторы.

- /g — допускает «глобальное» соответствие. Применяется с функцией замены, что позволяет выполнить замену во всех соответствующих местах, а не только в месте первого соответствия.
- /i — отключает в регулярном выражении чувствительность к регистру букв. Иными словами, вместо /[a-zA-Z]/ можно указать /[a-z]/i или /[A-Z]/i.
- /m — допускает многострочный режим работы, в котором знак вставки (^) и знак доллара ($) соответствуют позициям перед любыми символами новой строки в сравниваемой строковой переменной и после них. Обычно при поиске соответствия в многострочной строковой переменной знак ^ соответствует только позиции в ее начале, а символ $ — в ее конце.

Например, выражение `/cats/g` будет соответствовать обоим появлениям слова cats в предложении I like cats and cats like me. Аналогично этому выражение `/dogs/gi` будет соответствовать обоим появлениям слова dogs (Dogs и dogs) в предложении Dogs like other dogs, поскольку эти модификаторы допускают совместное использование.

Использование регулярных выражений в JavaScript

В JavaScript регулярные выражения используются в основном в двух методах: `test` (который вы уже рассматривали) и `replace`. Метод `test` просто сообщает, соответствует ли его аргумент регулярному выражению, а метод `replace` воспринимает второй параметр — строку, которой заменяется текст, соответствующий регулярному выражению. Как и большинство методов, `replace` генерирует в качестве возвращаемого значения новую строку, входные данные при этом не изменяются.

Если сравнивать эти два метода, то следующая инструкция просто возвращает `true`, позволяя узнать, что слово `cats` появляется в строке хотя бы один раз:

```
document.write(/cats/i.test("Cats are fun. I like cats."))
```

А следующая инструкция заменяет оба имеющихся слова `cats` словом `dogs`, выводя результат на экран. Поиск должен быть глобальным (`/g`), чтобы найти все экземпляры этого слова, и нечувствительным к регистру букв (`/i`), чтобы найти слова, начинающиеся с большой буквы (Cats):

```
document.write("Cats are fun. I like cats.".replace(/cats/gi,"dogs"))
```

Если испытать эту инструкцию в работе, то проявятся ограничения функции замены: поскольку текст заменяется строго той строкой, которую предписано использовать, первое слово Cats заменяется словом dogs, а не словом Dogs.

Использование регулярных выражений в PHP

В PHP наиболее часто используются следующие функции, в которых применяются регулярные выражения: `preg_match`, `preg_match_all` и `preg_replace`.

Чтобы проверить присутствие слова cats в любом месте строки, в любой комбинации букв в нижнем и верхнем регистрах, можно воспользоваться функцией `preg_match`:

```
$n = preg_match("/cats/i", "Cats are fun. I like cats.");
```

Поскольку в PHP используется значение `1` для `TRUE` и значение `0` для `FALSE`, предыдущая инструкция присвоит переменной `$n` значение `1`. Первым аргументом функции служит регулярное выражение, а вторым — текст, проверяемый на соответствие. Но функция `preg_match` способна выполнять более сложную задачу,

поскольку она воспринимает еще и третий аргумент, который показывает, какой именно текст соответствовал регулярному выражению:

```
$n = preg_match("/cats/i", "Cats are fun. I like cats.", $match);
echo "Количество соответствий $n: $match[0]";
```

Третий аргумент является массивом (здесь ему присвоено имя `$match`). Функция помещает текст, соответствующий регулярному выражению, в первый элемент массива, поэтому, если соответствие будет найдено, соответствующий регулярному выражению текст может быть найден в элементе `$match[0]`. В данном примере выводимая на экран информация покажет, что соответствующий текст начинался с прописной буквы:

```
Количество соответствий 1: Cats
```

Если нужно определить все соответствия, используется функция `preg_match_all`:

```
$n = preg_match_all("/cats/i", "Cats are fun. I like cats.", $match);
echo "Количество соответствий $n: ";
for ($j=0 ; $j < $n ; ++$j) echo $match[0][$j]." ";
```

Как и в предыдущем случае, функции передан массив `$match` и элементу `$match[0]` присваиваются найденные соответствия, только теперь они представляют собой подмассив. Для отображения содержимого подмассива в этом примере осуществляется последовательный перебор его элементов с помощью цикла `for`.

Если нужно заменить часть строки, можно воспользоваться функцией `preg_replace`. В этом примере все встречающиеся слова cats, независимо от регистра букв, заменяются словами dogs:

```
echo preg_replace("/cats/i", "dogs", "Cats are fun. I like cats.");
```

Тема регулярных выражений слишком обширна, и о ней написана целая книга. Если вам нужна дополнительная информация, я рекомендую статью из «Википедии» либо веб-сайт по адресу https://www.regular-expressions.info.

Повторное отображение формы после проверки данных PHP-программой

Вернемся к проверке формы. На данный момент нами создан HTML-документ `validate.html`, который будет отправлен PHP-программе `adduser.php`, но это произойдет только в том случае, если поля пройдут проверку средствами JavaScript или если JavaScript отключен или недоступен.

Теперь настало время создать программу, сохраняемую в файле `adduser.php`. Эта программа получает отправленную форму и проводит собственную проверку,

а затем, если проверка не будет пройдена, снова предоставляет форму визитеру. Код, который нужно будет набрать и сохранить (или загрузить с сайта, сопутствующего книге), показан в примере 16.3.

Пример 16.3. Программа adduser.php

```
<?php // adduser.php

  // Код PHP

  $forename = $surname = $username = $password = $age = $email = "";

  if (isset($_POST['forename']))
    $forename = fix_string($_POST['forename']);
  if (isset($_POST['surname']))
    $surname  = fix_string($_POST['surname']);
  if (isset($_POST['username']))
    $username = fix_string($_POST['username']);
  if (isset($_POST['password']))
    $password = fix_string($_POST['password']);
  if (isset($_POST['age']))
    $age      = fix_string($_POST['age']);
  if (isset($_POST['email']))
    $email    = fix_string($_POST['email']);

  $fail  = validate_forename($forename);
  $fail .= validate_surname($surname);
  $fail .= validate_username($username);
  $fail .= validate_password($password);
  $fail .= validate_age($age);
  $fail .= validate_email($email);

  echo "<!DOCTYPE html>\n<html><head><title>Пример формы</title>";

  if ($fail == "")
  {
    echo "</head><body>Проверка формы прошла успешно:
      $forename, $surname, $username, $password, $age,$email.</body></html>";

    // В этом месте отправленные поля будут вводиться в базу данных
    // с предварительным использованием хеш-шифрования для пароля

    exit;
  }

  // Теперь выводится HTML и код JavaScript

  echo <<<_END

  <!-- Раздел HTML и JavaScript -->

  <style>
    .signup {
      border: 1px solid #999999;
```

```
    font: normal 14px helvetica;
    color:#444444;
  }
</style>

<script>
  function validate(form)
  {
    fail  = validateForename(form.forename.value)
    fail += validateSurname(form.surname.value)
    fail += validateUsername(form.username.value)
    fail += validatePassword(form.password.value)
    fail += validateAge(form.age.value)
    fail += validateEmail(form.email.value)

    if (fail == "")     return true
    else { alert(fail); return false }
  }

  function validateForename(field)
  {
    return (field == "") ? "Не введено имя.\\n" : ""
  }

  function validateSurname(field)
  {
    return (field == "") ? "Не введена фамилия.\\n" : ""
  }

  function validateUsername(field)
  {
    if (field == "") return "Не введено имя пользователя.\\n"
    else if (field.length < 5)
      return "В имени пользователя должно быть не менее 5 символов.\\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))
      return "В имени пользователя разрешены только a-z, A-Z, 0-9, - и _.\\n"
    return ""
  }

  function validatePassword(field)
  {
    if (field == "") return "Не введен пароль.\\n"
    else if (field.length < 6)
      return "В пароле должно быть не менее 6 символов.\\n"
    else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) ||
             !/[0-9]/.test(field))
      return "Пароль требует 1 символ из каждого набора a-z, A-Z и 0-9.\\n"
    return ""
  }

  function validateAge(field)
  {
    if (isNaN(field)) return "Не введен возраст.\\n"
    else if (field < 18 || field > 110)
      return "Возраст должен быть между 18 и 110.\\n"
```

```
      return ""
    }

    function validateEmail(field)
    {
      if (field == "") return "Не введен адрес электронной почты.\\n"
      else if (!((field.indexOf(".") > 0) &&
                 (field.indexOf("@") > 0)) ||
                /[^a-zA-Z0-9.@_-]/.test(field))
        return "Электронный адрес имеет неверный формат.\\n"
      return ""
    }
  </script>
  </head>
<body>

  <table border="0" cellpadding="2" cellspacing="5" bgcolor="#eeeeee">
    <th colspan="2" align="center">Регистрационная форма</th>

      <tr><td colspan="2">К сожалению, в вашей форме <br>
        найдены следующие ошибки: <p><font color=red
        size=1><i>$fail</i></font></p>
      </td></tr>

    <form method="post" action="adduser.php" onSubmit="return
          validate(this)">
      <tr><td>Имя</td>
        <td><input type="text" maxlength="32" name="forename"
             value="forename">
        </td></tr><tr><td>Фамилия</td>
        <td><input type="text" maxlength="32" name="surname"
             value="surname">
        </td></tr><tr><td>Пользовательское имя</td>
        <td><input type="text" maxlength="16" name="username"
             value="username">
        </td></tr><tr><td>Пароль</td>
        <td><input type="text" maxlength="12" name="password"
             value="password">
        </td></tr><tr><td>Возраст</td>
        <td><input type="text" maxlength="3" name="age" value="age">
        </td></tr><tr><td>Электронный адрес</td>
        <td><input type="text" maxlength="64" name="email"
             value="email">
        </td></tr><tr><td colspan="2" align="center"><input
          type="submit" value="Зарегистрироваться"></td></tr>
      </form>
    </table>
  </body>
</html>

_END;

  // PHP-функции

  function validate_forename($field)
```

```
  {
    return ($field == "") ? "Не введено имя<br>" : "";
  }

  function validate_surname($field)
  {
    return ($field == "") ? "Не введена фамилия<br>" : "";
  }

  function validate_username($field)
  {
    if ($field == "") return "Не введено имя пользователя<br>";
    else if (strlen($field) < 5)
      return "В имени пользователя должно быть не менее 5 символов<br>";
    else if (preg_match("/[^a-zA-Z0-9_-]/", $field))
      return "В имени пользователя допускаются только буквы, цифры, - и _<br>";
    return "";
  }

  function validate_password($field)
  {
    if ($field == "") return "Не введен пароль<br>";
    else if (strlen($field) < 6)
      return "В пароле должно быть не менее 6 символов<br>";
    else if (!preg_match("/[a-z]/", $field) ||
             !preg_match("/[A-Z]/", $field) ||
             !preg_match("/[0-9]/", $field))
      return "Пароль требует 1 символ из каждого набора a-z, A-Z и 0-9<br>";
    return "";
  }

  function validate_age($field)
  {
    if ($field == "") return "Не введен возраст<br>";
    else if ($field < 18 || $field > 110)
      return "Возраст должен быть между 18 и 110<br>";
    return "";
  }

  function validate_email($field)
  {
    if ($field == "") return "Не введен адрес электронной почты<br>";
    else if (!((strpos($field, ".") > 0) &&
               (strpos($field, "@") > 0)) ||
                preg_match("/[^a-zA-Z0-9.@_-]/", $field))
      return "Электронный адрес имеет неверный формат<br>";
    return "";
  }

  function fix_string($string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return htmlentities ($string);
  }
?>
```

В этом примере все вводимые данные перед применением обезвреживаются, даже пароли, которые из-за возможного содержания в них символов, используемых для форматирования HTML, будут превращены в HTML-последовательности. Например, & станет & а < превратится в < и т. д. Если для сохранения зашифрованных паролей будет использоваться функция hash, это не создаст проблем в том случае, если при последующей проверке введенного пароля он будет обезвреживаться тем же способом и сравниваться будут такие же вводимые данные.

Результат отправки формы при отключенном JavaScript (и двумя неправильно заполненными полями) показан на рис. 16.5.

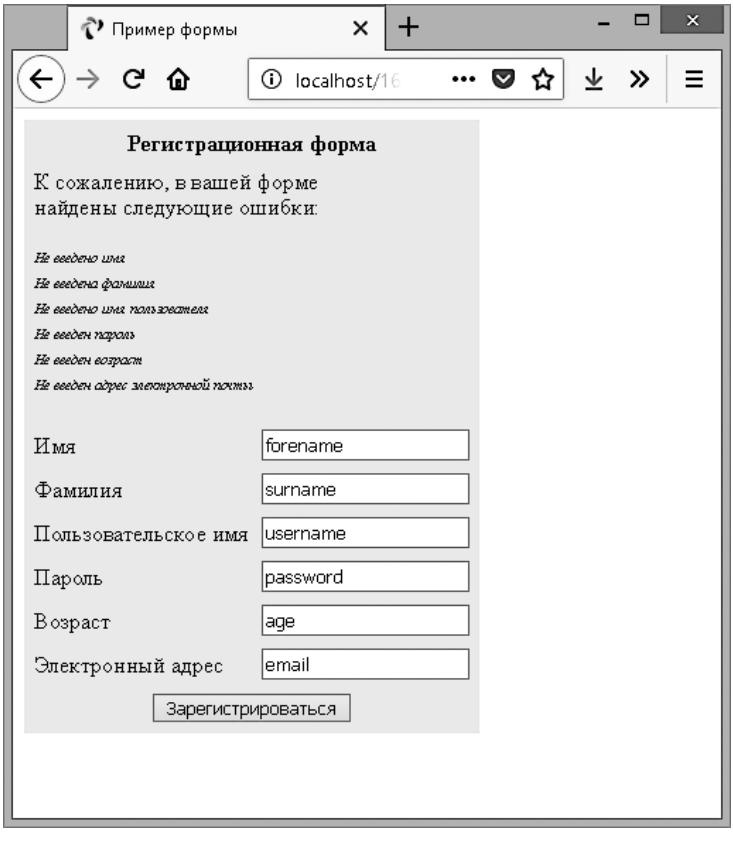

Рис. 16.5. Форма, отображаемая после того, как она не прошла проверку средствами PHP

PHP-раздел этого кода и изменения, внесенные в HTML-раздел, выделены полужирным шрифтом, чтобы сделать заметнее все отличия кода этого примера от кода, который был показан в примерах 16.1 и 16.2.

Если вы внимательно изучили этот пример (либо набрали или загрузили его с сайта http://lpmj.net), то вы увидели, что код PHP является практически клоном кода JavaScript. Для проверки каждого из полей в очень похожих функциях используются те же самые регулярные выражения.

Но здесь следует отметить две особенности. Во-первых, для обезвреживания содержимого каждого поля и предотвращения любых попыток внедрения кода применяется функция `fix_string`, которая находится в самом конце примера.

Во-вторых, вы должны были заметить, что код HTML из примера 16.1, повторенный в PHP-коде внутри структуры `<<<_END..._END;`, отображает форму со значениями, которые посетитель ввел при предыдущей попытке заполнения формы. Это сделано за счет простого добавления еще одного параметра `value` к каждому тегу `<input>` (например, `value="$forename"`). Проявление такой заботы о пользователе всячески приветствуется, поскольку при этом ему не приходится снова и снова заполнять все поля, а остается лишь отредактировать ранее введенные данные.

При разработке реального проекта вы вряд ли стали бы сначала создавать HTML-форму вроде той, что показана в примере 16.1. Скорее всего, вместо этого вы сразу перешли бы к созданию PHP-программы, показанной в примере 16.3, в которую включен весь код HTML. И разумеется, вам потребовались бы небольшие доработки для первого вызова этой программы, чтобы заблокировать отображение ошибок при еще не заполненных полях. К тому же следовало бы выделить шесть функций JavaScript в отдельный включаемый файл с расширением .js, в соответствии с рекомендациями, изложенными ранее в подразделе «Использование отдельного файла JavaScript».

После рассмотрения способа объединения кода PHP, HTML и JavaScript в следующей главе будет представлена технология *AJAX* (Asynchronous JavaScript And XML — асинхронный JavaScript и XML), в которой используются фоновые JavaScript-вызовы, обращенные к серверу для получения плавного обновления фрагментов веб-страницы, при котором не требуется повторная отправка всего ее содержимого с веб-сервера.

Вопросы

1. Каким методом JavaScript можно воспользоваться, чтобы послать данные формы на проверку перед их отправкой на сервер?
2. Какой метод JavaScript применяется для проверки соответствия строки регулярному выражению?
3. Используя определения синтаксиса регулярных выражений, напишите такое регулярное выражение, которое будет соответствовать любым символам, *не* использующимся в словах.

4. Напишите регулярное выражение, которое будет соответствовать как слову fox, так и слову fix.
5. Напишите регулярное выражение, которое будет соответствовать любому отдельному слову, за которым следует любой символ, не использующийся в словах.
6. Используя регулярное выражение, напишите функцию JavaScript, проверяющую наличие слова fox в строке `The quick brown fox`.
7. Используя регулярное выражение, напишите функцию PHP, заменяющую все экземпляры слова the в строке `The cow jumps over the moon` словом my.
8. Какой атрибут HTML используется для предварительного заполнения полей формы значениями?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

17 Использование технологии асинхронного обмена данными

Термин *AJAX* был придуман в 2005 году. Изначально он расшифровывался как *«асинхронный JavaScript и XML»* (Asynchronous JavaScript and XML), что, проще говоря, означало использование набора методов, встроенных в JavaScript, для обмена данными между браузером и сервером в фоновом режиме. В настоящее время этот термин в основном вышел из употребления, и теперь эту технологию называют просто асинхронным обменом данными.

Превосходным примером применения этой технологии является Google Maps (рис. 17.1), где новый участок карты загружается с сервера по мере необходимости, для чего не требуется обновление всей страницы.

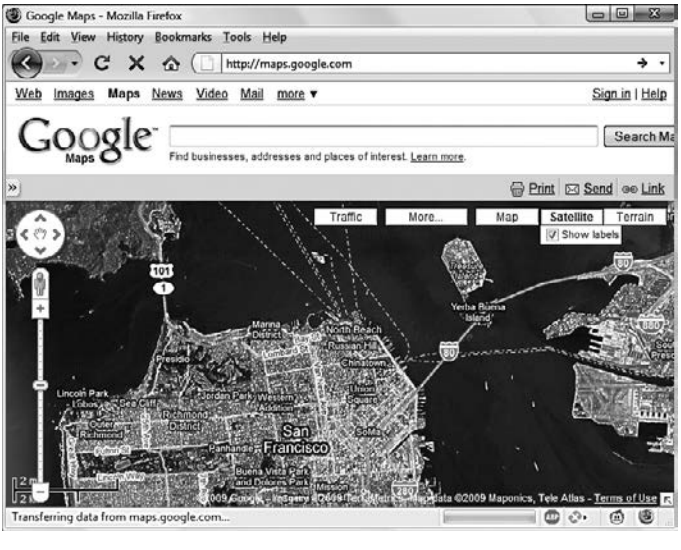

Рис. 17.1. Google Maps — превосходный пример использования технологии асинхронного обмена данными

Использование асинхронного обмена данными не только приводит к существенному снижению объема данных, передаваемых в обе стороны, но и обеспечивает плавную динамичность веб-страниц, делая их поведение характерным для самостоятельных приложений. В результате значительно улучшается пользовательский интерфейс и ускоряется реакция на действия пользователя.

Что такое асинхронный обмен данными

Используемый в наши дни асинхронный обмен данными берет свое начало в 1999 году с выпуска Internet Explorer 5, где был представлен новый ActiveX-объект `XMLHttpRequest`. Разработанная в корпорации Microsoft технология ActiveX предусматривает использование дополнительных программных модулей, устанавливаемых на ваш компьютер. Позже разработчики других браузеров поддержали этот почин, но вместо применения ActiveX они разработали функциональный модуль, ставший неотъемлемой частью интерпретатора JavaScript.

Но к тому времени уже появилась ранняя форма технологии, использующая на странице скрытые фреймы, которые взаимодействуют с сервером в фоновом режиме. Самыми первыми потребителями этой технологии были участники форумов, которые задействовали ее для опроса и отображения новых сообщений без перезагрузок страницы.

В этой главе мы рассмотрим, как реализовать асинхронный обмен данными, используя JavaScript.

XMLHttpRequest

Из-за различий в реализации `XMLHttpRequest`, имеющихся в разных браузерах, возникает необходимость в создании специальной функции, обеспечивающей работу вашего кода на всех основных браузерах. Для этого необходимо разобраться с тремя способами создания объекта `XMLHttpRequest`:

- IE 5: `request = new ActiveXObject("Microsoft.XMLHTTP");`
- IE 6+: `request = new ActiveXObject("Msxml2.XMLHTTP");`
- все остальные браузеры: `request = new XMLHttpRequest();`.

Дело в том, что Microsoft с выпуском Internet Explorer 6 решила внести изменения, тогда как все остальные браузеры используют несколько иной метод. Поэтому код, показанный в примере 17.1, будет работать на всех основных браузерах, выпущенных за последние несколько лет.

Пример 17.1. Кросс-браузерная функция асинхронного обмена данными

```
<script>
  function asyncRequest()
  {
    try  // Браузер не относится к семейству IE?
    {    // Да
      var request = new XMLHttpRequest()
    }
    catch(e1)
    {
      try   // Это IE 6+?
      {     // Да
        request = new ActiveXObject("Msxml2.XMLHTTP")
      }
      catch(e2)
      {
        try  // Это IE 5?
        {    // Да
          request = new ActiveXObject("Microsoft.XMLHTTP")
        }
        catch(e3) // Данный браузер не поддерживает
                  // асинхронный обмен данными
        {
          request = false
        }
      }
    }
    return request
  }
</script>
```

Вспомним элементарные способы обработки ошибок из предыдущей главы, где применялась конструкция `try...catch`. Код примера 17.1 является прекрасной иллюстрацией той пользы, которую можно извлечь из применения данной конструкции, поскольку в этом коде ключевое слово `try` задействуется для выполнения команды не в формате IE и в случае успеха — перехода к завершающей инструкции `return`, возвращающей новый объект.

В противном случае с помощью инструкции `catch` осуществляется перехват ошибки и выполняется следующая команда. И опять в случае успеха возвращается новый объект, а в случае неудачи предпринимается попытка выполнения последней из трех команд. Если эта попытка окажется неудачной, значит браузер не поддерживает асинхронный обмен данными и объект `request` получает значение `false`, а в случае удачи возвращается полноценный объект. Итак, теперь у вас есть кросс-браузерная функция запроса, которую можно будет добавить к вашей библиотеке полезных функций JavaScript.

Теперь, когда вы располагаете средством для создания объекта `XMLHttpRequest`, возникает вопрос о том, что можно делать с подобными объектами. Каждый такой объект поступает с набором свойств (переменных) и методов (функций), перечисленных в табл. 17.1 и 17.2 соответственно.

Таблица 17.1. Свойства объектов XMLHttpRequest

Свойства	Описание
onreadystatechange	Определяет функцию обработки события, вызываемую при изменении имеющегося в объекте свойства readyState
readyState	Целочисленное свойство, дающее представление о состоянии запроса. Оно может иметь любое из следующих значений: 0 = неинициализирован, 1 = загружается, 2 = загружен, 3 = в состоянии диалога или 4 = завершен
responseText	Данные, возвращенные сервером в текстовом формате
responseXML	Данные, возвращенные сервером в формате XML
status	Код статуса HTTP, возвращенный сервером
statusText	Текст статуса HTTP, возвращенный сервером

Таблица 17.2. Методы объектов XMLHttpRequest

Методы	Описание
abortO	Отмена текущего запроса
getANResponseHeaders()	Возвращение всех заголовков в виде строк
getResponseHeader(*параметр*)	Возвращение значения *параметра* в виде строки
open(*'метод', 'urf', 'асинхронно'*)	Определение используемого HTTP-метода (GET или POST), целевого URL-адреса и обязательности обработки запроса в асинхронном режиме (true или false)
send(*данные*)	Отправка *данных* серверу назначения с использованием указанного HTTP-метода
setRequestHeader(*'параметр', 'значение'*)	Установка в заголовок пары «параметр — значение»

Перечисленные свойства и методы позволяют управлять данными, отправляемыми на сервер и получаемыми в ответ, а также выбирать методы отправки и получения этих данных. Например, можно выбрать формат запрашиваемых данных: текстовый (который может включать HTML и прочие теги) или XML. Можно также решить, какой из методов — POST или GET — следует использовать для отправки данных на сервер.

Рассмотрим сначала метод POST, создав очень простую пару документов: комбинацию из HTML и JavaScript — и PHP-программу для взаимодействия с первым документом в асинхронном режиме. В этих примерах за счет использования лишь нескольких строк JavaScript у стороннего веб-сервера запрашивается веб-документ, который затем возвращается браузеру вашим сервером и размещается в определенном разделе текущего документа.

Ваша первая программа, использующая асинхронный обмен данными

Наберите и сохраните в файле `urlpost.html` код примера 17.2, но пока не загружайте его в свой браузер.

Пример 17.2. urlpost.html

```
<!DOCTYPE html>
<html>  <!-- urlpost.html -->
  <head>
    <title>Пример асинхронного обмена данными</title>
  </head>
  <body style='text-align:center'>
    <h1>Загрузка веб-страницы в контейнер DIV</h1>
    <div id='info'>Это предложение будет заменено</div>

    <script>
      params = "url= news.com"
      request = new asuncRequest()

      request.open("POST", "urlpost.php", true)
      request.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded")
      request.setRequestHeader("Content-length", params.length)
      request.setRequestHeader("Connection", "close")

      request.onreadystatechange = function()
      {
        if (this.readyState == 4)
        {
          if (this.status == 200)
          {
            if (this.responseText != null)
            {
              document.getElementById('info').innerHTML = this.responseText
            }
            else alert("Ошибка обмена данными: Данные не получены")
          }
          else alert( "Ошибка обмена данными: " + this.statusText)
        }
      }

      request.send(params)

      function asuncRequest()
      {
        try
        {
          var request = new XMLHttpRequest()
        }
```

```
        catch(e1)
        {
          try
          {
            request = new ActiveXObject("Msxml2.XMLHTTP")
          }
          catch(e2)
          {
            try
            {
              request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
              request = false
            }
          }
        }
        return request
      }
    </script>
  </body>
</html>
```

Разберем этот документ и посмотрим, что он делает, начиная с первых шести строк, в которых устанавливается, что это HTML-документ, и отображается его заголовок. В следующей строке создается тег `<div>` с ID `info`, в котором изначально содержится текст: `Это предложение будет заменено`. Позже сюда будет вставлен текст, возвращенный в результате вызова.

Следующие шесть строк нужны для создания HTTP-запроса POST. В первой строке переменной `params`, которая отправляется на сервер, присваивается значение, состоящее из пары *`параметр`* = *`значение`*. Затем создается AJAX-объект запроса. После этого вызывается метод `open`, настраивающий объект на создание POST-запроса по адресу `urlpost.php` в асинхронном режиме. Последние три строки в этой группе настраивают заголовки, необходимые для того, чтобы получающий сервер знал о поступлении POST-запроса.

Свойство readyState

Теперь мы наконец добрались до самых тонкостей асинхронного вызова, которые целиком базируются на использовании свойства `readyState`. Оно позволяет браузерам реагировать на пользовательский ввод и изменять содержимое экрана при условии, что наша программа настраивает свойство `onreadystatechange` на то, чтобы при каждом изменении свойства `readyState` вызывалась выбранная нами функция. В данном случае будет использоваться не отдельная функция, имеющая собственное имя, а безымянная (или анонимная) встроенная функция. Она относится к так называемым *функциям обратного вызова*, поскольку вызывается при каждом изменении свойства `readyState`.

Синтаксис объявления функции обратного вызова, в котором применяется встроенная безымянная функция, имеет следующий вид:

```
request.onreadystatechange = function()
{
  if (this.readyState == 4)
  {
    // какие-нибудь действия
  }
}
```

Если нужно воспользоваться отдельной функцией, имеющей собственное имя, применяется несколько иной синтаксис:

```
request.onreadystatechange = asuncCallback

function asuncCallback()
{
  if (this.readyState == 4)
  {
    // какие-нибудь действия
  }
}
```

При изучении табл. 17.1 можно увидеть, что у свойства `readyState` могут быть пять значений. Но только одно из них представляет для нас интерес: значение `4`, которое свидетельствует о завершении вызова. Поэтому при каждом вызове новой функции она возвращает управление без каких-либо действий до тех пор, пока свойство `readyState` не получит значение `4`. Когда наша функция обнаружит это значение, следующим своим действием она проверит статус вызова, чтобы убедиться в том, что он имеет значение `200`, означающее, что вызов прошел удачно.

Если этот статус не равен `200`, выводится окно предупреждения с сообщением об ошибке, которое содержится в свойстве `statusText`.

Обратите внимание на то, что на все эти свойства объекта идут ссылки this. readyState, this.status и т. д., без использования текущего имени объекта request, как в ссылках request.readyState или request.status. Это сделано для того, чтобы дать вам возможность просто скопировать и вставить код и чтобы он после этого смог работать с любым именем объекта, поскольку ключевое слово this всегда ссылается на текущий объект.

Итак, после того как установлено, что `readyState` равен `4`, а `status` равен `200`, проверяется наличие значения у свойства `responseText`. Если значение отсутствует, в окне предупреждения выводится сообщение об ошибке. В противном случае содержимому контейнера `<div>` присваивается значение свойства `responseText`:

```
document.getElementById('info').innerHTML = this.responseText
```

В этой строке с помощью метода `getElementByID` осуществляется ссылка на элемент `info`, а затем его свойству `innerHTML` присваивается значение, возвращенное вызовом. В результате чего данный элемент веб-страницы изменяется, а все остальное остается прежним.

После всех этих настроечных и подготовительных действий асинхронный запрос наконец-то посылается на сервер с помощью следующей команды, которой передаются параметры, заранее определенные в переменной `params`:

```
request.send(params)
```

Затем весь предыдущий код активизируется при каждом изменении свойства `readyState`. В конце документа находятся метод `asuncRequest` из примера 17.1 и теги, закрывающие сценарий JavaScript и код HTML.

Серверная половина процесса асинхронного обмена данными

Теперь мы добрались до PHP-половины этого уравнения, которая показана в примере 17.3. Наберите этот код и сохраните его в файле `urlpost.php`.

Пример 17.3. urlpost.php

```
<?php // urlpost.php
  if (isset($_POST['url'])) {
    echo file_get_contents('http://' . SanitizeString($_POST['url']));
  }
  function SanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var); return stripslashes($var);
  }
?>
```

Как видите, этот код невелик по объему и использует неизменно актуальную функцию обезвреживания содержимого строки — `SanitizeString`, которая должна применяться ко всем отправляемым в адрес сервера данным. В этом случае необезвреженные данные могут привести к получению пользователем возможностей управления вашим кодом.

В этой программе для загрузки веб-страницы, которая находится по URL-адресу, представленному в переменной `$_POST['url']`, применяется PHP-функция `file_get_contents`. Эта функция обладает достаточной универсальностью, позволяющей ей загружать все содержимое файла или веб-страницы как с локального, так и с удаленного сервера, — она даже учитывает перемещенные страницы и другие перенаправления.

После набора программы можно будет вызвать в браузере файл `urlpost.html`, и через несколько секунд должна появиться первая страница сайта news.com, содержимое которой загружено в `<div>`-контейнер, созданный нами для этих целей.

Произойдет это не так быстро, как при непосредственной загрузке веб-страницы, поскольку данные переносятся дважды: сначала на сервер, а потом с сервера на браузер. Результат должен быть похож на тот, что показан на рис. 17.2.

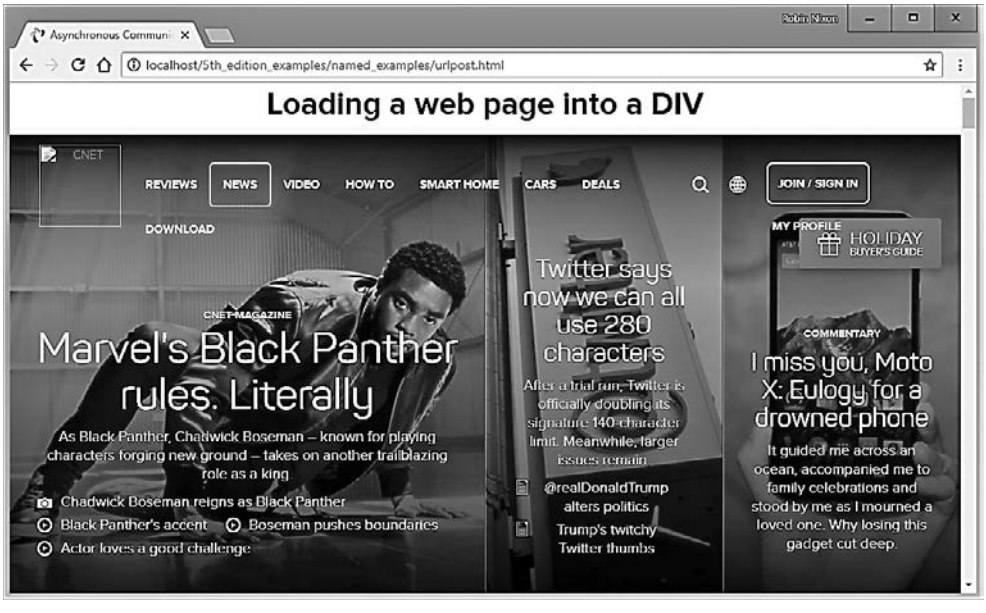

Рис. 17.2. Первая страница сайта news.com, загруженная в -контейнер

Мы не только добились осуществления асинхронного вызова и получения ответа, возвращенного JavaScript, но и воспользовались способностью PHP объединять совершенно не связанные друг с другом веб-объекты. Кстати, если бы мы попытались найти способ извлечения этой веб-страницы непосредственно через асинхронный вызов (без обращения к PHP-модулю на стороне сервера), у нас ничего бы не вышло, поскольку существуют блоки безопасности, не допускающие кросс-доменного применения технологии асинхронного обмена данными. Поэтому данный небольшой пример показывает также удобное решение весьма актуальной практической задачи.

Использование GET вместо POST

При отправке любых данных из формы можно выбрать GET-запросы, сэкономив на этом несколько строк кода. Но у таких запросов есть недостаток: некоторые браузеры могут кэшировать GET-запросы, притом что POST-запросы кэшированию никогда не подвергаются. Кэширование запроса нежелательно, потому что браузер просто-напросто заново отобразит то, что он получил в последний раз, и не станет обращаться к серверу за свежими входными данными. Решить эту проблему можно,

применив обходной маневр, заключающийся в добавлении к каждому запросу произвольного параметра, обеспечивающего уникальность каждого запрашиваемого URL-адреса.

В коде примера 17.4 показано, как можно добиться такого же результата, который был получен при использовании кода примера 17.2, но на этот раз применяя в AJAX не POST-, а GET-запрос.

Пример 17.4. urlget.html

```
<!DOCTYPE html>
<html>  <!-- urlget.html -->
  <head>
    <title>Пример асинхронного обмена данными </title>
  </head>
  <body style='text-align:center'>
    <h1>Загрузка веб-страницы в DIV-контейнер</h1>
    <div id='info'>Это предложение будет заменено<div>

    <script>
      nocache = "&nocache=" + Math.random() * 1000000
      request = new asuncRequest()
      request.open("GET", "urlget.php?url=news.com " + nocache, true)

      request.onreadystatechange = function()
      {
        if (this.readyState == 4)
        {
          if (this.status == 200)
          {
            if (this.responseText != null)
            {
              document.getElementById('info').innerHTML = this.responseText
            }
            else alert("Ошибка обмена данными: Данные не получены ")
          }
          else alert( "Ошибка обмена данными: " + this.statusText)
        }
      }

      request.send(null)

      function asuncRequest()
      {
        try
        {
          var request = new XMLHttpRequest()
        }
        catch(e1)
        {
          try
          {
            request = new ActiveXObject("Msxml2.XMLHTTP")
          }
```

```
          catch(e2)
          {
            try
            {
              request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
              request = false
            }
          }
        }
        return request
      }
    </script>
  </body>
</html>
```

Различия между этими двумя документами, на которые нужно обратить внимание, выделены полужирным шрифтом и состоят в следующем.

- Для GET-запроса не требуется отправка заголовков.
- Метод `open` вызывается с использованием GET-запроса с предоставлением URL- адреса, строка которого содержит символ `?`, а за ним следует пара «параметр — значение» — `url=news.com`.
- Вторая пара «параметр — значение» начинается с использования символа `&`, за которым для параметра `nocache` устанавливается случайное значение из диапазона от 0 до 1 000 000. Такой прием обеспечивает разное содержимое каждого запрашиваемого URL-адреса, что препятствует обслуживанию запросов из кэша.
- Вызов метода `send` теперь содержит только параметр `null`, поскольку здесь отсутствуют параметры, которые передаются при POST-запросе. Учтите, что опускать этот параметр нельзя, поскольку это вызовет ошибку.

Для сопровождения нового документа необходимо изменить PHP-программу так, чтобы она отвечала на GET-запрос. Файл `urlget.php`, в котором содержится код программы, показан в примере 17.5.

Пример 17.5. urlget.php

```
<?php
  if (isset($_GET['url']))
  {
    echo file_get_contents("http://".sanitizeString($_GET['url']));
  }

  function sanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var); return stripslashes($var);
  }
?>
```

Единственная разница между этим кодом и кодом примера 17.3 заключается в том, что ссылка на массив `$_POST` заменена ссылкой на массив `$_GET`. Конечный результат вызова `urlget.html` в вашем браузере будет идентичен результату вызова `urlpost.html`.

Отправка XML-запросов

Хотя создаваемые нами объекты называются объектами `XMLHttpRequest`, пока мы обходились без использования XML. Вы смогли убедиться в том, что мы запрашивали с помощью асинхронного запроса весь HTML-документ, но могли бы с таким же успехом запросить текстовую страницу, строку или число, или даже данные электронной таблицы.

Внесем изменения в приведенный ранее пример документа и PHP-программы и настроим их на извлечение данных в формате XML. Для этого рассмотрим сначала PHP-программу `xmlget.php`, показанную в примере 17.6.

Пример 17.6. xmlget.php

```
<?php
  if (isset($_GET['url']))
  {
    header('Content-Type: text/xml');
    echo file_get_contents("http://".sanitizeString($_GET['url']));
  }

  function sanitizeString($var)
  {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
  }
?>
```

Эта программа по сравнению с предыдущей подверглась небольшому изменению (которое выделено полужирным шрифтом), чтобы перед возвращением извлеченного документа выводился правильный XML-заголовок. Здесь не выполняются никакие проверки, поскольку предполагается, что вызов кода запросит настоящий XML-документ.

Теперь рассмотрим HTML-документ `xmlget.html`, показанный в примере 17.7.

Пример 17.7. xmlget.html

```
<!DOCTYPE html>
<html>  <!-- xmlget.html -->
  <head>
    <title>Пример асинхронного обмена данными</title>
  </head>
  <body>
```

```
<h1>Загрузка XML-содержимого в DIV-контейнер</h1>
<div id='info'>Это предложение будет заменено</div>

<script>
  nocache = "&nocache=" + Math.random() * 1000000
  url     = "rss.news.yahoo.com/rss/topstories"
  out     = "";

  request = new asuncRequest()
  request.open("GET", "xmlget.php?url=" + url + nocache, true)

  request.onreadystatechange = function()
  {
    if (this.readyState == 4)
    {
      if (this.status == 200)
      {
        if (this.responseText != null)
        {
          titles = this.responseXML.getElementsByTagName('title')

          for (j = 0 ; j < titles.length ; ++j)
          {
            out += titles[j].childNodes[0].nodeValue + '<br>'
          }
          document.getElementById('info').innerHTML = out
        }
        else alert("Ошибка обмена данными: Данные не получены")
      }
      else alert("Ошибка обмена данными: " + this.statusText)
    }
  }

  request.send(null)

  function asuncRequest()
  {
    try
    {
      var request = new XMLHttpRequest()
    }
    catch(el)
    {
      try
      {
        request = new ActiveXObject("Msxml2.XMLHTTP")
      }
      catch(e2)
      {
        try
        {
          request = new ActiveXObject("Microsoft.XMLHTTP")
        }
```

```
            catch(e3)
            {
              request = false
            }
          }
        }
        return request
      }
    </script>
  </body>
</html>
```

В этом коде все различия также выделены полужирным шрифтом, чтобы вы могли увидеть, что он очень похож на предыдущие версии, за исключением того, что теперь запрашивается URL-адрес rss.news.yahoo.com/rss/topstories, по которому находится XML-документ, содержащий поток последних новостей — Yahoo! News Top Stories feed.

Другое существенное изменение касается использования свойства `responseXML`, которым заменено свойство `responseText`. Когда сервер возвращает XML-данные, свойство `responseText` возвращает значение `null`, а свойство `responseXML` будет содержать возвращенные XML-данные.

Но `responseXML` не просто содержит строку XML-текста — на самом деле в нем находится полноценный объект XML-документа, который может быть проанализирован с использованием методов и свойств DOM-дерева. Это, к примеру, означает, что к нему можно применить JavaScript-метод `getElementsByTagName`.

Несколько слов о XML

Документ XML, как правило, имеет форму RSS-потока, показанного в примере 17.8. Но красота XML заключается в том, что этот тип структуры может быть сохранен внутри DOM-дерева (рис. 17.3), что дает возможность выполнять в нем быстрый поиск.

Пример 17.8. Документ XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>RSS-поток</title>
    <link>http://website.com</link>
    <description>RSS-поток website.com </description>
    <pubDate>Понедельник, 11 мая 2020 года, 00:00:00 GMT</pubDate>
    <item>
      <title>Заголовок</title>
      <guid>http://website.com/headline</guid>
      <description>Это заголовок</description>
    </item>
    <item>
      <title>Заголовок 2</title> <guid>http://website.com/headline2</guid>
```

```
        <description>Второй заголовок</description>
    </item>
  </channel>
</rss>
```

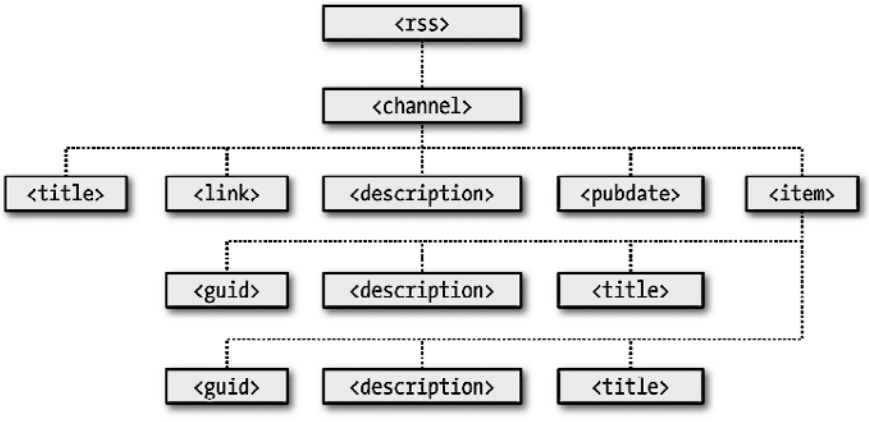

Рис. 17.3. DOM-дерево примера 17.8

Затем, используя метод `getElementsByTagName`, можно быстро извлечь значения, связанные с различными тегами, не занимаясь громоздким строчным поиском. Именно это и делается в коде примера 17.7, где выдается следующая команда:

```
titles = this.responseXML.getElementsByTagName('title')
```

За счет выполнения только этой одной команды все значения элементов `<title>` помещаются в массив `titles`. После этого остается лишь извлечь их с помощью следующего выражения (где `j` было присвоено целочисленное значение, представляющее заголовок, к которому осуществляется доступ):

```
titles[j].childNodes[0].nodeValue
```

Затем все заголовки добавляются к строковой переменной `out`, и, поскольку все они уже прошли обработку, результат вставляется в пустой `<div>`-контейнер в начале документа. При вызове в браузере файла `xmlget.html` будет получен результат, похожий на тот, что показан на рис. 17.4.

Следует вкратце напомнить, что каждый объект вроде заголовка, является узлом, и таким образом, к примеру, текст заголовка считается узлом внутри заголовка. Но даже после получения дочернего узла его нужно запросить в виде текста, для чего и предназначен метод .nodeValue. При запросе XML-данных, равно как и при работе со всеми другими формами данных, следует помнить, что можно воспользоваться либо методом POST, либо методом GET — ваш выбор на результат не повлияет.

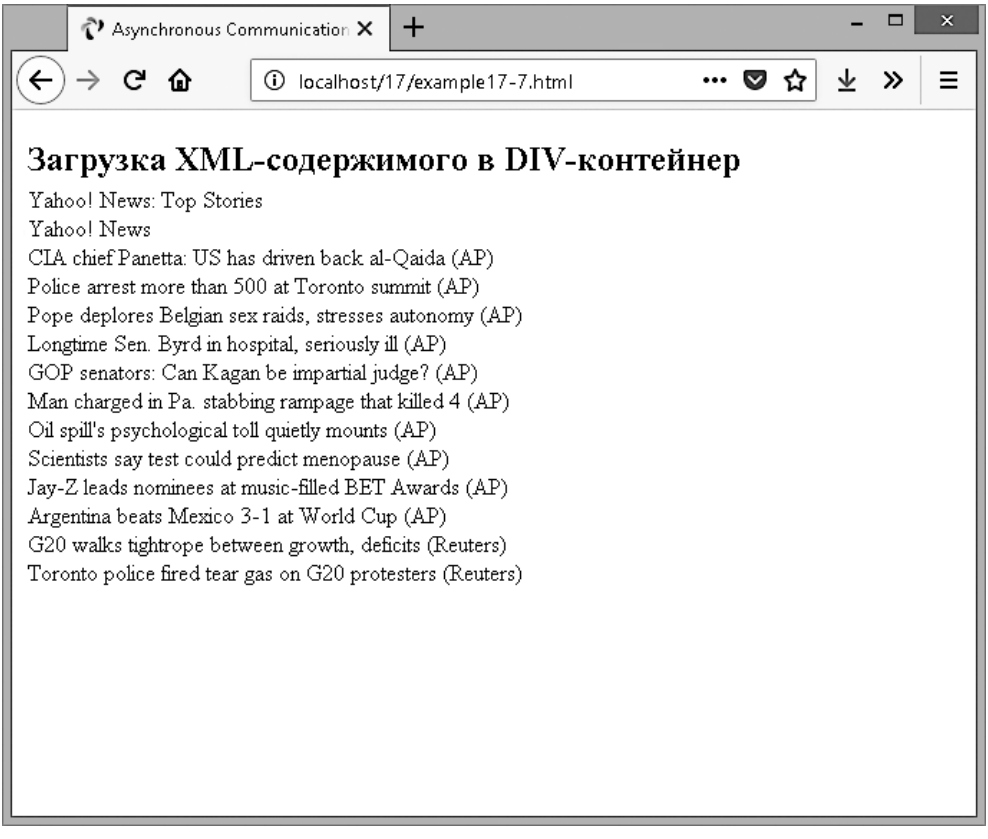

Рис. 17.4. Извлечение новостного XML-потока Yahoo! в асинхронном режиме

А зачем вообще использовать XML?

Может возникнуть вопрос, а для чего еще можно применять XML, кроме как для извлечения XML-документов в виде RSS-потоков? Проще всего ответить, что пользоваться им вас никто не заставляет, но если вам нужно возвращать структурированные данные своим приложениям, то отправка простых, неорганизованных фрагментов текста может превратиться в настоящую проблему, для решения которой потребуется довольно сложная обработка в JavaScript.

Вместо этого можно создать XML-документ и вернуть его вызывающей функции, которая автоматически поместит его в DOM-дерево в виде уже знакомого вам легкодоступного HTML DOM-объекта.

В наши дни в качестве формата обмена данными программисты больше склонны к использованию формата JSON (см. http://json.org), поскольку он является простым подмножеством JavaScript.

Использование специальных сред для асинхронного обмена данными

Теперь, когда вы узнали о том, как создавать собственные процедуры асинхронного обмена данными, можно будет исследовать некоторые из свободно распространяемых программных продуктов, представляющие собой среду, способную упростить работу с применением этой технологии и предлагающую множество более совершенных функциональных возможностей. В частности, я советую обратить внимание на библиотеку jQuery, которая, наверное, является наиболее востребованной средой для работы с применением AJAX и рассматривается в главе 21. А в следующей главе мы рассмотрим применение стилей к вашему сайту с помощью CSS.

Вопросы

1. Зачем нужна функция для создания новых `XMLHttpRequest`-объектов?
2. Для чего предназначена конструкция `try...catch`?
3. Сколько свойств и методов имеется у объекта `XMLHttpRequest`?
4. Как можно определить завершение асинхронного вызова?
5. Как узнать об успешном завершении асинхронного вызова?
6. В каком свойстве объекта `XMLHttpRequest` содержится текстовый ответ, возвращенный асинхронным вызовом?
7. В каком свойстве объекта `XMLHttpRequest` содержится XML-ответ, возвращенный AJAX-вызовом?
8. Как указать функцию обратного вызова, предназначенную для обработки ответов асинхронного вызова?
9. Какой метод объекта `XMLHttpRequest` используется для инициирования асинхронного запроса?
10. В чем состоит основное различие между GET- и POST-запросом при асинхронном обмене данными?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

18 Введение в CSS

Используя каскадные таблицы стилей — Cascading Style Sheets (CSS), — вы можете применить стиль к своим веб-страницам, чтобы придать им желаемый внешний вид. Работа CSS основана на их подключении к объектной модели документа — Document Object Model (DOM), которая была рассмотрена в главе 13.

Используя CSS и их интеграцию с DOM, можно быстро и просто изменить стиль любого элемента. Например, если не нравится исходный вид заголовков, определяемых тегами `<h1>`, `<h2>` и т. д., можно назначить новый стиль, отменяющий исходные настройки, касающиеся используемого семейства шрифтов и размера, применения полужирного шрифта или курсива, а также многих других свойств.

Один из способов добавления стилей к веб-странице заключается во вставке требуемых для этого инструкций в заголовок страницы между тегами `<head>` и `</head>`. Поэтому для изменения стиля, применяемого к содержимому тега `<h1>`, нужно воспользоваться следующим кодом (синтаксис которого будет рассмотрен чуть позже):

```
<style>
  hi { color:red; font-size:3em; font-family:Arial; }
</style>
```

Внутри HTML-страницы этот код может иметь вид, показанный в примере 18.1, в котором, подобно всем остальным примерам, используемым в данной главе, применяется стандартное HTML5-объявление `DOCTYPE` (рис. 18.1).

Пример 18.1. Простая HTML-страница

```
<!DOCTYPE html>
<html>
  <head>
    <title>Здравствуй, мир!</title>
    <style>
      h1 { color:red; font-size:3em; font-family:Arial; }
```

```
    </style>
  </head>
  <body>
    <h1>Всем привет</h1>
  </body>
</html>
```

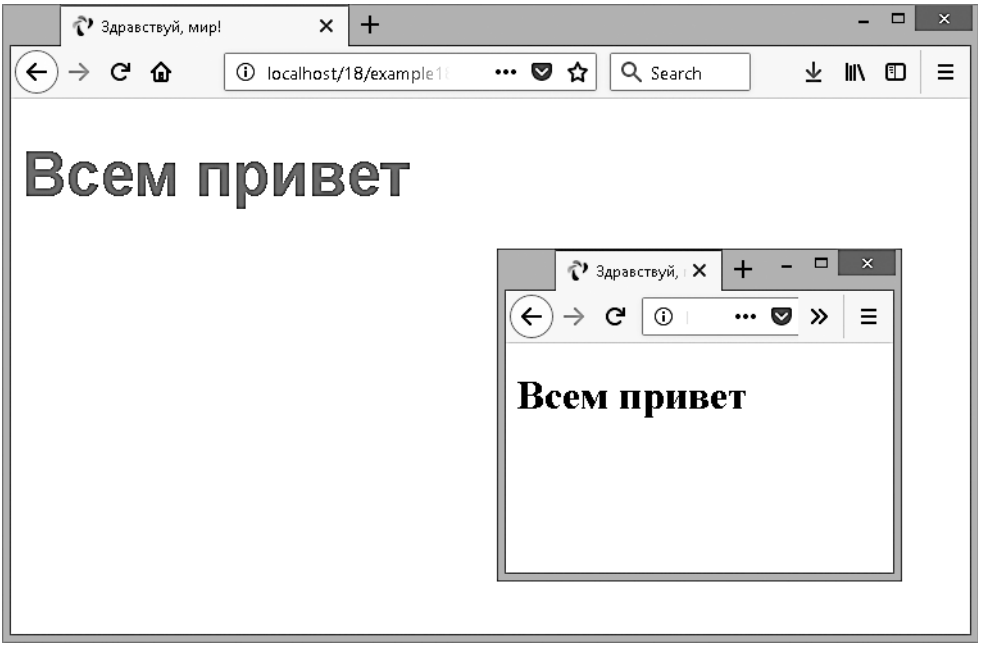

Рис. 18.1. Стилизация тега, оригинальный стиль которого показан во вставке

Импортирование таблицы стилей

Когда стиль нужно применить не к одной странице, а ко всему сайту, лучше управлять таблицами стилей путем их полного перемещения из веб-страниц в отдельные файлы с последующим импортом той таблицы, которая вам нужна. Тем самым предоставляется возможность применения разных таблиц стилей к разным форматам подачи информации (например, в варианте просматриваемой веб-страницы и в варианте вывода на печать) без изменения HTML.

Этого можно достичь двумя различными способами, первый из которых заключается в использовании CSS-директивы `@import`:

```
<style>
  @import url('styles.css');
</style>
```

Эта инструкция заставляет браузер извлечь таблицу стилей с именем `styles.css`. Гибкость, присущая команде `@import`, позволяет создавать таблицы стилей, которые сами импортируют другие таблицы стилей, а те, в свою очередь, могут импортировать другие таблицы и т. д. А вот теги `<style>` и `</style>` при вызове внешних таблиц стилей из других таблиц не нужны, и их присутствие сделает инструкцию неработоспособной.

Импортирование CSS из HTML-кода

Включить таблицу стилей можно также с помощью HTML-тега `<link>`:

```
<link rel='stylesheet' href='styles.css'>
```

Результат будет точно таким же, как и при использовании директивы `@import`, но `<link>` является тегом, применяемым только в HTML, и не относится к стилевым директивам, поэтому он не может задействоваться в одной таблице стилей для импорта другой такой таблицы. Он также не может помещаться внутри пары тегов `<style>...</style>`.

Точно так же, как в CSS, можно использовать несколько директив `@import` для включения в состав таблицы стилей нескольких внешних таблиц, в коде HTML можно применять любое нужное количество элементов, задействующих теги `<link>`.

Встроенные настройки стиля

Можно также выполнять индивидуальные настройки или заменять конкретные стили, вставляя объявления стилей непосредственно в код HTML следующим образом (в данном случае внутри тегов задается курсивный текст синего цвета):

```
<div style='font-style:italic; color:blue;'>Всем привет</div>
```

Но подобные настройки стоит отложить до тех пор, пока не сложатся самые крайние обстоятельства, поскольку они нарушают принцип отделения содержимого от представления.

Идентификаторы (ID)

Более удачным решением для настроек стиля отдельно взятого элемента является назначение формирующему его HTML-коду идентификатора:

```
<div id='welcome'>Всем привет</div>
```

Тем самым устанавливается, что содержимое `<div>`-контейнера с идентификатором, имеющим значение `welcome`, должно иметь применяемый к нему стиль, который

определен в стилевых настройках `welcome`. Соответствующая CSS-инструкция для этого может иметь такой вид:

```
#welcome { font-style:italic; color:blue; }
```

Обратите внимание на использование символа решетки (#), который указывает на то, что эта инструкция задает стилевые настройки только для идентификатора по имени welcome.

Классы

Значение элемента `id` на веб-странице должно быть уникальным, так как именно это позволяет служить ему в качестве идентификатора. Если нужно применить один и тот же стиль ко многим элементам, не следует давать каждому из них особый идентификатор, поскольку для управления всеми этими элементами можно указать класс:

```
<div class='welcome'>Привет</div>
```

Тем самым утверждается, что стиль, определенный в классе `iblue`, должен применяться к содержимому данного элемента (и любых других элементов, относящихся к этому классу). При использовании класса можно либо в заголовке страницы, либо во внешней таблице стилей задействовать для настройки стилей класса следующее правило:

```
.welcome { font-style:italic; color:blue; }
```

Вместо использования символа решетки (`#`), который закреплен за идентификаторами (`ID`), инструкции, относящиеся к классу, предваряются символом точки (`.`).

Точки с запятой

В CSS точки с запятой применяются в качестве разделителей нескольких инструкций CSS, расположенных в одной и той же строке. Но при наличии в правиле только одной инструкции (или при встраивании настройки стиля в HTML-теге) точку с запятой можно опустить, и то же самое можно сделать в отношении последней инструкции в группе.

Но чтобы при использовании CSS избавиться от ошибок, которые трудно будет распознать, можно взять за правило использовать точку с запятой после каждой настройки CSS. В дальнейшем их можно копировать и вставлять или же изменять свойства, не заботясь об удалении точек с запятой там, где они в принципе не нужны, или о добавлении их туда, где они необходимы.

Правила CSS

Каждая инструкция в CSS-правиле начинается с *селектора*, являющегося элементом, к которому будет применяться правило. Например, в следующем назначении `hi` является селектором, для которого задается размер шрифта на 240 % больше, чем у используемого по умолчанию:

```
hi { font-size:240%; }
```

`font-size` является *свойством*. Задавая для принадлежащего селектору `hi` свойства `font-size` значение `240%`, мы гарантируем, что содержимое всех пар тегов `<h1>...</h1>` будет отображено с размером шрифта, превосходящим на 240 % исходный размер. Все изменения в правиле должны быть внутри символов `{` и `}`, следующих за селектором. В `font-size:240%;` та часть, которая находится перед `:` (двоеточием), является свойством, а все остальное является применяемым к нему значением.

И наконец, следует точка с запятой (`;`), завершающая инструкцию. В данном примере, поскольку `font-size` является последним свойством правила, точка с запятой не требуется (но она должна присутствовать, если за этим свойством будет задано значение еще одного свойства).

Множественные задания стиля

Задать несколько стилевых настроек можно двумя разными способами. Можно объединить их в одной строке:

```
hi { font-size:240%; color:blue; }
```

Здесь добавлено второе задание стиля, изменяющее цвет всех заголовков, задаваемых тегом `<hi>`, на синий. Можно также расположить задания построчно:

```
hi { font-size:240%;
color:blue; }
```

Или же можно разнести задания еще дальше, расположив столбцами по двоеточиям:

```
hi {
  font-size :240%;
  color     :blue;
}
```

Тогда будет проще заметить, где начинается каждый новый набор правил, поскольку селектор всегда находится в первом столбце, а следующие за ним задания аккуратно выстраиваются благодаря одинаковому горизонтальному смещению всех значений свойств. В предыдущих примерах замыкающие точки с запятой не нужны, но если придется объединять какие-нибудь подобные группы инструк-

ций в одну строку, то при наличии всех точек с запятой это можно будет сделать довольно быстро.

Один и тот же селектор можно указывать произвольное количество раз, и CSS будет объединять все свойства. Иными словами, предыдущему примеру можно также придать следующий вид:

```
hi { font-size: 240%; }
hi { color    : blue; }
```

Каких-либо правильных или неправильных способов раскладки кода CSS не существует, но я рекомендую вам по крайней мере стараться соблюдать единообразие в построении каждого блока CSS, чтобы в нем можно было разобраться с первого взгляда.

А что произойдет, если задать одно и то же свойство для одного и того же селектора дважды?

```
hi { color : red; }
hi { color : blue; }
```

Будет применено последнее заданное свойство, в данном случае то, которое имеет значение `blue`. Повторять в одном файле одно и то же свойство для одного и того же селектора было бы бессмысленно, но такие повторения часто бывают при реальном использовании веб-страниц, когда для них применяются сразу несколько стилей. Это и есть одно из ценных свойств CSS, которое называется *каскадированием*.

Комментарии

CSS-правила желательно прокомментировать: пусть даже не все или не основную их часть, а только главную группу инструкций. Это делается путем размещения комментариев внутри пары следующих тегов:

```
/* Это комментарий CSS */
```

Комментарий можно развернуть и на несколько строк:

```
/*
  Много-
  строчный
  комментарий
*/
```

При использовании многострочного комментария нужно иметь в виду, что в них нельзя вкладывать однострочные (или любые другие) комментарии. Это может привести к непредсказуемым ошибкам.

Типы стилей

Существует несколько разных типов стилей, начиная с исходных стилей, установленных в вашем браузере (и любых пользовательских стилей, которые вы можете применить в своем браузере, чтобы переопределить исходные значения), продолжая вложенными или встроенными стилями и заканчивая внешними таблицами стилей. Для стилей, которые были определены, действует иерархическая последовательность выполнения, направленная снизу вверх.

Более подробно все касающееся понятия *каскадности* таблиц стилей будет рассмотрено далее в разделе «Каскадность CSS». Но прежде чем перейти к деталям, будет полезно предложить вам краткое введение.

Исходные стили

В браузере применяется задание исходных стилей, имеющих самый низкий уровень приоритета. Этот набор стилей создается на тот случай, когда у веб-страницы нет определений каких-нибудь других стилей. Он предназначен для использования в качестве общего набора стилей, достаточно корректно отображаемого в большинстве случаев.

До создания CSS это были только стили, применяемые к документу, и лишь небольшая часть этих стилей могла быть изменена веб-страницей (например, внешний вид шрифта, его цвет и размер плюс несколько аргументов, относящихся к размеру элементов).

Пользовательские стили

У пользовательских стилей следующий высочайший уровень приоритета. Они поддерживаются большинством современных браузеров, но каждым из них реализуются по-разному, поэтому сегодня для создания своих собственных предпочитаемых стилей оформления просматриваемой информации проще всего воспользоваться таким дополнительным модулем, как Stylish (http://userstyles.org/). Он доступен для большинства наиболее популярных браузеров, за исключением Microsoft Internet Explorer и Edge, для которых придется загрузить свою собственную таблицу стилей через меню свойств обозревателя Internet Options.

Если вы хотите узнать, как создавать собственные исходные настройки стилей для просмотра веб-страниц, воспользуйтесь какой-нибудь поисковой системой и введите в нее название вашего браузера и далее слова «пользовательские стили» (user styles). На рис. 18.2 показано окно выбора таблицы пользовательских стилей Microsoft Internet Explorer.

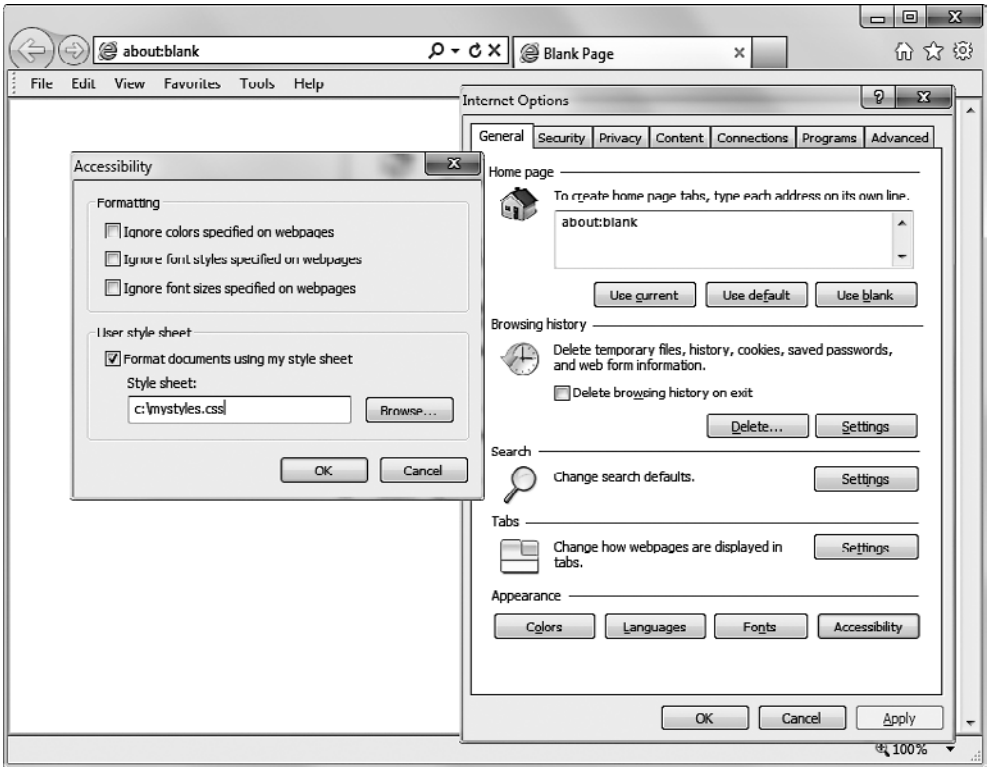

Рис. 18.2. Применение пользовательской таблицы стилей для Internet Explorer

Если задан пользовательский стиль, который уже был определен в качестве исходного стиля браузера, то пользовательский стиль заменит исходный стиль браузера. Любые стили, не определенные в пользовательской таблице стилей, сохранят свои исходные значения, установленные в браузере.

Внешние таблицы стилей

К следующему типу относятся стили, которые задаются во внешней таблице стилей и заменяют любые стили, заданные как пользователем, так и браузером. Внешние таблицы стилей являются рекомендуемым способом создания ваших стилей, поскольку вы можете создавать разные таблицы стилей для разных целей, например для общего использования в Интернете, для просмотра страниц в браузерах мобильных устройств с небольшими экранами, для получения распечатки и т. д. Нужно будет просто применить при создании веб-страницы один требуемый набор стилей для каждого типа носителя информации.

Внутренние стили

Затем следуют внутренние стили, создаваемые внутри тегов <style>...</style>, которые имеют более высокий уровень приоритета над всеми предыдущими типами стилей. Но с этого момента принцип разделения стилевого оформления и содержимого начинает нарушаться, поскольку любые внешние таблицы стилей, загруженные в то же самое время, будут иметь более низкий уровень приоритета.

Внедренные стили

И наконец, рассмотрим внедренные стили, представляющие собой назначение свойства непосредственно элементу. Они также имеют наивысший уровень приоритета над любым типом стилей, и их использование имеет следующий вид:

```
<a href="http://google.com" style="color:green;">Посетите Google</a>
```

В этом примере определяемая ссылка будет отображена зеленым цветом, независимо от любых исходных или других цветовых настроек, применяемых любой другой таблицей стилей либо непосредственно к этой ссылке, либо общим порядком ко всем ссылкам.

При использовании этого типа образования стилей вы нарушаете отделение разметки от содержимого, поэтому применять подобные решения рекомендуется только при крайней необходимости.

Селекторы CSS

Средства доступа к одному или нескольким элементам называются *селекцией*, а та часть правила CSS, которая этим занимается, известна как *селектор*. И, как вы уже, наверное, догадались, существует множество разнообразных селекторов.

Селектор типа

Селектор типа работает в отношении типов HTML-элементов, например или <i>. Следующее правило, к примеру, обеспечивает полное выравнивание всего текста, находящегося между тегами ... :

```
p { text-align:justify; }
```

Селектор потомков

Селекторы потомков позволяют применять стили к элементам, содержащимся внутри других элементов. Например, следующее правило настраивает вывод всего текста внутри тегов `<b>...</b>` красным цветом, но только если эти теги окажутся внутри тегов `<p>...</p>` (как в этом случае: `<p><b>Hello</b> there</p>`):

```
p b { color:red; }
```

Вложенность селекторов потомков может продолжаться до бесконечности, поэтому следующее правило является вполне приемлемым для того, чтобы полужирный текст внутри элемента маркированного списка выводился синим цветом:

```
ul li b { color:blue; }
```

В качестве практического примера представим себе, что нужно использовать другую систему нумерации, отличающуюся от исходной для пронумерованного списка, вложенного в другой пронумерованный список. Этого можно достичь следующим способом, заменяющим исходную нумерацию (начинающуюся с 1) буквами в нижнем регистре (начинающимися с a):

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol ol { list-style-type:lower-alpha; }}
    </style>
  </head>
  <body>
    <ol>
      <li>Один</li>
      <li>Два</li>
      <li>Три
        <ol>
          <li>Один</li>
          <li>Два</li>
          <li>Три</li>
        </ol>
      </li>
    </ol>
  </body>
</html>
```

Загрузка этого кода HTML в браузер даст следующий результат — как видите, элементы второго списка отображаются иначе:

```
1. Один
2. Два
3. Три
  a. Один
  b. Два
  c. Три
```

Селектор дочерних элементов

Селектор дочерних элементов похож на селектор потомков, но он еще больше конкретизирует область применения стиля, выбирая только те элементы, которые являются непосредственными дочерними элементами другого элемента. Например, следующий код использует селектор потомков, который изменит цвет любого текста, выделенного полужирным шрифтом, внутри абзаца на красный, даже если сам полужирный текст находится внутри выделения курсивом (подобно следующему коду: `<p><i><b>Привет</b> всем</i></p>`):

```
p b { color:red; }
```

В данном случае слово `Привет` отображается красным цветом. Но когда этот более общий тип поведения не требуется, чтобы сузить область применения селектора еще больше, может использоваться селектор дочерних элементов. Например, в следующем правиле знак «больше чем» вставлен с целью создания селектора дочерних элементов, устанавливающего красный цвет для текста, выделенного полужирным шрифтом, только в том случае, если элемент будет непосредственным дочерним элементом абзаца и внутри не содержится другой элемент:

```
p > b { color:red; }
```

Теперь слово `Привет` не изменит свой цвет, потому что оно не является непосредственным дочерним элементом абзаца.

В качестве практического примера представим себе, что нужно применить стиль только к тем `<li>`-элементам, которые являются непосредственными дочерними элементами `<ol>`-элементов. Добиться этого можно с помощью следующего кода, где на `<li>`-элементы, являющиеся непосредственными дочерними элементами `<ul>`-элементов, стиль применяться не будет:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol > li { font-weight:bold; }
    </style>
  </head>
  <body>
    <ol>
      <li>Один</li>
      <li>Два</li>
      <li>Три</li>
    </ol>
    <ul>
      <li>Один</li>
      <li>Два</li>
      <li>Три</li>
    </ul>
  </body>
</html>
```

Результат загрузки этого HTML-кода в браузер будет иметь следующий вид:

1. Один
2. Два
3. Три

- Один
- Два
- Три

Селектор элементов, имеющих идентификатор

Если у элемента есть имя-идентификатор (наподобие следующего: `<div id='mydiv'>`), к нему можно обратиться из CSS напрямую следующим способом, выделяющим весь текст в названном элементе курсивом:

```
#mydiv { font-style:italic; }
```

Каждый идентификатор может использоваться в документе только один раз, поэтому только первое найденное появление идентификатора приведет к применению нового значения того или иного свойства, заданного правилом CSS. Но в CSS можно непосредственно ссылаться на любые идентификаторы, имеющие одинаковые имена, если они появляются в элементах разного типа:

```
<div id='myid'>Привет</div> <span id='myid'>Привет</span>
```

Поскольку идентификаторы обычно применяются только к уникальным элементам, следующее правило будет задавать подчеркивание только первому появлению `myid`:

```
#myid { text-decoration:underline; }
```

Но можно добиться того, чтобы правило в CSS применялось к обоим появлениям данного идентификатора:

```
span#myid { text-decoration:underline; }
div#myid  { text-decoration:underline; }
```

Или в сокращенной записи (см. далее раздел «Групповая селекция»):

```
span#myid,div#myid { text-decoration:underline; }
```

Я не рекомендую использовать такую форму селекции, поскольку она возводит барьеры в использовании JavaScript. Любой код JavaScript, который также должен обращаться к данным элементам, не сможет с этим справиться, так как широко применяемая функция getElementById вернет только первое появление элемента с таким идентификатором. Для ссылки на любые другие экземпляры программе придется перебрать весь список элементов в документе, что является куда более сложной задачей. Лучше всегда выбирать для идентификаторов только уникальные имена.

Селектор класса

Когда на странице имеются элементы, для которых нужно применить один и тот же стиль, всем этим элементам можно задать одно и то же имя класса (например: `<span class='myclass'>`), а затем создать единое правило для одновременного изменения всех этих элементов, как в следующем правиле. Оно создает смещение левого края на 10 пикселов для всех элементов, которые используют данный класс:

```
.myclass { margin-left:10px; }
```

В современных браузерах могут быть HTML-элементы, использующие более одного класса, если имена классов разделить пробелами, например: `<span class='class1 class2 class3'>`. Но следует запомнить, что некоторые очень старые браузеры допускают применение в аргументе `class` только одного имени.

Вы можете сузить область действия класса, указав тип элементов, к которым должно применяться правило. Например, следующее правило применяет настройки только к абзацам, использующим класс `main`:

```
p.main { text-indent:30px; }
```

В данном примере только те абзацы, которые используют класс `main` (как этот: `<p class="main">`), получат новое значение свойства. На любые другие типы элементов, которые могут применять этот класс (такие как `<div class="main">`), это правило распространяться не будет.

Селектор атрибутов

Многие HTML-теги поддерживают атрибуты, и использование селектора данного типа может избавить вас от применения идентификаторов и классов для ссылок на элементы, задаваемые этими тегами. Например, можно непосредственно сослаться на атрибуты следующим образом, установив для всех элементов, задействующих атрибут `type="submit"`, ширину, равную 100 пикселам:

```
[type="submit"] { width:100px; }
```

Если нужно ограничить область действия селектора, к примеру, до элементов ввода, принадлежащих форме и имеющих это значение атрибута типа, можно вместо предыдущего воспользоваться следующим правилом:

```
form input[type="submit"] { width:100px; }
```

Селекторы атрибутов также работают применительно к идентификаторам и классам, например селектор [class~="dassname"] работает точно так же, как и .classname (за исключением того, что у последнего из них более высокий уровень приоритета). Точно таким же образом селектор [id="idname"] может

использоваться вместо селектора идентификатора #idname. Селекторы классов и идентификаторов, предваряемые символами решетки (#) и точки (.), могут рассматриваться в качестве краткой формы селекторов атрибутов, имеющей при этом более высокий уровень приоритета. Оператор ~= определяет соответствие атрибуту, даже если он входит в группу атрибутов, разделенных запятыми.

Универсальный селектор

Групповой символ *, или универсальный селектор, соответствует любому элементу, поэтому следующее правило приведет к полному беспорядку в документе, установив зеленое обрамление для всех его элементов:

```
* { border:1px solid green; }
```

Скорее всего, универсальный селектор будет использоваться не сам по себе, а как часть какого-нибудь составного правила, где он будет весьма эффективен. Например, следующее правило будет применять тот же самый стиль, что предыдущее, но только ко всем абзацам, являющимся подчиненными для того элемента, у которого имеется идентификатор со значением `boxout`, и только в том случае, если они не являются непосредственными дочерними элементами:

```
#boxout * p {border:1px solid green; }
```

Разберемся в том, что здесь происходит. Первым селектором, следующим за `#boxout`, является символ звездочки (*), стало быть, он ссылается на любой элемент внутри объекта `boxout`. Затем следующий селектор `p` сужает фокус селекции, направляя его только на абзацы (что и определяется символом `p`), являющиеся подчиненными элементами, возвращаемыми селектором *. Поэтому данное CSS-правило приводит к выполнению следующих действий (в которых для ссылки на одни и те же вещи я использую взаимозаменяемые понятия *«объект»* и *«элемент»*).

1. Поиск объекта с идентификатором, имеющим значение `boxout`.
2. Поиск всех подчиненных элементов объекта, возвращенного при выполнении действия 1.
3. Поиск всех подчиненных `p`-элементов тех объектов, которые были возвращены при выполнении действия 2, и, поскольку это последний селектор в группе, поиск также всех подчиненных `p`-элементов, подчиняющихся этим подчиненным элементам (и т. д.) того объекта, который был возвращен при выполнении действия 2.
4. Применение стилей, заданных внутри символов { и }, к объектам, возвращенным при выполнении действия 3.

В результате зеленое обрамление применяется только к абзацам, являющимся внучатыми (или правнучатыми и т. д.) элементами основного элемента.

Групповая селекция

При использовании CSS имеется возможность одновременного применения правила более чем к одному элементу, классу или любому другому типу селектора путем разделения селекторов запятыми. Например, следующее правило поместит пунктирную оранжевую линию под всеми абзацами, элементом с идентификатором `idname` и всеми элементами, использующими класс со значением `classname`:

```
p, #idname, .classname { border-bottom:1px dotted orange; }
```

На рис. 18.3 показан результат применения разных селекторов, а рядом — применяемые к ним правила.

Рис. 18.3. Фрагменты кода HTML и применяемые в отношении этих фрагментов правила CSS

Каскадность CSS

Как уже вкратце упоминалось, одной из основополагающих особенностей свойств CSS является их каскадность, благодаря которой они и называются каскадными таблицами стилей (Cascading Style Sheets). Но что это означает?

Каскадирование — это метод, используемый для решения потенциальных конфликтов между различными типами стилей, поддерживаемых браузером, и применения их в порядке приоритетности в зависимости от создателя стилей, от метода, который использован для создания стиля, и от типов выбранных свойств.

Создатели таблиц стилей

Все современные браузеры поддерживают три основных типа таблиц стилей. В порядке приоритетности сверху вниз они располагаются следующим образом.

1. Созданные автором документа.
2. Созданные пользователем.
3. Созданные браузером.

Эти три набора таблиц стилей обрабатываются в обратном порядке. Сначала к документу применяются исходные настройки браузера. Без них веб-страницы, не использующие таблицы стилей, выглядели бы ужасно. Они включают внешний вид, размер и цвет шрифта, интервалы между элементами, обрамление и отступы в таблицах и все остальные разумные стандарты, ожидаемые пользователем.

Затем, если пользователь создал какие-нибудь стили, которые предпочитает применять в качестве стандартных, эти стили заменяют исходные стили браузера, с которыми они могут конфликтовать.

И наконец, применяются любые стили, созданные автором текущего документа, заменяя любые стили, либо созданные в качестве исходных стилей браузера, либо созданные пользователем.

Методы создания таблиц стилей

Таблицы стилей могут создаваться с помощью трех различных методов. Если расположить их в порядке приоритетности сверху вниз, получится следующий список.

1. Внедренные стили.
2. Встроенная таблица стилей.
3. Внешняя таблица стилей.

Эти методы создания таблиц стилей также применяются в порядке, обратном порядку их приоритетности. Поэтому сначала обрабатываются все внешние таблицы стилей и к документу применяются их стили.

Затем обрабатываются любые встроенные стили (которые находятся внутри тегов `<style>...</style>`). Все, что конфликтует с внешними правилами, получает приоритет и заменяет эти правила.

И наконец, наивысший приоритет получают любые стили, применяемые непосредственно к элементу в качестве внедренного стиля (такие как `<div style="...">...</div>`), которые заменяют все предыдущие заданные свойства.

Селекторы таблиц стилей

Существует три разных способа выбора стилизуемых элементов. В порядке убывания приоритетности их список имеет такой вид.

1. Обращение по индивидуальному идентификатору, или селектор атрибутов.
2. Обращение в группах по классу.
3. Обращение по тегам элементов.

Селекторы обрабатываются согласно количеству и типам элементов, подпадающих под правило, которое несколько отличается от предыдущих двух правил разрешения конфликтов. Причина состоит в том, что правила не должны сразу применяться только к одному типу селектора и могут иметь отношение к разным селекторам.

Таким образом, метод, необходимый для определения уровня приоритета правил, может содержать любую комбинацию селекторов. Это делается вычислением специфики каждого правила путем выстраивания их в порядке убывания области действия.

Вычисление специфики

Специфика правила вычисляется путем создания трехкомпонентных чисел на основе типов селекторов в показанном выше списке. Эти составные числа сначала выглядят как `[0,0,0]`. При обработке правила каждый селектор, который ссылается на идентификатор, увеличивает первое число на единицу, и составное число приобретает вид `[1,0,0]`.

Посмотрим на следующее правило. У него имеется семь ссылок, три из которых — ID-ссылки (`#heading`, `#main` и `#menu`), поэтому составное число приобретает вид `[3,0,0]`:

```
#heading #main #menu .text .quote p span {
  // Здесь размещаются правила;
}
```

Количество классов в селекторе помещается во второй части составного числа. В данном примере два класса (`.text` и `.quote`), поэтому составное число приобретает вид `[3,2,0]`.

И наконец, вычисляется количество селекторов, ссылающихся на теги элементов, и результат помещается в последнюю часть составного числа. В нашем примере таких селекторов два (`p` и `span`), поэтому составное число приобретает следующий окончательный вид: `[3,2,2]`.

Этого вполне достаточно для сравнения специфики данного правила с другими спецификами. В случаях, подобных этому, когда в составном числе набирается девять или меньше селекторов каждого типа, его можно преобразовать непосред-

ственно в десятичное число, в нашем случае это 322. Правила с меньшим числом, чем это, будут иметь меньший приоритет, а правила с более высоким числом будут иметь больший приоритет. Когда у двух правил будет одно и то же значение, выиграет последнее из применявшихся.

Предположим, к примеру, что у нас также есть следующее правило:

```
#heading #main .text .quote .news p span {
  // Здесь размещаются правила;
}
```

Здесь, несмотря на то что ссылка также идет на семь элементов, имеется только две ID-ссылки, но три ссылки на классы, в результате чего получается составное число `[2,3,2]`. Поскольку 322 больше, чем 232, у первого примера приоритет выше, чем у второго.

Использование другой системы счисления

Когда в составном числе набирается более девяти типов селекторов, нужно переходить на более старшую систему счисления. Например, составное число `[11,7,19]` не подлежит преобразованию в десятичное простым объединением трех частей.

Вместо этого его можно преобразовать в число с более высоким основанием системы счисления, например с основанием 20 (или выше, если будет больше 19 селекторов любого типа).

Для этого нужно умножить все три части и сложить результаты, как показано ниже, начиная с крайнего справа числа и переходя влево:

```
              20 x 19 = 380
            20x20 x 7 = 2800
       20x20x20 x 11 = 88000
Всего в десятичном виде = 91180
```

Замените значения `20` слева значениями применяемого по необходимости основания. Затем, когда все составные числа набора правил пройдут преобразование из этого основания в десятичное, будет просто определить специфику, а стало быть, и уровень приоритета каждого.

К счастью, процессор CSS делает все это за вас, но понимание принципов данной работы поможет правильно создавать правила и разбираться в тех уровнях приоритета, которые у них будут.

Если предыдущие вычисления кажутся вам слишком сложными, полезно будет усвоить одно простое общее правило, которого можно придерживаться в большинстве случаев: чем меньше подвергаемых изменению элементов и чем более конкретно они указаны, тем более высокий приоритет получает правило.

Одни правила бывают равнее других

Когда два правила задания стилей или более имеют абсолютно одинаковый уровень приоритета, то по умолчанию будет применяться последнее обработанное правило. Но вы можете придать правилу более высокий уровень приоритета по сравнению с другими равными ему правилами, используя объявление `!important`:

```
p { color:#ff0000 !important; }
```

При этом все предыдущие равные настройки заменяются (даже те, в которых используется объявление `!important`), и любые равные правила, обрабатываемые позже, игнорируются. Например, второе из двух следующих правил в обычном случае имело бы приоритет, но из-за применения объявления `!important` в ранее заданном правиле оно игнорируется:

```
p { color:#ff0000 !important; }
p { color:#ffff00 }
```

Пользовательские таблицы стилей могут создаваться для определения исходных стилей браузера, и в них может применяться объявление !important В этом случае пользовательская настройка стиля будет иметь преимущество над аналогичными свойствами, указанными на текущей веб-странице. Но в очень старых браузерах, использующих CSS1, эта особенность не поддерживается. Также следует заметить, что установки пользовательского стиля, не имеющие объявления !important, будут переписаны любыми стилями с этим объявлением, имеющимися на веб-странице.

Разница между элементами Div и Span

Оба элемента — `<div>` и `<span>` — относятся к контейнерам, но некоторые качества у них отличаются. По умолчанию `<div>`-элемент имеет бесконечную ширину (как минимум до края окна браузера), которую можно увидеть, если применить к контейнеру единичное обрамление:

```
<div style="border:1px solid green;">Привет</div>
```

А `<span>`-элемент не шире того текста, который в нем содержится. Поэтому следующий код HTML создаст обрамление только вокруг слова `Привет`, и оно не будет расширяться до правого края браузерного экрана:

```
<span style="border:1px solid green;">Привет</span>
```

Кроме того, `<span>`-элемент сопровождает текст или другие объекты и при их переносе на следующие строки, поэтому может иметь довольно сложное обрамление. Например, в примере 18.2 код CSS использован для создания желтого фона для всех `<div>`-элементов, для создания голубого фона для всех `<span>`-элементов и для добавления обрамления и к тем и к другим, перед тем как создать несколько примеров `<span>`- и `<div>`-блоков.

Пример 18.2. Пример контейнеров и

```
<!DOCTYPE html>
<html>
  <head>
    <title>Пример div и span</title>
    <style>
      div, span { border :1px solid black; }
      div       { background-color:yellow; }
      span      { background-color:cyan;   }
    </style>
  </head>
  <body>
    <div>Этот текст находится внутри тега div</div>
    А этот - нет. <div>А этот снова внутри тега div.</div><br>

    <span> Этот текст находится внутри тега span .</span>
      А этот - нет. <span> А этот снова внутри тега span.</span><br><br>

    <div>Это более объемный текст в теге div, который переносится
      на следующую строку браузера </div><br>

    <span> Это более объемный текст в теге span, который переносится
      на следующую строку браузера </span>
  </body>
</html>
```

Как выполнение кода этого примера выглядит в окне браузера, показано на рис. 18.4. Хотя все в печатной версии книги изображается в серых тонах, на рисунке четко показано, как -элементы расширяются до правого края окна браузера и заставляют следующее за ними содержимое появляться с начала первой доступной позиции ниже себя.

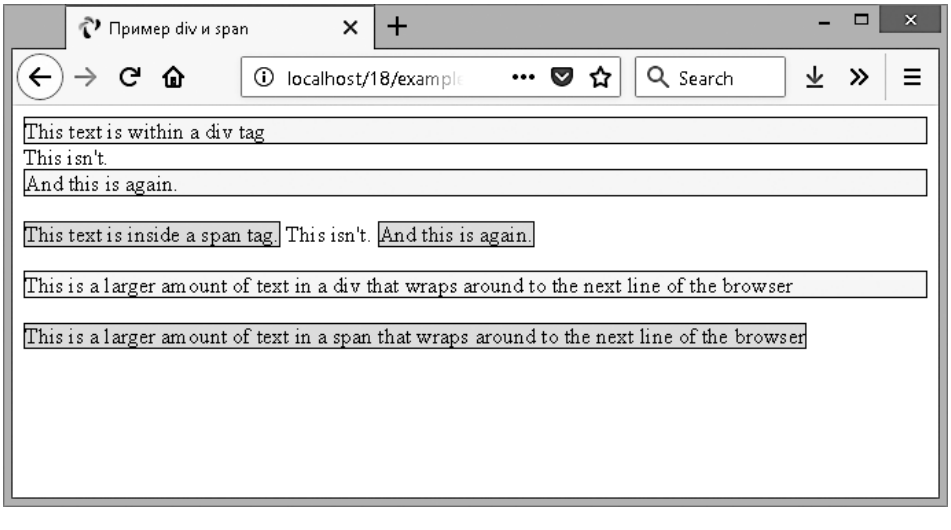

Рис. 18.4. Разнообразные элементы, имеющие разную ширину

На рисунке также видно, как ведут себя -элементы, занимая только то пространство, которое требуется для размещения их содержимого, не заставляя при этом последующее содержимое страницы появляться под ними.

Вдобавок к этому в двух самых нижних примерах, показанных на рисунке, можно увидеть, что, когда -элементы при достижении края экрана переносятся на новую строку, они сохраняют прямоугольную форму, в то время как -элементы просто следуют за потоком содержащегося в них текста (или другого содержимого).

Поскольку -теги могут создавать только прямоугольные контейнеры, они лучше подходят для содержания таких объектов, как изображения, блоки, цитаты и т. д., а контейнеры, создаваемые -тегами, больше подходят для текста или других атрибутов, которые размещаются один за другим на одной линии и должны располагаться слева направо (или в некоторых языках справа налево).

Измерения

CSS поддерживает впечатляющий диапазон различных единиц измерения, позволяя очень точно выкраивать веб-страницы для конкретных значений или относительных размеров. Я обычно пользуюсь следующими единицами измерений (и считаю, что вам они также будут наиболее полезны): пикселами, пунктами, эмами и процентами. Рассмотрим подробнее эти, а также другие единицы измерения.

- *Пиксел* (pixel) — его размер варьируется в соответствии с размерами и глубиной пиксела на пользовательском экране. Один пиксел равен ширине и высоте отдельной точки на экране, поэтому данную единицу измерения лучше всего использовать для экранов, а не для распечаток. Например:

```
.classname { margin:5px; }
```

- *Пункт* (point) — равен по размеру 1/72 дюйма. Эта единица измерения пришла из полиграфии и лучше всего подходит для той среды, но также широко используется и для отображений на экранах. Например:

```
.classname { font-size:14pt; }
```

- *Дюйм* (inch) — равен 72 пунктам и также относится к типу единиц измерения, наиболее приспособленных для организации вывода на печать. Например:

```
.classname { width:3in; }
```

- *Сантиметр* (centimeter) — еще одна единица измерения, которая наиболее пригодна для организации вывода на печать. Один сантиметр немного превышает по размеру 28 пунктов. Например:

```
.classname { height:2cm; }
```

- *Миллиметр* (millimeter) — это 1/10 сантиметра (или почти 3 пункта). Миллиметры являются еще одной единицей измерения, наиболее подходящей для организации вывода на печать. Например:

```
.classname { font-size:5mm; }
```

- *Пика* (pica) — еще одна типографская единица измерения, равная 12 пунктам. Например:

```
.classname { font-size:1pc; }
```

- *Эм* (em) — равен текущему размеру шрифта (ширине латинской буквы m). Это одна из наиболее полезных единиц измерения для CSS, поскольку используется для описания относительных размеров. Например:

```
.classname { font-size:2em; }
```

- *Экс* (ex) — также относится к текущему размеру шрифта. Он равен высоте буквы x нижнего регистра. Это менее популярная единица измерения, которая чаще всего используется в качестве хорошего приблизительного значения, помогающего установить ширину прямоугольного блока, который будет содержать некий текст. Например:

```
.classname { width:20ex; }
```

- *Процент* (percent) — эта единица сродни эму (em) и ровно в 100 раз больше (применительно к шрифту). Если 1 em эквивалентен текущему размеру шрифта, в процентах тот же размер выражается цифрой 100. Когда эта единица не относится к шрифту, она относится к размеру того контейнера, к которому применяется данное свойство. Например:

```
.classname { height:120%; }
```

На рис. 18.5 каждый из этих типов измерений показан по очереди применительно к отображаемому тексту почти одинаковых размеров.

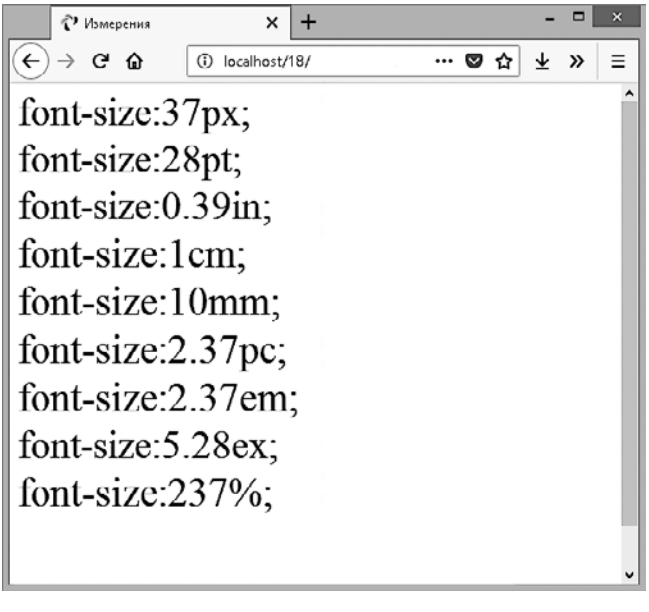

Рис. 18.5. Различные измерения, приводящие примерно к одинаковому результату

Шрифты и оформление

С помощью CSS можно настроить четыре основных свойства шрифта: семейство — `family`, стиль — `style`, размер — `size` и насыщенность — `weight`. Пользуясь этими свойствами, можно точно настроить способ отображения текста в ваших веб-страницах и (или) вывода его на печать.

font-family

Это свойство назначает используемый шрифт. Оно также поддерживает перечисление множества шрифтов в порядке предпочтения слева направо, чтобы стилевое оформление при отсутствии у пользователя установленного предпочитаемого шрифта постепенно переходило в сторону менее предпочитаемых шрифтов. Например, для установки шрифта по умолчанию для абзацев можно воспользоваться следующим CSS-правилом:

```
p { font-family:Verdana, Arial, Helvetica, sans-serif; }
```

Если название шрифта состоит из двух и более слов, его нужно заключить в кавычки:

```
p { font-family:"Times New Roman", Georgia, serif; }
```

Для использования на веб-страницах больше всего подходит такое семейство шрифтов, как Arial, Helvetica, Times New Roman, Times, Courier New и Courier, поскольку все эти шрифты доступны практически во всех браузерах и операционных системах. Шрифты Verdana, Georgia, Comic Sans MS, Trebuchet MS, Arial Black и Impact можно смело применять на Mac и PC, но они могут быть не установлены в других операционных системах, таких как Linux. Другими распространенными, но менее надежными шрифтами являются Palatino, Garamond, Bookman и Avant Garde. Если используется один из менее надежных шрифтов, нужно убедиться, что в ваших настройках CSS предложены один или несколько менее предпочтительных шрифтов, чтобы веб-страницы в отсутствие в браузерах предпочитаемых вами шрифтов шли в их использовании по нисходящей.

На рис. 18.6 показано применение этих двух наборов CSS-правил.

Рис. 18.6. Выбор семейства шрифтов

font-style

С помощью этого свойства можно выбрать вывод шрифта в обычном — `normal`, курсивном — `italic` или наклонном — `oblique` виде. Следующие правила создают три класса (`normal`, `italic` и `oblique`), которые могут применяться к элементам для создания соответствующих эффектов:

```
.normal  { font-style:normal;  }
.italic  { font-style:italic;  }
.oblique { font-style:oblique; }
```

font-size

В предыдущем разделе, касающемся единиц измерения, было рассмотрено множество способов изменения размера шрифта, но все они сводятся к двум основным типам: фиксированному и относительному. Фиксированная настройка похожа на следующее правило, которым для абзацев устанавливается размер шрифта, равный 14 пунктам:

```
p { font-size:14pt; }
```

В качестве альтернативы можно предпочесть работу с текущим размером шрифта по умолчанию, используя его для стилевого решения таких видов текста, как заголовки. В следующих правилах определены относительные размеры некоторых заголовков, где тег `<h4>` начинает с прибавки в 20 % к размеру по умолчанию и каждое возрастание размера задается больше предыдущего на 40 %:

```
h1 { font-size:240%; }
h2 { font-size:200%; }
h3 { font-size:160%; }
h4 { font-size:120%; }
```

На рис. 18.7 показана подборка размеров шрифтов в действии.

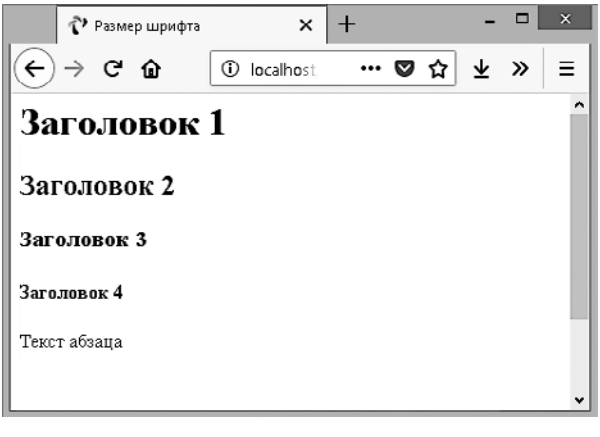

Рис. 18.7. Настройка четырех размеров заголовков и размер абзаца, используемый по умолчанию

font-weight

Используя это свойство, можно задать насыщенность, или степень жирности, шрифта. Оно поддерживает несколько значений, но в основном востребованы `normal` и `bold`:

```
.bold { font-weight:bold; }
```

Управление стилями текста

Независимо от используемого шрифта в способ вывода текста можно внести дополнительные изменения, меняя его оформление — `decoration`, разрядку — `spacing` и выравнивание — `alignment`. Но свойства текста и шрифта перекликаются в том смысле, что курсивный и полужирный текст можно получить, используя свойства `font-style` и `font-weight`, в то время как другой текст, например подчеркнутый, требует применения свойства `text-decoration`.

Оформление

Используя свойство `text-decoration`, можно применить к тексту такие эффекты, как подчеркивание — `underline`, перечеркивание — `line-through`, верхнее подчеркивание — `overline` и мигание — `blink`. Следующее правило создает новый класс по имени `over`, который применяет верхнее подчеркивание к тексту (насыщенность линий, используемых для подчеркивания снизу и сверху и для перечеркивания, будет соответствовать насыщенности шрифта):

```
.over { text-decoration :overline; }
```

На рис. 18.8 можно увидеть подборку стилей, насыщенности и оформления шрифтов.

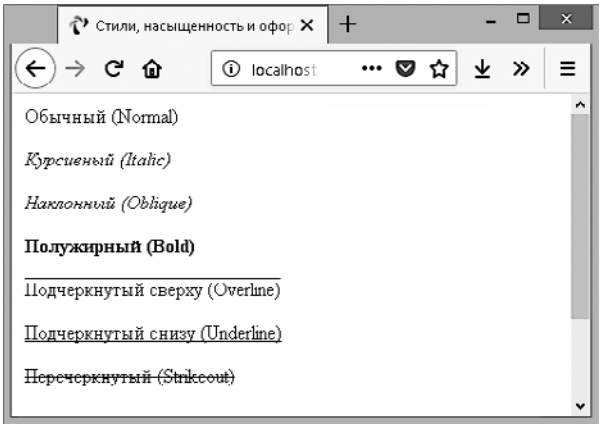

Рис. 18.8. Примеры доступных правил стилевых решений и оформления

Разрядка

Существуют свойства, позволяющие изменить разрядку строк, слов и букв. Например, следующие правила настраивают разрядку строк для абзацев путем изменения свойства `line-height`, делая его на 25 % больше, устанавливают свойство `word-spacing` равным 30 пикселам и разрядку букв 3 пиксела:

```
p {
   line-height :125%;
   word-spacing :30px;
   letter-spacing:3px;
}
```

Процентным значением можно также воспользоваться для настройки значения свойства `word-spacing` или `letter-spacing` с целью уменьшения или увеличения исходной разрядки, применяемой к шрифту, используя значения меньше или больше 100 %, что будет приемлемо как для пропорциональных, так и для непропорциональных (моноширинных) шрифтов.

Выравнивание

В CSS доступны четыре типа выравнивания текста: по левому краю — `left`, по правому краю — `right`, по центру — `center` и по ширине содержимого — `justify`. В следующем правиле текст абзаца изначально настроен на полное выравнивание по ширине:

```
p { text-align:justify; }
```

Преобразование

Для преобразования текста доступны четыре свойства: отсутствие преобразования — `none`, преобразование первых букв слов в заглавные — `capitalize`, преобразование всех букв в заглавные — `uppercase` и преобразование всех букв в строчные — `lowercase`. Следующее правило создает класс по имени `upper`, гарантирующий при его применении вывод всего текста в верхнем регистре:

```
.upper { text-transform:uppercase; }
```

Отступы

С помощью свойства `text-indent` можно создать отступ первой строки блока текста на указанную величину. Следующее правило создает отступ первой строки каждого абзаца на 20 пикселов, но могут быть применены и другие единицы измерений или процентное увеличение:

```
p { text-indent:20px; }
```

На рис. 18.9 к блоку текста было применено следующее правило:

```
p         { line-height    :150%;
            word-spacing   :10px;
            letter-spacing :1px;
}
.justify   { text-align    :justify;   }
.uppercase { text-transform:uppercase; }
.indent    { text-indent  :20px;       }
```

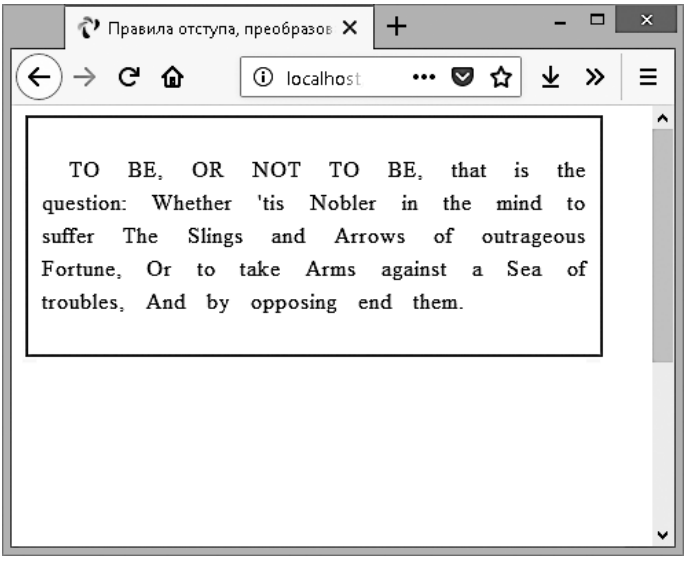

Рис. 18.9. Примененные правила отступа, преобразования в верхний регистр и разрядки

Цвета CSS

Цвета могут применяться к первому плану, а также к фону текста и объектов путем использования свойства цвета — `color` и фонового цвета — `background-color` (или путем предоставления единственного аргумента свойству фона — `background`). Указанный цвет может быть одним из именованных цветов (например, `red` или `blue`), цветом, составленным из трех шестнадцатеричных чисел RGB (например, `#ff0000` или `#0000ff`), или цветом, составленным с использованием CSS-функции `rgb`.

Названия стандартных 16 цветов, определенных организацией по стандартам W3C (http://www.w3.org), следующие: аквамарин — `aqua`, черный — `black`, синий — `blue`, яркий пурпурно-красный, или фуксия, — `fuchsia`, серый — `gray`, зеленый — `green`, яркий светло-зеленый — `lime`, красно-коричневый — `maroon`, темно-синий — `navy`, оливковый — `olive`, фиолетовый — `purple`, красный — `red`, серебристый — `silver`,

зеленовато-голубой — `teal`, белый — `white` и желтый — `yellow`. Следующее правило использует одно из этих названий для установки фонового цвета для объекта с ID, имеющим значение `object`:

```
#object { background-color:silver; }
```

В показанном ниже правиле фоновый цвет текста во всех `<div>`-элементах установлен желтым (поскольку на мониторе шестнадцатеричные уровни `ff` красного плюс `ff` зеленого плюс `00` синего составляют желтый цвет):

```
div { color:#ffff00; }
```

Или, если не хочется работать с шестнадцатеричными числами, можно указать свои три цветовые составляющие с помощью функции `rgb`, как в следующем правиле, которое изменяет фоновый цвет текущего документа на аквамарин:

```
body { background-color:rgb(0, 255, 255); }
```

Если вы не хотите работать в диапазоне 256 уровней для каждого основного цвета, можете вместо них в функции rgb использовать процентный показатель со значениями от 0 до 100, в диапазоне от самого низкого (0) количества до самого высокого (100) основного цвета, например: rgb(58%, 95%, 74%). Можно также для более тонкого управления цветом применять числа с плавающей точкой, например: rgb(23.4%, 67.6%, 15.5%).

Сокращенные цветовые строки

Есть также короткая форма строки шестнадцатеричных чисел, в которой для каждого цвета используется только первое число из каждой двухбайтовой пары. Например, вместо назначения цвета `#fe4692` можно применять `#f49`, где опущены вторые шестнадцатеричные цифры из каждой пары, что равно цветовому значению `#ff4499`.

Получается почти такой же цвет. Подобную запись можно применить там, где точный цвет не нужен. Разница между строками из шести и из трех цифр в том, что первые поддерживают 16 миллионов различных цветов, а вторые — всего 4 тысячи.

Там, где используется такой цвет, как `#883366`, полным эквивалентом ему будет `#836` (поскольку повторяющиеся цифры подразумеваются в сокращенной версии) и для создания одного и того же цвета можно применять любую строку.

Градиенты

Вместо использования сплошного цветового фона можно применить градиент, который будет автоматически переходить от выбранного исходного до выбранного конечного цвета. Градиент лучше использовать в связке с простым цветовым

правилом, чтобы браузеры, не поддерживающие градиенты, отображали хотя бы сплошной цвет.

В примере 18.3 задействуется правило отображения оранжевого градиента (или просто обычного оранжевого цвета в браузерах, не поддерживающих градиенты), как показано в средней части рис. 18.10.

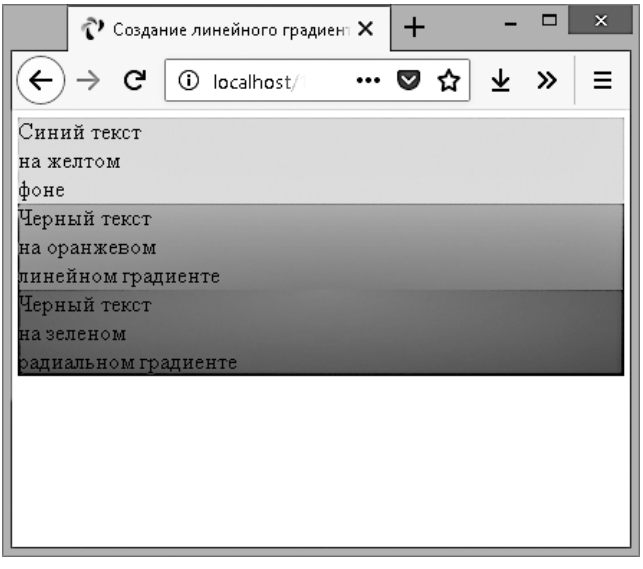

Рис. 18.10. Сплошной цвет фона, а также линейный и радиальный градиенты

Пример 18.3. Создание линейного градиента

```
<!DOCTYPE html>
<html>
  <head>
    <title>Создание линейного градиента</title>
    <style>
      .orangegrad { background:orange;
        background:linear-gradient(top, #fb0, #f50);
        background:-moz-linear-gradient(top, #fb0, #f50);
        background:-webkit-linear-gradient(top, #fb0, #f50);
        background:-o-linear-gradient(top, #fb0, #f50);
        background:-ms-linear-gradient(top, #fb0, #f50); }
    </style>
  </head>
  <body>
    <div class='orangegrad'>Черный текст <br>
      на оранжевом <br>линейном градиенте</div>
  </body>
</html>
```

Как показано в предыдущем примере, многие CSS-правила требуют префиксы, которые предназначены для того или иного браузера, например -moz-, -webkit-, -o- и -ms- (соответственно для браузеров на основе движка Mozilla, таких как Firefox, для браузеров на основе движка WebKit, таких как Apple Safari, Google Chrome и браузеров iOS Android, а также для браузеров Opera и Microsoft). Основные CSS-правила и атрибуты, а также указания на то, где требуются версии, подстроенные под тот или иной браузер, перечислены на сайте http://caniuse.com. Если сомневаетесь, укажите все префиксы, специфичные для браузеров, а также имя стандартного правила, и ваши стили должны будут правильно применяться во всех браузерах (поддерживающих правила, которые вы используете).

Для создания градиента нужно выбрать, где он будет начинаться: вверху (`top`), внизу (`bottom`), слева (`left`), справа (`right`), по центру (`center`) или в любых составных местах, например в левом верхнем углу (`top left`) или от центра вправо (`center right`). Затем следует ввести нужные начальный и конечный цвета и применить правило либо линейного (`linear-gradient`), либо радиального (`radial-gradient`) градиента, обеспечив правила для всех браузеров, на которые вы нацелились.

Вы также можете не только использовать начальный и конечный цвета, но и предоставлять между ними в качестве дополнительных аргументов составляющие *конечные* цвета. Например, если предоставлены пять аргументов, каждый из них будет управлять изменением цвета одной пятой области (в соответствии с его местом в списке аргументов).

Кроме градиентов, к CSS-объектам можно также применять такое свойство, как прозрачность, более подробно рассматриваемое в главе 19.

Позиционирование элементов

Элементы попадают на веб-страницу туда, где они находятся в документе, но могут перемещаться путем изменения свойства позиции элемента от исходной статической (`static`) до абсолютной (`absolute`), относительной (`relative`), прилипчивой (`sticky`) или фиксированной (`fixed`).

Абсолютное позиционирование

Элемент с абсолютным позиционированием удаляется из документа, и любые другие элементы, которые в состоянии это сделать, займут освободившееся пространство. Затем вы можете позиционировать объект в любое нужное место в документе, используя свойства «верх» — `top`, «право» — `right`, «низ» — `bottom` и «лево» — `left`.

Например, для перемещения объекта с ID, имеющим значение `object`, в абсолютное место, находящееся на 100 пикселов ниже начала документа и на 200 пикселов от

левого края, к нему нужно применить следующие правила (вы также можете использовать любые другие единицы измерений, поддерживаемые CSS):

```
#object {
  position:absolute;
  top :100px;
  left :200px;
}
```

Объект будет располагаться либо поверх других элементов, либо за другими перекрывающими его элементами в зависимости от значения, присвоенного свойству `z-index` (которое работает только в отношении позиционированных элементов). По умолчанию это свойство имеет значение `auto`, присваиваемое браузером без вашего участия. Вместо этого вы можете дать данному свойству целочисленное значение (которое может быть и отрицательным):

```
#object {
  position:absolute;
  top :100px;
  left :200px;
  z-index:100;
}
```

После этого объекты будут появляться на экране в порядке от самого низкого и до самого высокого значения их свойства `z-index`, при этом объекты с более высокими значениями станут отображаться поверх объектов с более низкими значениями.

Относительное позиционирование

Подобным образом можно переместить объект относительно того места, которое он занимал бы при обычном ходе формирования документа. Так, например, для перемещения объекта на 10 пикселов вниз и 10 пикселов вправо от его обычного положения нужно воспользоваться следующими правилами:

```
#object {
  position:relative;
  top     :10px;
  left    :10px;
}
```

Фиксированное позиционирование

Заключительные настройки свойства `position` позволяют переместить объект в абсолютное положение, но только внутри окна просмотра текущего браузера. Затем при прокрутке документа объект остается именно там, куда он был помещен, а основной документ будет прокручиваться под ним — это неплохой способ создания док-панелей и других подобных устройств. Для фиксирования объекта в левом верхнем углу браузера нужно воспользоваться следующими правилами:

```
#object {
  position:fixed;
  top     :0px;
  left    :0px;
}
```

В примере 18.4 показано применение к объектам на странице различных значений позиционирования.

Пример 18.4. Применение разных значений позиционирования

```
<!DOCTYPE html>
<html>
  <head>
    <title>Позиционирование</title>
    <style>
      #container {
        position :absolute;
        top      :50px;
        left     :0px;
      }
      #object1 {
        position  :absolute;
        background:pink;
        width     :100px;
        height    :100px;
        top       :0px;
        left      :0px;
      }
      #object2 {
        position  :relative;
        background:lightgreen;
        width     :100px;
        height    :100px;
        top       :0px;
        left      :110px;
      }
      #object3 {
        position  :fixed;
        background:yellow;
        width     :100px;
        height    :100px;
        top       :50px;
        left      :220px;
      }
    </style>
  </head>
  <body>
    <br><br><br><br><br>
    <div id='container'>
      <div id='object1'>Абсолютное позиционирование</div>
      <div id='object2'>Относительное позиционирование</div>
      <div id='object3'>Фиксированное позиционирование</div>
    </div>
  </body>
</html>
```

На рис. 18.11 код примера 18.4 был загружен в браузер и окно браузера было уменьшено по ширине и высоте. В результате появилась необходимость в прокрутке вниз, чтобы можно было увидеть всю веб-страницу.

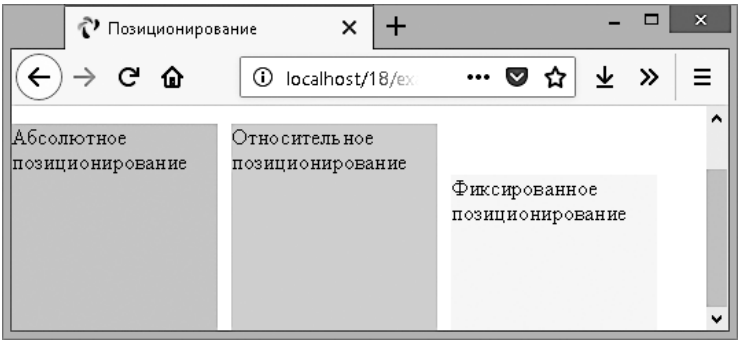

Рис. 18.11. Использование разных значений позиционирования

После прокрутки тут же выяснится, что элемент с фиксированной позицией (`object3`) останется на своем месте, несмотря на прокрутку. Также можно будет наблюдать, что элемент-контейнер (по имени `container`) обладает абсолютным позиционированием и размещается в точности на 50 пикселов ниже, с горизонтальным смещением 0 пикселов, поэтому элемент `object1` (имеющий абсолютное позиционирование внутри элемента `container`) появляется именно в этом месте. А элемент `object2` имеет относительное позиционирование, поэтому он смещен от левой границы элемента `container` на 110 пикселов, выстраиваясь рядом с элементом `object1`.

Показанный на рисунке элемент `object3`, несмотря на то что он в коде HTML появляется внутри элемента `container`, имеет фиксированное позиционирование, в результате чего он фактически совершенно независим от других объектов и не скован границами элемента `container`. Изначально он был настроен на нахождение на одной линии с элементами `object1` и `object2`, но остался на месте, в то время как другие элементы были прокручены вверх по странице, и теперь `object3` смещен ниже этих элементов.

Псевдоклассы

Существуют селекторы и классы, используемые только внутри таблиц стилей и не имеющие каких-либо соответствующих тегов или атрибутов в HTML. Их задача заключается в том, чтобы классифицировать элементы, используя характеристики, отличные от их имен, атрибутов или содержимого, то есть характеристики, которые не могут быть прослежены по дереву документа. К их числу относятся такие псевдоклассы, как `link` и `visited`. Существуют также псевдоэлементы, с помощью

которых осуществляется выбор и которые могут состоять из отдельных элементов, таких как первая строка — `first-line` или первая буква — `first-letter`.

Псевдоклассы и псевдоэлементы отделяются с помощью символа двоеточия (`:`). Например, для создания класса по имени `bigfirst` для выделения первой буквы элемента можно воспользоваться таким правилом:

```
.bigfirst:first-letter {
  font-size:400%;
  float    : left;
}
```

Когда класс `bigfirst` применится к элементу, первая буква будет отображаться сильно увеличенной, а остальной текст будет показан в обычном размере, аккуратно ее обтекая (благодаря свойству `float`), как будто первая буква является изображением или другим объектом. В число псевдоклассов входят `hover`, `link`, `active` и `visited`. Все они наиболее полезны применительно к anchor-элементам, как показано в следующих правилах, которые устанавливают для ссылок в качестве исходного синий цвет, а для посещенных ссылок — светло-синий:

```
a:link    { color:blue;      }
a:visited { color:lightblue; }
```

Следующие правила интересны тем, что в них используется псевдокласс `hover`, поэтому они применяются только при нахождении указателя мыши над элементом. В этом примере они изменяют в ссылке цвет текста на белый на красном фоне, предоставляя динамический эффект, который можно было бы ожидать только от использования кода JavaScript:

```
a:hover {
  color     :white;
  background:red;
}
```

Здесь вместо более длинного свойства цвета фона `background-color` я использовал свойство фона `background` с единственным аргументом.

Псевдокласс `active` также имеет динамический характер, выражающийся в том, что он влияет на изменение ссылки в промежутке времени между щелчком кнопкой мыши и освобождением этой кнопки, как в следующем правиле, изменяющем цвет ссылки на темно-синий:

```
a:active { color:darkblue; }
```

Еще одним интересным динамическим псевдоклассом является `focus`, который применяется, только когда элемент получает фокус путем выбора его пользователем с помощью клавиатуры или мыши. Следующее правило применяет универсальный селектор, чтобы всегда помещать светло-серую пунктирную границу толщиной 2 пиксела вокруг объекта, имеющего фокус:

```
*:focus { border:2px dotted #888888; }
```

Здесь рассматриваются вопросы применительно к традиционной веб-разработке, а не к разработке для мобильных устройств с сенсорными экранами, речь о которой пойдет в главе 22, в которой будет рассмотрена работа с библиотекой jQuery Mobile.

Как показано на рис. 18.12, код примера 18.5 выводит две ссылки и поле ввода. Первая ссылка показана серым цветом, поскольку она уже посещалась в этом браузере, а вторая ссылка еще не посещалась и показана синим цветом. Была нажата клавиша Tab, и фокусом ввода теперь служит поле ввода, поэтому цвет его фона поменялся на желтый. Когда будет выполнен щелчок кнопкой мыши на любой из ссылок, она отобразится фиолетовым цветом, а когда над ней будет проходить указатель мыши, она будет показана красным цветом.

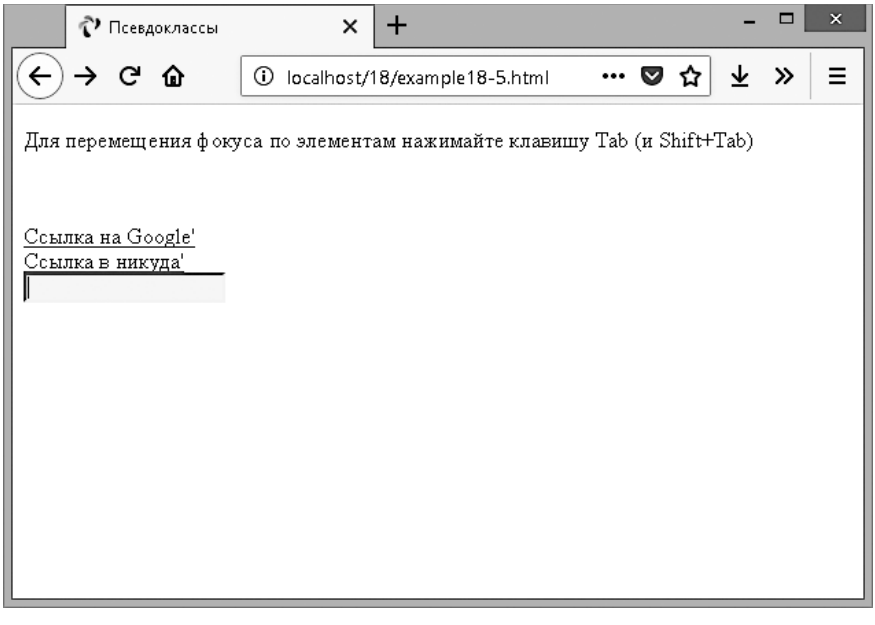

Рис. 18.12. Псевдоклассы, примененные к подборке элементов

Пример 18.5. Псевдоклассы link и focus

```
<!DOCTYPE html>
<html>
  <head>
    <title>Псевдоклассы</title>
    <style>
      a:link { color:blue; }
      a:visited { color:gray; }
      a:hover { color:red; }
```

```
      a:active { color:purple; }
      *:focus { background:yellow; }
    </style>
  </head>
  <body>
    <p>Для перемещения фокуса по элементам нажимайте
       клавишу Tab (и Shift+Tab)</p> <br> <br>
    <a href='http://google.com'>Ссылка на Google'</a><br>
    <a href='nowhere'>Ссылка в никуда'</a><br>
    <input type='text'>
  </body>
</html>
```

Доступны и другие псевдоклассы, дополнительную информацию о них можно получить на сайте http://tinyurl.com/pseudodasses.

Применять псевдокласс focus к универсальному селектору *, как показано в этом примере, нужно с большой осторожностью — Internet Explorer воспринимает документ, не имеющий фокуса, как уже имеющий фокус, который применяется ко всей веб-странице. В этом случае будет желтой вся страница, пока пользователь не нажмет клавишу Tab или же фокус не перейдет на один из элементов страницы.

Сокращенная запись правил

Для экономии пространства группы родственных CSS-свойств могут объединяться в простое сокращенное назначение. Например, я уже несколько раз использовал сокращение для создания границы, как в правиле `focus` в предыдущем разделе:

```
*:focus { border:2px dotted #ff8800; }
```

На самом деле это сокращенное объединение такого набора правил:

```
*:focus {
  border-width:2px;
  border-style:dotted;
  border-color:#ff8800;
}
```

При использовании сокращенной записи правила нужно лишь применить свойства к тому пункту, у которого следует изменить значения. Для установки только ширины и стиля границы без изменения ее цвета можно также использовать следующее правило:

```
*:focus { border:2px dotted; }
```

Порядок размещения свойств в сокращенной записи правила может играть важную роль, и их неправильная расстановка приводит обычно к неожиданным результатам. Поскольку изложить многочисленные подробности в данной главе не представляется возможным, при необходимости воспользоваться сокращенной записью CSS вы сможете найти описание свойств, задаваемых по умолчанию, и порядок их применения в руководстве по CSS или в какой-нибудь поисковой системе. Для начала могу порекомендовать изучение статьи Трентона Мосса (Trenton Moss), опубликованной по адресу https://www.webcredible.com/blog/css-shorthand-properties/.

Модель блока и макет страницы

Свойства CSS, влияющие на макет страницы, основаны на модели блока, вложенном наборе свойств, окружающем элемент. Фактически такие свойства есть (или могут быть) у всех элементов, включая тело документа, чье поле вы можете (к примеру) удалить с помощью следующего правила:

```
body { margin:0px; }
```

Модель блока объекта начинается снаружи, с поля объекта. Внутри него находится граница, затем следует отступ содержимого от границы. И наконец, идет содержимое объекта.

Если приобрести навыки работы с моделью блока, то можно существенно продвинуться на пути создания профессионально спланированных страниц, поскольку даже только эти свойства во многом определяют стилевое оформление страницы.

Установка полей

Поле является самым крайним уровнем модели блока. Оно отделяет элементы друг от друга и требует разумного использования. Предположим, к примеру, что вы решили выбрать по умолчанию поле 10 пикселов вокруг каждого из нескольких элементов. Именно на этот промежуток между элементами вы станете рассчитывать, располагая два элемента друг над другом, но, если у каждого из них будет поле в 10 пикселов, будет ли этот промежуток составлять 20 пикселов?

На самом деле CSS устраняют эту потенциальную проблему: когда два элемента с полями позиционируются непосредственно один над другим, для отделения их друг от друга используется только самое большое из двух полей. Если оба поля имеют одинаковую ширину, применяется только одна ширина. Благодаря этому вы, скорее всего, добьетесь желаемого результата. Но при этом имейте в виду, что поля элементов с заданным абсолютным позиционированием или встраиваемых элементов не подвергаются поглощению другими полями.

Поля элемента могут быть изменены целиком с помощью свойства `margin` или отдельно друг от друга с помощью свойств `margin-left`, `margin-top`, `margin-right` и `margin-bottom`. При установке свойства `margin` можно предоставить один, два, три или четыре аргумента, в результате чего получится эффект, прокомментированный в следующих правилах:

```
/* Установка всех полей шириной 1 пиксел */
margin:1px;

/* Установка верхнего и нижнего полей шириной 1 пиксел, а левого
и правого - 2 пиксела */
margin:1px 2px;

/* Установка верхнего поля шириной 1 пиксел, левого и правого - 2 пиксела
и нижнего - 3 пиксела */
margin:1px 2px 3px;

/* Установка верхнего поля шириной 1 пиксел, правого - 2, нижнего - 3
и левого - 4 пиксела */
margin:1px 2px 3px 4px;
```

В примере 18.6 правило свойства `margin` (выделенное в коде полужирным шрифтом) применяется к прямоугольным элементам, помещенным внутри элемента `table`. На рис. 18.13 показан результат выполнения кода этого примера после его загрузки в браузер. Размер таблицы не задан, поэтому она будет просто охватывать как можно плотнее внутренний `<div>`-элемент. Вследствие этого сверху будет поле шириной 10 пикселов, справа — поле шириной 20 пикселов, снизу — поле шириной 30 пикселов и слева — поле шириной 40 пикселов.

Пример 18.6. Порядок применения полей

```
<!DOCTYPE html>
<html>
  <head>
    <title>Поля CSS</title>
    <style>
      #object1 {
        background  :lightgreen;
        border-style:solid;
        border-width:1px;
        font-family :Courier New;
        font-size   :9px;
        width       :100px;
        height      :100px;
        padding     :5px;
        margin      :10px 20px 30px 40px;
      }
      table {
        padding     :0;
        border      :1px solid black;
```

```
        background  :cyan;
      }
    </style>
  </head>
  <body>
    <table>
      <tr>
        <td>
          div id='object1'>margin:<br>10px 20px 30px 40px;</div>
        </td>
      </tr>
    </table>
  </body>
</html>
```

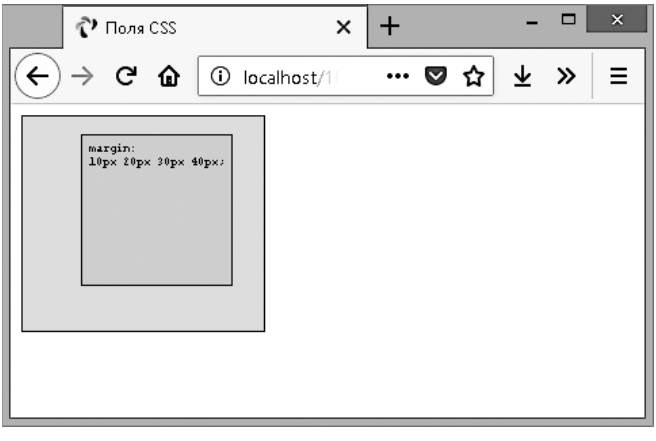

Рис. 18.13. Охватывающая таблица расширяется в соответствии с шириной полей

Применение границ

Уровень границ модели блока похож на уровень полей, за исключением того, что здесь отсутствует поглощение. Это следующий уровень по мере продвижения к центру модели блока. Основными свойствами, используемыми для изменения границ, являются `border`, `border-left`, `border-top`, `border-right` и `border-bottom`. Каждое из них может иметь другие подсвойства, добавляемые в виде суффиксов, например `-color`, `-style` и `-width`.

Четыре способа установки отдельных свойств, которые использовались для свойства `margin`, применимы и для свойства ширины границы — `border-width`, поэтому все следующие правила составлены верно:

```
/* Все границы */
border-width:1px;

/* Верхняя/нижняя и левая/правая */
```

```
border-width:1px 5px;

/* Верхняя, левая/правая и нижняя */
border-width:1px 5px 10px;

/* Верхняя, правая, нижняя и левая */
border-width:1px 5px 10px 15px;
```

На рис. 18.14 показано каждое из этих правил, примененное по очереди к группе квадратных элементов. Если смотреть на первый из них, сразу становится понятно, что ширина всех границ равна 1 пикселу. А вот у второго элемента верхняя и нижняя границы имеют ширину 1 пиксел, а его боковые границы имеют ширину по 5 пикселов. У третьего элемента верхняя граница шириной 1 пиксел, его боковые границы — по 5 пикселов, а его нижняя — 10 пикселов. Четвертый элемент изображен с верхней границей шириной 1 пиксел, правой границей — 5 пикселов, нижней — шириной 10 пикселов и левой — шириной 15 пикселов.

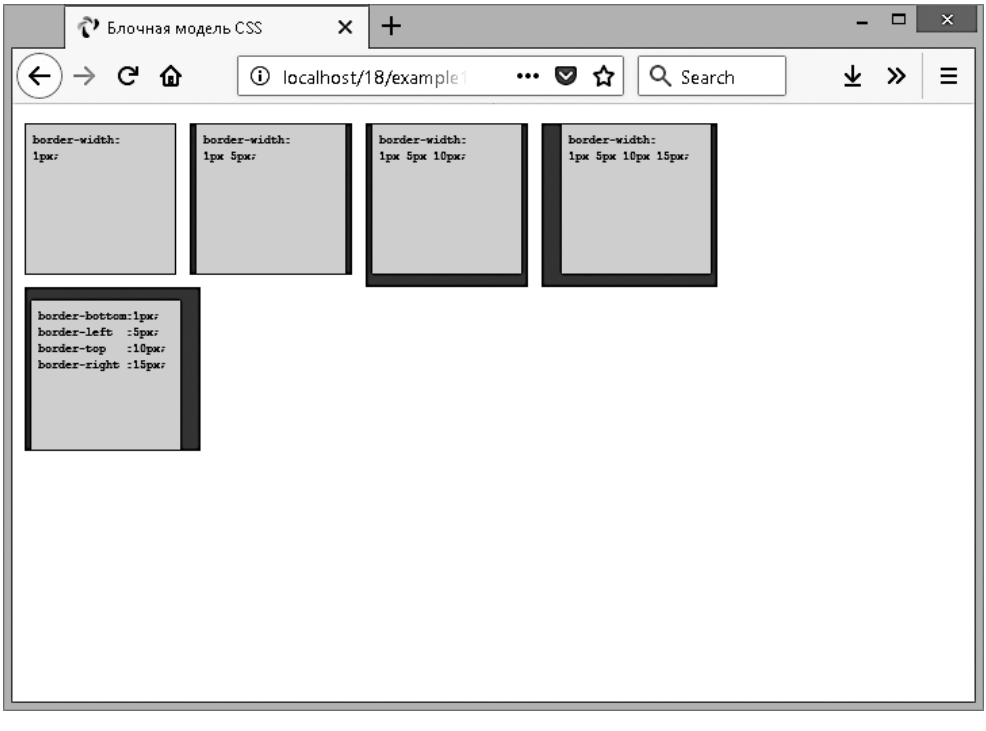

Рис. 18.14. Применение правил задания границ в полной и сокращенной записи

Для последнего элемента, расположенного под предыдущими элементами, сокращенная запись правил не использовалась. Вместо этого каждая из его границ задавалась отдельно. Как видите, для получения аналогичного результата потребовалось набрать значительно больше символов.

Настройка отступов

Самыми глубокими уровнями модели блока (отличающимися от содержимого элемента) являются отступы, применяемые внутри любых границ и (или) полей. Основными свойствами, используемыми для изменения отступов, являются `padding`, `padding-left`, `padding-top`, `padding-right` и `padding-bottom`.

Те четыре способа установки отдельных свойств, которые задействовались для свойств `margin` и `border`, применимы и для свойства отступа — `padding`, поэтому все следующие правила составлены верно:

```
/* Все отступы */
padding:1px;

/* Верхний/нижний и левый/правый */
padding:1px 2px;

/* Верхний, левый/правый и нижний */
padding:1px 2px 3px;

/* Верхний, правый, нижний и левый */
padding:1px 2px 3px 4px;
```

На рис. 18.15 показаны правила отступов, выделенные в примере 18.7 полужирным шрифтом и примененные к тексту в ячейке таблицы (как определено с помощью правила `display:table-cell;`, которое задает охват `<div>`-элемента наподобие ячейки таблицы). Размеры ячейки не заданы, поэтому она максимально плотно охватывает текст. Вследствие этого получается отступ, равный 10 пикселам над внутренним элементом, 20 пикселам справа, 30 пикселам снизу и 40 пикселам слева.

Пример 18.7. Применение отступов

```
<!DOCTYPE html>
<html>
  <head>
    <title>Отступы CSS</title>
    <style>
      #object1 {
        border-style:solid;
        border-width:1px;
        background  :orange;
        color       :darkred;
        font-family :Arial;
        font-size   :12px;
        text-align  :justify;
        display     :table-cell;
        width       :148px;
        padding     :10px 20px 30px 40px; }
    </style>
  </head>
  <body>
    <div id='object1'>To be, or not to be that is
```

```
      the question: Whether 'tis Nobler in the mind
      to suffer The Slings and Arrows of outrageous
      Fortune, Or to take Arms against a Sea of
      troubles, And by opposing end them.</div>
  </body>
</html>
```

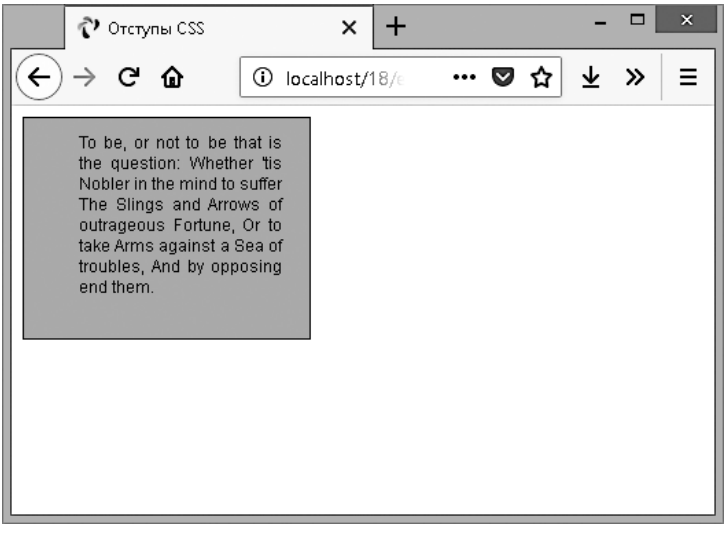

Рис. 18.15. Применение к объекту разных значений отступов

Содержимое объекта

И наконец, в глубине модели блока, в его центре, находится элемент, стиль которого может быть задан всеми способами, рассмотренными ранее в этой главе. Как вы теперь уже знаете, этот элемент может содержать (а зачастую и содержит) еще и подчиненные элементы, у которых, в свою очередь, могут быть свои подчиненные элементы и т. д., и у каждого из них могут быть свои настройки стиля и модели блока.

Вопросы

1. Какая инструкция используется для импорта одной таблицы стилей в другую (или в блок `<style>` кода HTML)?
2. Каким HTML-тегом можно воспользоваться для импорта таблицы стилей в документ?
3. Какой атрибут HTML-тега применяется для непосредственной вставки стиля в элемент?

4. В чем разница между идентификатором CSS и классом CSS?
5. Какие символы используются в качестве префиксов в CSS-правилах: а) идентификаторы и б) классы?
6. Каково назначение точки с запятой в CSS-правилах?
7. Как в таблице стилей добавляется комментарий?
8. Какой символ используется CSS для представления «любого элемента»?
9. Как в CSS можно выбрать группу разных элементов и (или) типов элементов?
10. Как можно задать преимущество одного из двух CSS-правил, имеющих одинаковый уровень приоритета?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

19 Расширение CSS с помощью CSS3

Первая реализация CSS была разработана в 1996 году, выпущена в 1999-м и к 2001 году была поддержана всеми выпусками браузеров. Стандарт для этой версии, CSS1, был еще раз пересмотрен в 2008 году. Со второй спецификацией, CSS2, разработчики начали работать в 1998 году, ее стандарт был в конечном итоге завершен в 2007-м, а затем еще раз пересмотрен в 2009 году.

В 2001 году началась разработка спецификации CSS3. Ее характеристики были предложены сравнительно недавно, в 2009 году, а самые последние рекомендации были включены еще в 2016 году. Рабочей группой CSS уже предлагается спецификация CSS4, хотя она не является существенным рывком вперед. Скорее это более глубокая проработка одной части CSS — селекторов, и рассмотрение ее подробностей выходит за рамки данной книги, тем более с учетом того, что первая черновая версия документации (https://drafts.csswg.org/selectors-4/) публиковалась на момент написания этих строк.

Но, к нашему удовольствию, рабочая группа CSS регулярно публикует краткие описания CSS-модулей, считающихся стабильными. До сих пор в виде заметок были опубликованы четыре таких документа с наилучшими актуальными примерами использования, причем самые последние публикации относятся к 2017 году (см. https://www.w3.org/TR/css-2017/). Это лучшее место, где можно оценить текущее состояние дел в мире CSS.

В этой главе будут описаны функциональные особенности CSS3, которые уже приняты всеми основными браузерами. Многие из них предоставляют такие функциональные возможности, которые до этого могли быть осуществлены лишь с помощью JavaScript.

Я рекомендую использовать CSS3 для реализации динамических свойств везде, где только можно, вместо JavaScript. Предоставляемые CSS свойства делают атрибуты документа частью самого документа, они уже не присоединяются к нему с помощью JavaScript. А когда они являются частью документа, замысел становится понятнее.

Следует заметить, что функций CSS запланировано очень много, и различные установки по-разному реализуются браузерами (если вообще реализуются). Поэтому при желании убедиться в работоспособности создаваемых вами установок CSS во всех браузерах я рекомендую сначала получить соответствующую справку на сайте http://caniuse.com. Там содержится свежая информация о том, какие функции доступны в браузерах.

Селекторы атрибутов

В предыдущей главе были подробно рассмотрены различные селекторы атрибутов, применяемые в CSS, которые мы сейчас вкратце повторим. Селекторы используются в CSS для сопоставления с HTML-элементами.

Существует десять разных типов селекторов, показанных в табл. 19.1.

Таблица 19.1. CSS-селекторы, псевдоклассы и псевдоэлементы

Тип селектора	Пример
Универсальный селектор	* { color:#555; }
Селекторы типов	b { color:red; }
Селекторы классов	.classname { color:blue; }
Селекторы идентификаторов	#id { background:cyan; }
Селекторы потомков	span em { color:green; }
Селекторы дочерних элементов	div > em { background:lime; }
Селекторы смежных элементов	i + b { color:gray; }
Селекторы атрибутов	a[href='info.htm'] { color:red; }
Псевдоклассы	a:hover { font-weight:bold; }
Псевдоэлементы	p::first-letter { font-size:300%; }

Разработчики CSS3 решили, что большинство из этих селекторов работают достаточно хорошо и в представленном на данный момент виде, но три усовершенствования, направленные на упрощение поиска соответствия элементам на основе содержимого их атрибутов, они все же внесли. Эти усовершенствования будут рассмотрены в следующих разделах.

Соответствие частям строк. В CSS2 для поиска соответствия строке `'info.htm'`, находящейся в `href`-атрибуте, можно было использовать такой селектор, как `a[href='info.htm']`, но поиска соответствия только части строки не существовало. В CSS3 пошли дальше и определили три новых оператора: `^`, `$` и `*`. Если один из них непосредственно предшествует символу равенства (`=`), то с помощью этих

символов в том порядке, в котором они перечислены, можно искать соответствие в начале, в конце или в любой части строки.

Оператор ^=

Этот оператор задает поиск соответствия в начале строки, например следующему селектору будет соответствовать любой `href`-атрибут, чье значение начинается со строки `http://website`:

```
a[href^='http://website']
```

Таким образом, ему будет соответствовать следующий элемент:

```
<a href='http://website.com'>
```

А этот элемент соответствовать не будет:

```
<a href='http://mywebsite.com'>
```

Оператор $=

Для поиска соответствия только в конце строки можно использовать следующий селектор, которому будет соответствовать любой `img`-тег, чей `src`-атрибут заканчивается на `.png`:

```
img[src$='.png']
```

Например, ему будет соответствовать такой тег:

```
<img src='photo.png' />
```

А этот тег соответствовать не будет:

```
<img src='snapshot.jpg' />
```

Оператор *=

Для поиска соответствия любой подстроке, находящейся где-либо в атрибуте, можно воспользоваться следующим селектором. Он найдет любые ссылки на странице, имеющие строку `google` в любом месте ссылки:

```
a[href*='google']
```

Например, ему будет соответствовать эта часть кода HTML:

```
<a href='http://google.com'>
```

а эта часть соответствовать не будет:

```
<a href='http://gmail.com'>
```

Свойство box-sizing

В модели блока W3C определено, что ширина и высота объекта должны относиться только к размерам содержимого элемента, игнорируя любые отступы или границы. Но некоторые веб-дизайнеры выразили желание указывать размеры, относящиеся ко всему элементу, включая любые отступы и границы.

Чтобы предоставить такое свойство, CSS3 позволяет вам выбрать желаемую модель блока со свойством задания размеров блока — `box-sizing`. Например, для использования общей ширины и высоты объекта, включая отступы и границы, нужно применять следующее объявление:

```
box-sizing:border-box;
```

Или, чтобы ширина и высота объекта относились только к содержимому, нужно воспользоваться таким объявлением (применяемым по умолчанию):

```
box-sizing:content-box;
```

Создание фона в CSS3

Спецификация CSS3 предоставляет два новых свойства: `background-clip` и `background-origin`, которые могут использоваться для указания, где фон должен начинаться внутри элемента и как усекать фон так, чтобы он не появлялся в тех частях модели блока, где это нежелательно.

Для выполнения названных задач оба свойства поддерживают следующие значения:

- `border-box` — относится к внешнему краю границы;
- `padding-box` — относится к внешнему краю области отступа;
- `content-box` — относится к внешнему краю области содержимого.

Свойство background-clip

Это свойство определяет, должен ли фон игнорироваться (усекаться), если он появляется либо внутри границы, либо в области отступов элемента. Например, следующее объявление определяет, что фон может отображаться во всех частях элемента вплоть до внешнего края границы:

```
background-clip:border-box;
```

Если не нужно, чтобы фон появлялся в области границы элемента, его можно ограничить только той частью элемента, которая находится внутри и заканчивается внешним краем его области отступов, например:

```
background-clip:padding-box;
```

Или же можно ограничить фон, чтобы он отображался только внутри области содержимого элемента, воспользовавшись следующим объявлением:

```
background-clip:content-box;
```

На рис. 19.1 показаны три ряда элементов, отображаемых в браузере Safari: в первом ряду для свойства `background-clip` используется значение `border-box`, во втором применяется значение `padding-box`, а в третьем используется значение `content-box`.

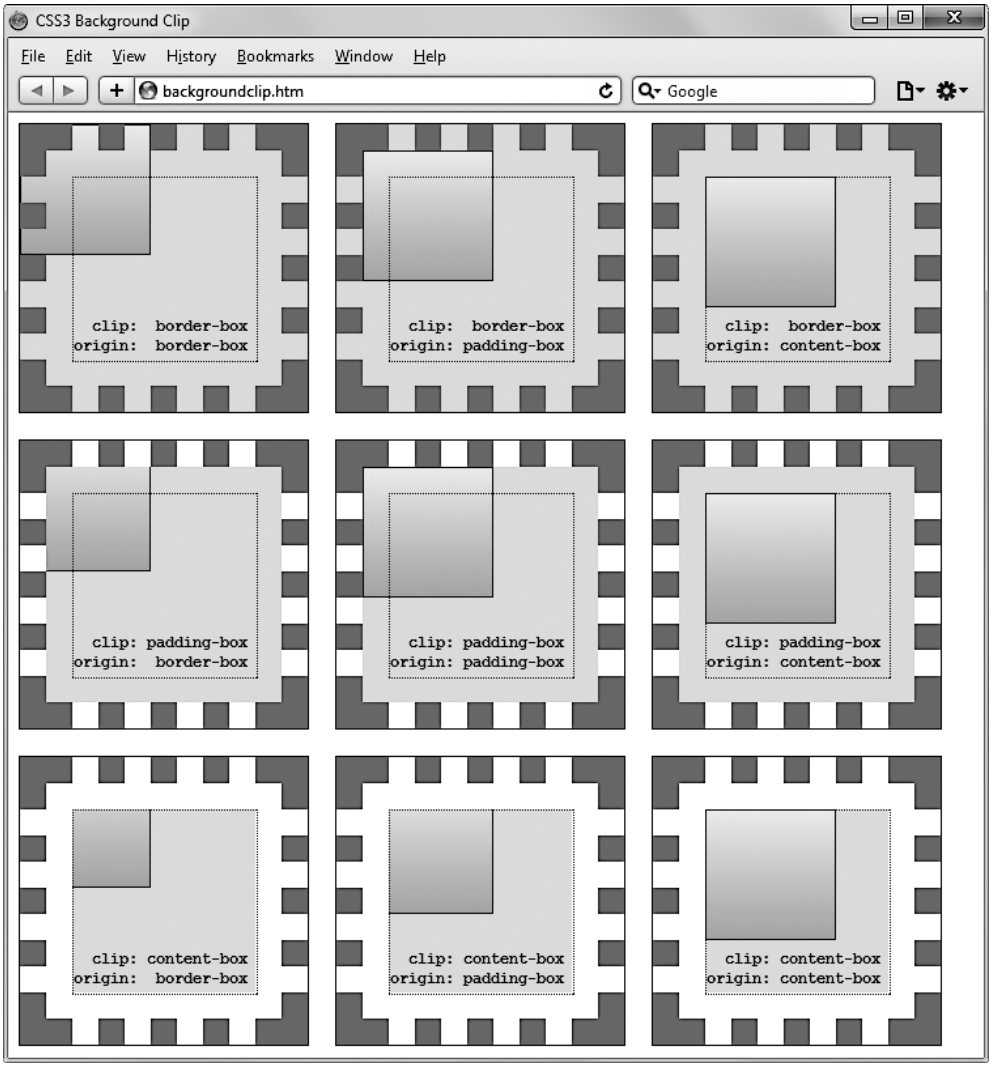

Рис. 19.1. Разные способы сочетания свойств фона CSS3

В первом ряду внутреннему блоку (файлу изображения, загруженному в левую верхнюю часть элемента с отключенным повторением) разрешается отображаться в элементе везде. Можно также совершенно отчетливо видеть, что он отображается в области границы первого блока, поскольку стиль границы указан пунктирным.

Во втором ряду в области границы не отображаются ни фоновое изображение, ни фоновое затенение, поскольку они были усечены по области отступов с помощью установки для свойства `background-clip` значения `padding-box`.

И наконец, в третьем ряду и фоновое затенение, и фоновое изображение были усечены для отображения только внутри области содержимого каждого элемента (показанного внутри светлого, ограниченного пунктирной линией блока) путем установки для свойства `background-clip` значения `content-box`.

Свойство background-origin

С помощью этого свойства можно также указать, где должно располагаться фоновое изображение, определив для этого, где должен начинаться левый верхний угол данного изображения. Например, следующее объявление указывает, что начало фонового изображения должно быть в левом верхнем углу внешнего края границы:

```
background-origin:border-box;
```

Чтобы установить начало изображения в левый верхний внешний угол области отступов, нужно воспользоваться таким объявлением:

```
background-origin:padding-box;
```

И чтобы установить начало изображения в левый верхний угол области внутреннего содержимого элемента, нужно воспользоваться следующим объявлением:

```
background-origin:content-box;
```

Посмотрите еще раз на рис. 19.1. В каждом ряду для первого блока используется свойство `background-origin` со значением `border-box`, для второго блока это же свойство применяется со значением `padding-box`, а для третьего — со значением `content-box`. Следовательно, в каждом ряду меньший по размеру внутренний блок отображается для первого блока в левом верхнем углу границы, для второго он отображается в левом верхнем углу области отступов, а для третьего — в левом верхнем углу содержимого.

Единственное отличие рядов, которое стоит отметить в отношении начала внутреннего блока на рис. 19.1, состоит в том, что в рядах 2 и 3 внутренний блок усекается соответственно областями отступов и содержимого, поэтому та часть блока, которая находится за пределами этих областей, не отображается.

Свойство background-size

Точно так же, как это делалось для указания ширины и высоты изображения при использовании тега `<img>`, в последних версиях всех браузеров можно сделать то же самое для изображений фона.

Свойство можно применить следующим образом (здесь *ww* — ширина, а *hh* — высота):

```
background-size:wwpx hhpx;
```

При необходимости можно использовать только один аргумент, и для обоих размеров будет установлено указанное значение. Кроме того, если применить данное свойство к блочному элементу, например к `<div>` (но не к такому встроенному элементу, как `<span>`), можно указать ширину и (или) высоту в процентном отношении, а не в виде фиксированного значения.

Использование значения auto. Если нужно масштабировать только один размер фонового изображения, чтобы при этом автоматически масштабировался и другой его размер для соблюдения прежних пропорций, для другого размера можно воспользоваться значением `auto`:

```
background-size:100px auto;
```

Этим объявлением устанавливается ширина, равная 100 пикселам, и высота, равная значению, пропорциональному увеличению или уменьшению ширины.

Разные браузеры могут требовать различных версий имен свойства background, поэтому при их использовании обратитесь к сайту http://caniuse.com, чтобы убедиться, что вы применяете все версии, требуемые тем браузерам, на работу с которыми вы рассчитываете.

Использование нескольких фонов

Используя CSS3, вы можете прикрепить к элементу несколько фонов, каждый из которых может применять ранее рассмотренные свойства фона CSS3. Соответствующий пример показан на рис. 19.2. На этом рисунке в качестве фона были назначены восемь изображений, которые используются для создания четырех углов и четырех кромок границы сертификата.

Для вывода нескольких фоновых изображений с помощью одного CSS-объявления нужно разделить их запятыми. В примере 19.1 показан код HTML и CSS, использованный для создания фона рис. 19.2.

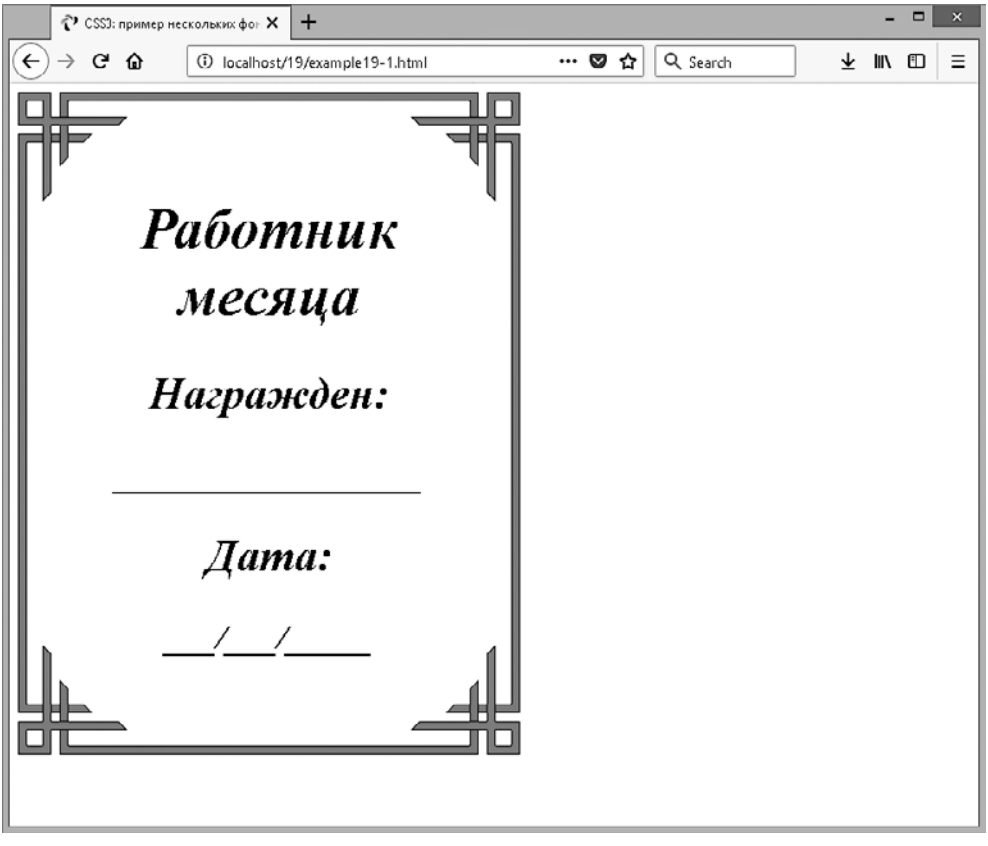

Рис. 19.2. Фон, созданный с помощью нескольких изображений

Пример 19.1. Использование в фоне сразу нескольких изображений

```
<!DOCTYPE html>
<html> <!-- backgroundimages.html -->
  <head>
    <title>CSS3: пример нескольких фоновых изображений</title>
    <style>
      .border {
        font-family :'Times New Roman';
        font-style  :italic;
        font-size   :170%;
        text-align  :center;
        padding     :60px;
        width       :350px;
        height      :500px;
        background  :url('b1.gif') top    left  no-repeat,
                     url('b2.gif') top    right no-repeat,
                     url('b3.gif') bottom left  no-repeat,
                     url('b4.gif') bottom right no-repeat,
```

```
                    url('ba.gif') top          repeat-x,
                    url('bb.gif') left         repeat-y,
                    url('bc.gif') right        repeat-y,
                    url('bd.gif') bottom       repeat-x
      }
    </style>
  </head>
  <body>
    <div class='border'>
      <h1>Работник месяца</h1>
      <h2>Награжден:</h2>
      <h3>____________________</h3>
      <h2>Дата:</h2>
      <h3>___/___/_____</h3>
    </div>
  </body>
</html>
```

Первые четыре строки объявления фона в блоке CSS расставляют угловые изображения по четырем углам элемента, а последние четыре строки помещают изображения кромок, которые обрабатываются в последнюю очередь, потому что порядок приоритетности для фоновых изображений имеет направление сверху вниз. Иными словами, когда они накладываются друг на друга, дополнительные фоновые изображения будут появляться позади уже размещенных изображений. Если бы GIF-изображения были перечислены в обратном порядке, то повторяющиеся изображения кромок отображались бы поверх углов, что было бы неправильно.

Используя этот код CSS, можно изменять размеры содержащего фон элемента по любым направлениям, и граница будет всегда правильно изменяться в размерах, чтобы поместиться в элементе, что намного проще, чем применять таблицы или несколько элементов для получения такого же эффекта.

Границы CSS3

CSS3 также придает намного больше гибкости способам возможного представления границ, разрешая независимо менять цвета всех четырех кромок, отображать изображения для кромок и углов, предоставлять значения радиусов для придания границам закругленных углов и помещать прямоугольные тени под элементами.

Свойство border-color

Применять цвета к границе можно двумя способами. Начнем с того, что свойству можно передать всего один цвет:

```
border-color:#888;
```

Это объявление устанавливает светло-серый цвет для всех границ элемента. Можно также установить цвета границ по отдельности (здесь цвета границы устанавливаются в различные градации серого):

```
border-top-color   :#000;
border-left-color  :#444;
border-right-color :#888;
border-bottom-color:#ccc;
```

Кроме того, можно назначить все цвета по отдельности в одном объявлении:

```
border-color:#f00 #0f0 #880 #00f;
```

Это объявление устанавливает цвет верхней границы в `#f00`, правой границы — в `#0f0`, нижней границы — в `#880` и левой границы — в `#00f` (в красный, зеленый, оранжевый и синий соответственно). Можно также использовать в качестве аргументов названия цветов.

Свойство border-radius

До появления CSS3 способные веб-разработчики придумали множество различных настроек с целью получения закругленных границ, используя, как правило, теги `<table>` или `<div>`.

Но теперь добавление закругленных границ к элементу дается по-настоящему легко и, как показано на рис. 19.3, работает в последних версиях всех основных браузеров. На этом рисунке граница толщиной 10 пикселов выведена различными способами. Код HTML для получения такого результата показан в примере 19.2.

Пример 19.2. Свойство border-radius

```
<!DOCTYPE html>
<html> <!-- borderradius.html -->
  <head>
    <title>CSS3: примеры радиусов границ</title>
    <style>
      .box {
        margin-bottom :10px;
        font-family   :'Courier New', monospace;
        font-size     :12pt;
        text-align    :center;
        padding       :10px;
        width         :380px;
        height        :75px;
        border        :10px solid #006; }
      .b1 {
        -moz-border-radius    :40px;
        -webkit-border-radius :40px;
        border-radius         :40px; }
      .b2 {
        -moz-border-radius    :40px 40px 20px 20px;
```

```
        -webkit-border-radius :40px 40px 20px 20px;
        border-radius         :40px 40px 20px 20px; }
      .b3 {
        -moz-border-radius-topleft         :20px;
        -moz-border-radius-topright        :40px;
        -moz-border-radius-bottomleft      :60px;
        -moz-border-radius-bottomright     :80px;
        -webkit-border-top-left-radius     :20px;
        -webkit-border-top-right-radius    :40px;
        -webkit-border-bottom-left-radius  :60px;
        -webkit-border-bottom-right-radius :80px;
        border-top-left-radius             :20px;
        border-top-right-radius            :40px;
        border-bottom-left-radius          :60px;
        border-bottom-right-radius         :80px; }
      .b4 {
        -moz-border-radius-topleft         :40px 20px;
        -moz-border-radius-topright        :40px 20px;
        -moz-border-radius-bottomleft      :20px 40px;
        -moz-border-radius-bottomright     :20px 40px;
        -webkit-border-top-left-radius     :40px 20px;
        -webkit-border-top-right-radius    :40px 20px;
        -webkit-border-bottom-left-radius  :20px 40px;
        -webkit-border-bottom-right-radius :20px 40px;
        border-top-left-radius             :40px 20px;
        border-top-right-radius            :40px 20px;
        border-bottom-left-radius          :20px 40px;
        border-bottom-right-radius         :20px 40px;
      }
    </style>
  </head>
  <body>
    <div class='box b1'>
      border-radius:40px;
    </div>

    <div class='box b2'>
       border-radius:40px 40px 20px 20px;
    </div>

    <div class='box b3'>
      border-top-left-radius    :20px;<br/>
      border-top-right-radius   :40px;<br>
      border-bottom-left-radius :60px;<br>
      border-bottom-right-radius:80px;
    </div>

    <div class='box b4'>
      border-top-left-radius    :40px 20px;<br>
      border-top-right-radius   :40px 20px;<br>
      border-bottom-left-radius :20px 40px;<br>
      border-bottom-right-radius:20px 40px;
    </div>
  </body>
</html>
```

Рис. 19.3. Смешивания и сопоставления различных свойств радиусов границы

Так, например, для создания закругленной границы с радиусом 20 пикселов можно просто воспользоваться следующим объявлением:

```
border-radius:20px;
```

Хотя многие браузеры (включая Internet Explorer) со свойствами радиусов границы будут работать хорошо, некоторые текущие (и многие более старые) версии основных браузеров используют разные имена свойств. Если нужна поддержка всех этих браузеров, придется также включить для них соответствующие префиксы, характерные для тех или иных браузеров, такие как -moz- и -webkit-. Чтобы обеспечить работу примера 19.2 во всех браузерах, я добавил в код все требуемые префиксы.

Можно указать отдельные радиусы для каждого из четырех углов (по часовой стрелке, начиная с левого верхнего угла):

```
border-radius:10px 20px 30px 40px;
```

При необходимости можете также указать радиус отдельно для каждого угла элемента:

```
border-top-left-radius     :20px;
border-top-right-radius    :40px;
border-bottom-left-radius  :60px;
border-bottom-right-radius :80px;
```

И при ссылке на отдельные углы можно предоставить два аргумента, выбирая тем самым разные вертикальные и горизонтальные радиусы (в результате чего получаются более интересные и тонко настраиваемые границы):

```
border-top-left-radius     :40px 20px;
border-top-right-radius    :40px 20px;
border-bottom-left-radius  :20px 40px;
border-bottom-right-radius :20px 40px;
```

Первым аргументом задается горизонтальный, а вторым — вертикальный радиус.

Прямоугольные тени

Для применения прямоугольной тени нужно указать горизонтальное и вертикальное смещение от объекта и величину размытости, добавляемой к тени, а также используемый для тени цвет:

```
box-shadow:15px 15px 10px #888;
```

Два значения по `15px` задают (по порядку) горизонтальное вертикальное смещение от элемента, и эти значения могут быть отрицательными, нулевыми или положительными. Значение `10px` указывает величину, где меньшие значения приводят к меньшей размытости, а `#888` — это цвет тени, который может быть любым допустимым цветом. Результат этого объявления можно увидеть на рис. 19.4.

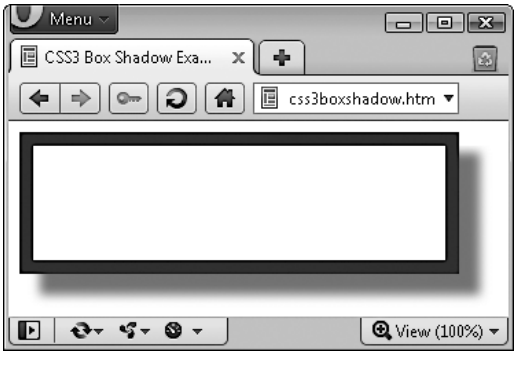

Рис. 19.4. Прямоугольная тень, отображенная под элементом element

При использовании этого свойства в браузерах, основанных на движках WebKit и Mozilla, нужно применять префиксы -webkit- и -moz-.

Выход элемента за пределы размеров

В CSS2 можно определить, что делать, когда один элемент слишком велик, чтобы полностью поместиться в другом, родительском по отношению к нему элементе, путем указания для свойства `overflow` значения `hidden`, `visible`, `scroll` или `auto`. Но теперь в CSS3 можно также отдельно применить эти значения к горизонтальному или вертикальному направлению, как в следующих примерах объявлений:

```
overflow-x:hidden;
overflow-x:visible;
overflow-y:auto;
overflow-y:scroll;
```

Разметка с использованием нескольких колонок

Использование нескольких колонок уже давно стало у веб-разработчиков наиболее востребованным свойством, и в CSS3 оно наконец-то было реализовано, а Internet Explorer 10 стал последним из основных браузеров, принявшим это свойство.

Теперь перетекание текста по нескольким колонкам задать не сложнее, чем указать количество колонок, а затем (дополнительно) выбрать разрядку между ними и тип разделительной линии (если она нужна). На рис. 19.5 показан результат выполнения кода примера 19.3.

Пример 19.3. Использование CSS для создания нескольких колонок

```
<!DOCTYPE html>
<html> <!-- multiplecolumns.html -->
  <head>
    <title>Использование колонок</title>
    <style>
      .columns {
        text-align            :justify;
        font-size             :16pt;
        -moz-column-count     :3;
        -moz-column-gap       :1em;
        -moz-column-rule      :1px solid black;
        -webkit-column-count  :3;
        -webkit-column-gap    :1em;
        -webkit-column-rule   :1px solid black;
        column-count          :3;
        column-gap            :1em;
        column-rule           :1px solid black;
      }
    </style>
  </head>
  <body>
    <div class='columns'>
      Now is the winter of our discontent
      Made glorious summer by this sun of York;
```

```
      And all the clouds that lour'd upon our house
      In the deep bosom of the ocean buried.
      Now are our brows bound with victorious wreaths;
      Our bruised arms hung up for monuments;
      Our stern alarums changed to merry meetings,
      Our dreadful marches to delightful measures.
      Grim-visaged war hath smooth'd his wrinkled front;
      And now, instead of mounting barded steeds
      To fright the souls of fearful adversaries,
      He capers nimbly in a lady's chamber
      To the lascivious pleasing of a lute.
    </div>
  </body>
</html>
```

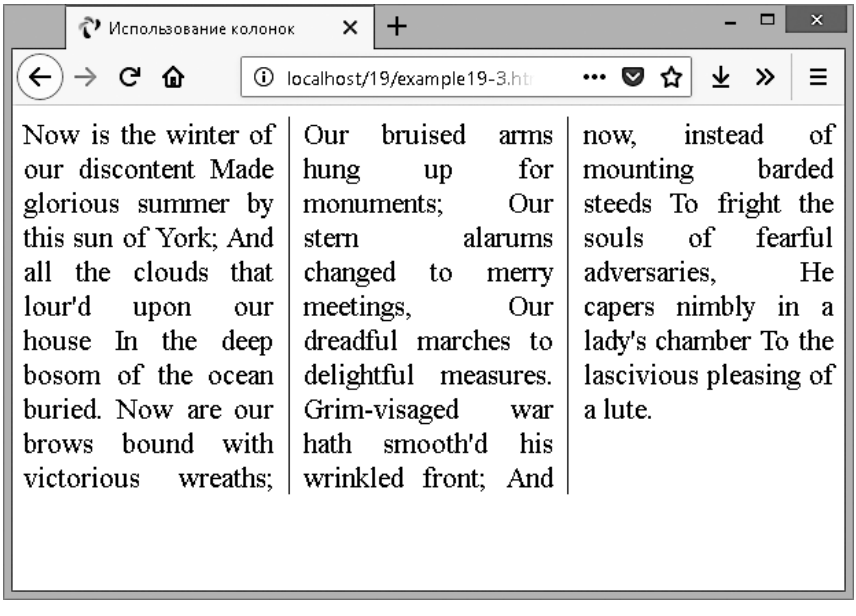

Рис. 19.5. Перетекание текста по нескольким колонкам

Внутри класса `.columns` первые две строки просто предписывают браузеру выровнять текст по правому краю и установить для него размер шрифта `16pt`. Эти объявления для нескольких колонок не нужны, но они улучшают отображение текста. В остальных строках элемент настраивается таким образом, чтобы внутри него текст перетекал по трем колонкам с разрывом между колонками, равным `1em`, и с границей 1 пиксел, проходящей посередине каждого разрыва.

В примере 19.3 браузеры на основе движков Mozilla и WebKit могут потребовать для объявлений соответствующих этим браузерам префиксов.

Цвета и непрозрачность

Способы определения цветов в CSS3 существенно расширились: теперь вы можете также использовать CSS-функции для применения цветов в широко распространенных форматах RGB (красный, зеленый, синий), RGBA (красный, зеленый, синий, альфа), HSL (тон, насыщенность, яркость) и HSLA (тон, насыщенность, яркость, альфа). Значение *альфа* определяет прозрачность цвета, позволяющую увидеть элементы, расположенные ниже.

Цвета HSL

Для определения цвета с помощью функции `hsl` сначала нужно выбрать из цветового круга значение для тона в диапазоне от 0 до 359. Любой более высокий номер цвета просто возвращается по кругу к началу, таким образом значение `0` соответствует красному цвету точно так же, как и значения `360` и `720`.

В цветовом круге основные цвета — красный, зеленый и синий — занимают по 120 градусов, поэтому чистый красный цвет соответствует значению `0`, зеленый — значению `120`, а синий — значению `240`. Числа между этими значениями представляют собой оттенки, содержащие различные пропорции основных цветов с обеих сторон.

Затем нужен уровень насыщенности, значение которого лежит в диапазоне от 0 до 100 %. Он определяет то, насколько сильно цвет будет размыт или ярок. Значения насыщенности начинаются в центре колеса со светло-серого цвета (насыщенность равна 0 %), а затем по направлению к краю (где насыщенность равна 100 %) она становится все более отчетливой.

Вам остается только решить, насколько ярким требуется цвет, для чего нужно выбрать значение яркости в диапазоне от 0 до 100 %. Значение `50%` для яркости дает наполненный, яркий цвет, а уменьшение значения (вниз, вплоть до минимума `0%`) делает его темнее до тех пор, пока цвет не станет черным. Увеличение значения (вверх, вплоть до максимума `100%`) делает цвет светлее до тех пор, пока он не станет белым. Вы можете визуально представить это подмешиванием в цвет либо черного, либо белого цвета.

Так, например, для выбора полностью насыщенного желтого цвета со стандартной яркостью нужно воспользоваться следующим объявлением:

```
color:hsl(60, 100%, 50%);
```

Или для выбора темно-синего цвета можно воспользоваться таким объявлением:

```
color:hsl(240, 100%, 40%);
```

Этим также можно воспользоваться (как и всеми остальными CSS-функциями, связанными с заданием цвета) с любым свойством, ожидающим применения цветовых настроек, например с `background-color` и т. д.

Цвета HSLA

Для обеспечения еще большего контроля над способом цветоообразования можно воспользоваться функцией `hsla`, предоставив ей четвертый (альфа) уровень настройки цвета, значение которого задается числом с плавающей точкой в диапазоне от 0 до 1. Значение `0` определяет, что цвет полностью прозрачный, а число `1` задает полную непрозрачность цвета.

Выбрать желтый цвет с полной насыщенностью, стандартной яркостью и 30%-ной непрозрачностью можно с помощью следующего объявления:

```
color:hsla(60, 100%, 50%, 0.3);
```

Или же для выбора полностью насыщенного, но чуть более светлого синего цвета с 82%-ной непрозрачностью можно воспользоваться таким объявлением:

```
color:hsla(240, 100%, 60%, 0.82);
```

Цвета RGB

Наверное, вам более знакома система выбора цвета RGB, поскольку она похожа на использование форматов цвета `#nnnnnn` и `#nnn`. Например, для задания желтого цвета можно воспользоваться любым из следующих объявлений (первое из них поддерживает 16 миллионов цветов, а второе — 4 тысячи):

```
color :#ffff00;
color :#ff 0;
```

Для получения такого же результата можно также воспользоваться CSS-функцией `rgb`, но при этом нужно применять не шестнадцатеричные, а десятичные числа (где десятичное число 255 соответствует шестнадцатеричному числу `ff`):

```
color:rgb(255, 255, 0);
```

Но еще лучше вам будет даже не задумываться больше о том, чему соответствуют значения до 256, поскольку можно указать процентные значения:

```
color:rgb(100%, 100%, 0);
```

Фактически теперь вы можете с большой точностью определить настройки для нужного цвета, просто думая о его основных цветовых составляющих. Например, сочетание зеленого и синего дает фиолетовый цвет, поэтому для задания цвета, близкого к фиолетовому, но с синей составляющей, преобладающей над зеленой, можно составить первое предположение, что для него нужно определить 0 % красного, 40 % зеленого и 60 % синего цвета и попробовать воспользоваться следующим объявлением:

```
color:rgb(0%, 60%, 40%);
```

Цвета RGBA

Как и функция `hsla`, функция `rgba` поддерживает четвертый (альфа) аргумент, позволяющий, к примеру, с помощью следующего объявления применить к прежнему фиолетовому цвету 40%-ную непрозрачность:

```
color:rgba(0%, 60%, 40%, 0.4);
```

Свойство opacity

Свойство `opacity` предоставляет такое же альфа-управление, что и функции `hsla` и `rgba`, но позволяет изменять непрозрачность объекта (или прозрачность, если это вам больше нравится) отдельно от его цвета.

Для использования этого цвета нужно применить к элементу следующее объявление (которое в данном примере устанавливает непрозрачность, равную 25 %, или прозрачность, равную 75 %):

```
opacity:0.25;
```

Для браузеров на основе движков WebKit и Mozilla для этого свойства требуются соответствующие этим браузерам префиксы. Кроме того, для обратной совместимости с выпусками Internet Explorer, предшествующими версии 9, нужно добавить такое объявление (в котором значение непрозрачности умножено на 100):

```
filter:alpha(opacity='25');
```

Эффекты, применяемые к тексту

Теперь с помощью CSS3 к тексту могут применяться новые эффекты, включая тени и перенос слов.

Свойство text-shadow

Это свойство аналогично свойству `box-shadow` и получает такой же набор аргументов: горизонтальное и вертикальное смещение, величину размытости и используемый цвет. Например, следующее объявление задает смещение тени на 3 пиксела как в горизонтальном, так и в вертикальном направлениях и отображает тень темно-серым цветом с размытостью 4 пиксела:

```
text-shadow:3px 3px 4px #444;
```

Результат применения этого объявления выглядит, как показано на рис. 19.6. Это объявление работает в последних версиях всех основных браузеров (кроме Internet Explorer 9 или ниже).

Рис. 19.6. Применение тени к тексту

Свойство text-overflow

При использовании любого из CSS-свойств `overflow` со значением, равным `hidden`, можно также воспользоваться свойством `text-overflow` для помещения многоточия сразу же после текста, подвергшегося усечению:

```
text-overflow:ellipsis;
```

Если это свойство не использовать, то когда текст «To be, or not to be. That is the question.» усекается, результат выглядит так, как показано на рис. 19.7. С применением объявления результат выглядит так, как изображено на рис. 19.8.

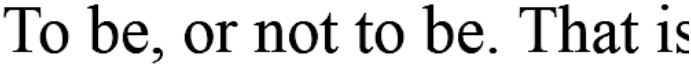

Рис. 19.7. Текст автоматически усекается

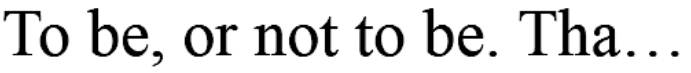

Рис. 19.8. Вместо простого усечения текст завершается многоточием

Чтобы это работало, требуется выполнить три условия.

- У элемента должно быть свойство `overflow`, настроенное на невидимость, например `overflow:hidden`.
- Элемент должен иметь свойство `white-space:nowrap`, настраивающее на ограничение текста.
- Ширина элемента должна быть меньше, чем усекаемый текст.

Свойство word-wrap

Когда используется по-настоящему длинное слово, шире того элемента, в котором оно содержится, оно либо выйдет за пределы, либо будет усечено. Но в качестве альтернативного варианта свойству `text-overflow` и усечению текста можно воспользоваться свойством `word-wrap` со значением `break-word` для переноса длинных строк:

```
word-wrap:break-word;
```

Например, на рис. 19.9 показано, что слово Honorificabilitudinitatibus шире, чем содержащее его поле (чей правый край показан в виде сплошной вертикальной черты между буквами t и a), и, поскольку свойство `overflow` применено не было, слово выходит за границу своего контейнера.

Рис. 19.9. Слово имеет слишком большую ширину для своего контейнера, поэтому выходит за его границу

Но на рис. 19.10 свойству `word-wrap` элемента было присвоено значение `break-word`, поэтому слово аккуратно перенесено на следующую строку.

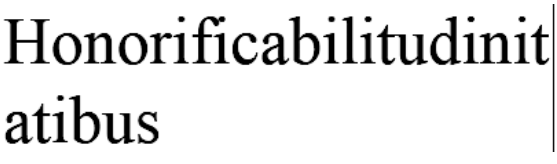

Рис. 19.10. Теперь слово переносится по достижении правого края

Веб-шрифты

Применение веб-шрифтов CSS3 существенно повысило возможности оформления текста, доступные веб-дизайнерам, позволяя загружать шрифты из Интернета, а не только со своего пользовательского компьютера, и отображать их по всей Всемирной сети. Для достижения такого результата нужно объявить веб-шрифт с помощью свойства `@font-face`:

```
@font-face
{
  font-family:FontName; src:url('FontName.otf');
}
```

Функция `url` требует значение, содержащее путь либо URL-адрес шрифта. В большинстве браузеров можно использовать или шрифты TrueType (`.ttf`), или шрифты OpenType (`.otf`), но Internet Explorer ограничивает вас применением шрифтов TrueType, преобразованных в шрифты Embedded Open Type (`.eot`).

Чтобы сообщить браузеру тип шрифта, можно воспользоваться функцией `format`, как в следующем примере для шрифтов OpenType:

```
@font-face
{
  font-family:FontName;
```

```
  src:url('FontName.otf') format('opentype');
}
```

или в этом примере (для шрифтов TrueType):

```
@font-face
{
  font-family:FontName;
  src:url('FontName.ttf') format('truetype');
}
```

Но поскольку Internet Explorer принимает только EOT-шрифты, он игнорирует объявления `@font-face`, содержащие функцию `format`.

Веб-шрифты Google. Один из лучших способов использования веб-шрифтов — их бесплатная загрузка с серверов Google. Дополнительную информацию по данному вопросу можно найти на сайте веб-шрифтов Google (http://google.com/webfonts) (рис. 19.11), где можно получить доступ более чем к 848 семействам шрифтов!

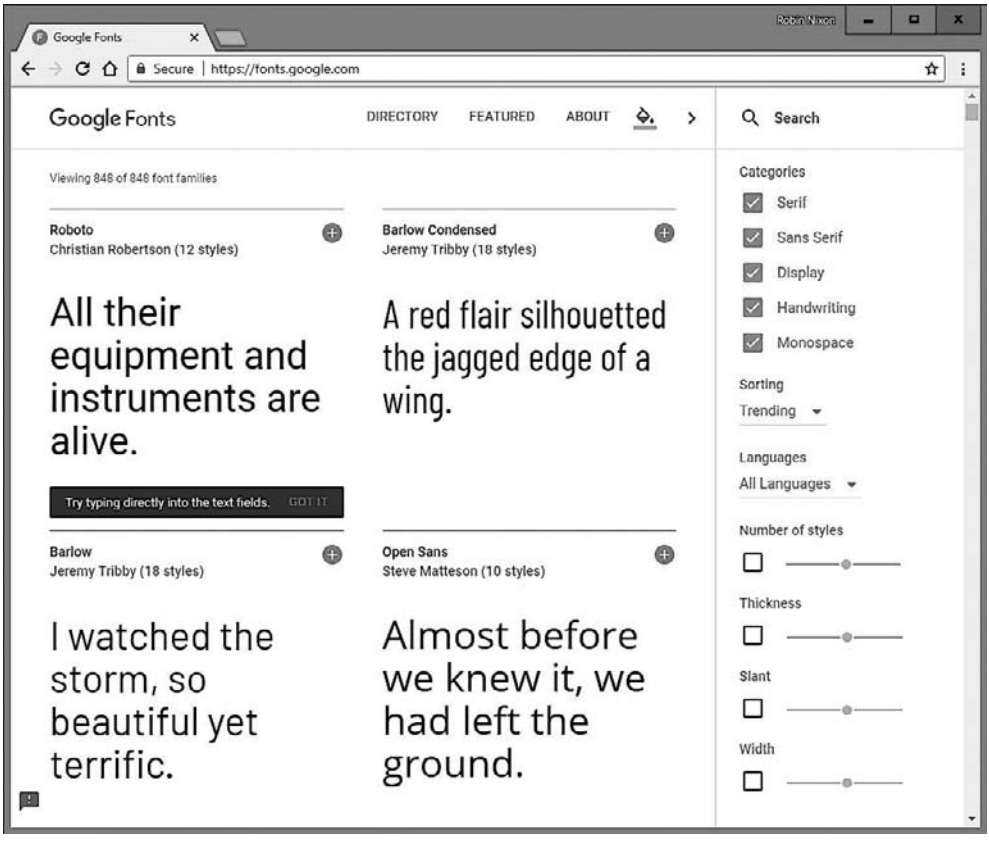

Рис. 19.11. Включить веб-шрифты Google не составляет труда

Чтобы вы увидели, насколько легко можно использовать один из этих шрифтов, в следующем примере показано, как загрузить шрифт Google (в данном случае Lobster) в ваш HTML-код для использования в заголовках `<h1>`:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      hi { font-family:'Lobster', arial, serif; }
    </style>
    <link href='http://fonts.googleapis.com/css?family=Lobster'
      rel='stylesheet'
  </head>
  <body>
    <h1>Hello</h1>
  </body>
</html>
```

При выборе шрифта с сайта Google предоставляет тег `<link>` для копирования и вставки его в раздел `<head>` вашей веб-страницы.

Трансформации

Используя трансформации, можно наклонять, вращать, растягивать и сжимать элементы в любом из трех измерений (да, 3D поддерживается, но пока только в браузерах, работающих на движке WebKit). Это упрощает создание впечатляющих эффектов путем выхода за пределы однообразных макетов на основе `<div>`-контейнеров и других элементов, поскольку теперь они могут быть показаны под различными углами и в различных формах.

Для выполнения трансформаций нужно воспользоваться свойством `transform` (у которого, к сожалению, должны быть соответствующие префиксы для использования в браузерах Mozilla, WebKit, Opera и Microsoft; по этому вопросу снова следует обратиться на сайт http://caniuse.com).

К свойству `transform` можно применять множество значений, начиная со значения `none`, которое переключает объект в состояние, не допускающее трансформаций:

```
transform:none;
```

Свойство `transform` можно дополнить одной или несколькими из следующих разнообразных функций:

- `matrix` — трансформирует объект, применяя к нему матрицу значений;
- `translate` — перемещает исходную точку элемента;
- `scale` — масштабирует объект;

- `rotate` — вращает объект;
- `skew` — наклоняет объект.

Единственное, с чем может быть трудно разобраться, — это наклон (`skew`). При применении этой функции одна из координат смещается в одном из направлений пропорционально своему расстоянию от координатной плоскости или оси. Таким образом при наклоне прямоугольник, к примеру, преобразуется в параллелограмм.

Существуют также отдельные версии многих из этих функций, например `translateX`, `scaleY` и т. д.

Так, например, чтобы повернуть элемент по часовой стрелке на 45°, можно применить к нему такое объявление:

```
transform:rotate(45deg);
```

В то же время вы можете увеличить объект, как это делается с помощью следующего объявления, приводящего к увеличению его ширины в полтора, а высоты в два раза с дальнейшим поворотом:

```
transform:scale(1.5, 2) rotate(45deg);
```

На рис. 19.12 показан объект до и после применения трансформации.

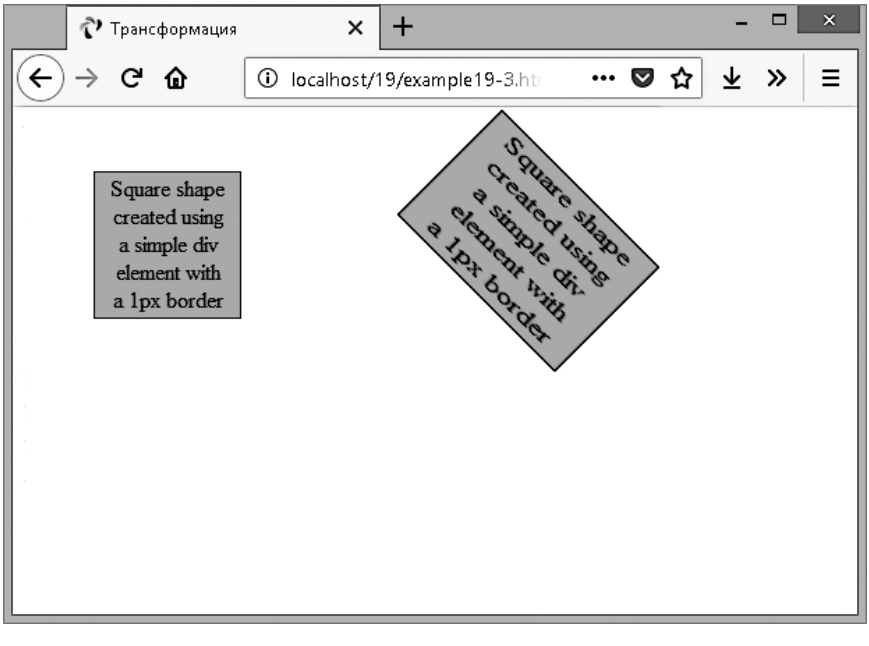

Рис. 19.12. Объект до и после трансформации

Трехмерная трансформация. Объекты на странице можно также трансформировать в трех измерениях, используя следующие свойства трехмерной трансформации CSS3:

- `perspective` — освобождение элемента из двумерного пространства и создание третьего измерения, в котором он может перемещаться; требуется для работы с 3D-функциями CSS;
- `transform-origin` — установка с использованием перспективы того места, где все линии сходятся в одну точку;
- `translate3d` — перемещение элемента в другое место трехмерного пространства;
- `scale3d` — изменение масштаба одного или нескольких измерений;
- `rotate3d` — вращение элемента вокруг любой из осей *X*, *Y* и *Z*.

На рис. 19.13 показан двумерный объект, подвергшийся вращению в трехмерном пространстве с помощью следующего CSS-правила:

```
transform:perspective(200px) rotateX(10deg) rotateY(20deg) rotateZ(30deg);
```

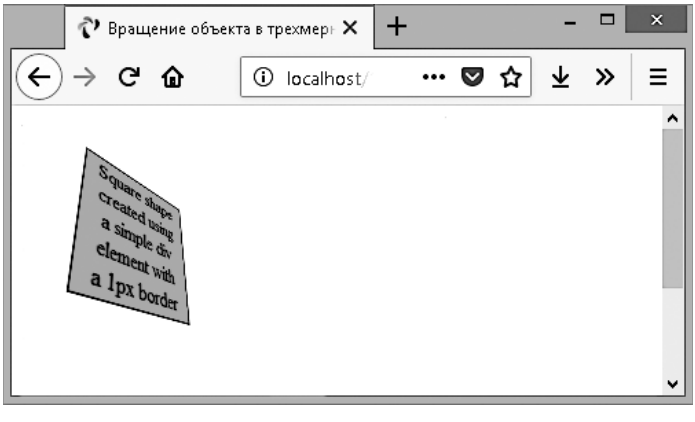

Рис. 19.13. Вращение объекта в трехмерном пространстве

За дополнительной информацией обратитесь к руководству по адресу http://tinyurl.com/3dcsstransforms.

Переходы

В последних версиях основных браузеров (включая Internet Explorer 10, не ниже) появилось новое динамичное свойство, называемое *переходами*. Переходы определяют эффект анимации, который нужно применить при трансформации элемента, и браузер автоматически позаботится за вас обо всех промежуточных кадрах.

Для настройки перехода можно предоставить четыре свойства:

```
transition-property       :свойство;
transition-duration       :время;
transition-delay          :время;
transition-timing-function:тип;
```

Свойства нужно предварять соответствующими префиксами браузеров, работающих на движках Mozilla, WebKit, Opera и Microsoft.

Свойства, применяемые к переходам

У переходов есть такие свойства, как `height` и `border-color`. При указании свойств преследуется цель изменения CSS-свойства по имени `transition-property` (здесь слово property («свойства») используется двумя разными способами: для CSS-свойства и для устанавливаемых им свойств переходов). Можно включить в объявление сразу несколько свойств, разделяя их запятыми:

```
transition-property:width, height, opacity;
```

Или, если вам нужно абсолютно все, относящееся к элементу, подвергаемому переходу (включая цвета), используется значение `all`:

```
transition-property:all;
```

Продолжительность перехода

Свойство `transition-duration` требует значения от нуля и более секунд. Следующее объявление задает завершение перехода через 1,25 с:

```
transition-duration :1.25s;
```

Задержка перехода

Если свойству `transition-delay` дается значение более нуля секунд (то есть более значения по умолчанию), происходит задержка между исходным отображением элемента и началом его перехода. Следующее объявление задает начало перехода после задержки 0,1 с:

```
transition-delay:0.1s;
```

Если свойству `transition-delay` дается значение меньше нуля секунд (иными словами, отрицательное значение), переход будет выполнен в момент изменения свойства, но проявится таким образом, будто оно началось с указанным смещением по времени, то есть на каком-то своем промежуточном цикле.

Задание скорости перехода

Свойству `transition-timing-function` требуется присвоить одно из следующих значений:

- `ease` — медленное начало, ускорение и медленное завершение;
- `linear` — переход с постоянной скоростью;
- `ease-in` — медленное начало, а затем быстрый переход до самого завершения;
- `ease-out` — быстрое начало, сохранение высокой скорости почти до завершения и медленное завершение;
- `ease-in-out` — медленное начало, быстрый переход, затем медленное завершение.

Использование любого из этих значений со словом `ease` обеспечивает исключительную плавность и естественность перехода, в отличие от линейного (`linear`) перехода, который выглядит более механическим. И если этих изменений вам недостаточно, вы можете также создать свой собственный переход, используя функцию `cubic-bezier`.

Например, следующие объявления применялись для создания пяти предыдущих типов переходов и показывают, как просто можно создавать свои собственные переходы:

```
transition-timing-function:cubic-bezier(0.25, 0.1, 0.25, 1);
transition-timing-function:cubic-bezier(0,    0,   1,    1);
transition-timing-function:cubic-bezier(0.42, 0,   1,    1);
transition-timing-function:cubic-bezier(0,    0,   0.58, 1);
transition-timing-function:cubic-bezier(0.42, 0,   0.58, 1);
```

Сокращенный синтаксис

Возможно, проще будет воспользоваться сокращенной версией этого свойства и включить все значения в одно объявление (такое, как показано далее), которым задается переход всех свойств в линейном режиме за период 0,3 с, после начальной (необязательной) задержки 0,2 с:

```
transition:all .3s linear .2s;
```

Это избавит вас от хлопот, связанных со вводом многих очень похожих друг на друга объявлений, особенно если вы поддерживаете префиксы всех основных браузеров.

В примере 19.4 продемонстрировано, как можно сразу воспользоваться и переходом, и трансформацией. С помощью CSS создается квадратный оранжевый элемент с неким текстом внутри, а псевдокласс `hover` указывает на то, что при проходе над этим объектом указателя мыши объект должен повернуться на 180° и изменить свой цвет с оранжевого на желтый (рис. 19.14).

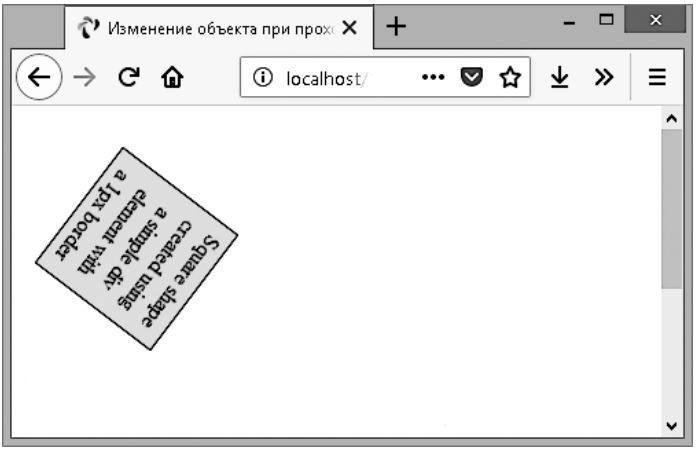

Рис. 19.14. Объект поворачивается и меняет цвет при прохождении над ним указателя мыши

Пример 19.4. Эффект перемещения, связанный с применением псевдокласса hover

```
<!DOCTYPE html>
<html>
  <head>
    <title>Перемещение при проходе мыши </title>
    <sty1e>
      #square {
        position          :absolute;
        top               :50px;
        left              :50px;
        width             :100px;
        height            :100px;
        padding           :2px;
        text-align        :center;
        border-width      :1px;
        border-style      :solid;
        background        :orange;
        transition        :all .8s ease-in-out;
        -moz-transition   :all .8s ease-in-out;
        -webkit-transition:all .8s ease-in-out;
        -o-transition     :all .8s ease-in-out;
        -ms-transition    :all .8s ease-in-out;
      }
      #square:hover {
        background        :yellow;
        -moz-transform    :rotate(180deg);
        -webkit-transform :rotate(180deg);
        -o-transform      :rotate(180deg);
        -ms-transform     :rotate(180deg);
        transform         :rotate(180deg);
      }
    </style>
  </head>
  <body>
```

```
    <div id='square'>
      Square shape<br>
      created using<br>
      a simple div<br>
      element with<br>
      a 1px border </div>
  </body>
</html>
```

Пример кода удовлетворяет требованиям всех разнообразных браузеров благодаря предоставлению версий объявлений, характерных для тех или иных браузеров. На всех самых последних браузерах (включая Internet Explorer 10 и выше) объект будет поворачиваться по часовой стрелке при прохождении над ним указателя мыши и в то же время станет медленно менять свой цвет с оранжевого на желтый.

CSS-переходы выполняются вполне продуманно, что выражается в том, что после прекращения перехода все плавно возвращается к своему исходному значению. Поэтому, если убрать указатель мыши до завершения перехода, он тут же пойдет в обратную сторону и начнет переход назад к своему исходному состоянию.

Вопросы

1. Чем занимаются операторы селектора атрибутов CSS3 ^=, $= и *=?
2. Какое свойство используется для указания размера фонового изображения?
3. С помощью какого свойства можно указать радиус границы?
4. Как можно задать перетекание текста по нескольким колонкам?
5. Назовите четыре функции, с помощью которых можно указать CSS-цвета.
6. Как можно создать серую тень под каким-нибудь текстом с диагональным отступом вправо и вниз на 5 пикселов и с размытостью 3 пиксела?
7. Как можно показать многоточием, что текст усечен?
8. Как включить в состав своей веб-страницы веб-шрифты Google?
9. Какое CSS-объявление нужно использовать для поворота объекта на 90°?
10. Как указать переход объекта таким образом, чтобы при изменении любого из его свойств переход осуществлялся сразу в линейном режиме в течение 0,5 с?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

20 Доступ к CSS из JavaScript

После того как вы поняли, что такое объектная модель документа — Document Object Model (DOM) и каскадные таблицы стилей — CSS, из этой главы вы узнаете, как к ним можно получать доступ непосредственно из кода JavaScript, что позволит вам создавать высокодинамичные и быстро реагирующие на действия пользователей сайты.

Мы также рассмотрим использование прерываний, что позволит создавать анимацию или предоставлять любой код, который должен продолжать работу на вебстранице (например, часы). Кроме того, я объясню, как в DOM добавляются новые элементы или удаляются существующие элементы, чтобы вам не приходилось заранее создавать элементы в HTML на тот случай, если коду JavaScript может чуть позже понадобиться получить к ним доступ.

Еще одно обращение к функции getElementById

В качестве помощи в работе с примерами, приводимыми в остальной части книги, я намереваюсь представить вам улучшенную версию функции `getElementById`, чтобы вы могли работать с элементами DOM и стилями CSS быстро и эффективно, не испытывая потребности включения таких сред, как jQuery.

Но чтобы избежать конфликтов со средами программирования, использующими символ `$`, я буду просто применять в качестве имени функции заглавную букву `O`, поскольку это первая буква слова Object (объект), а именно объект будет возвращаться при вызове этой функции (тот самый объект, представленный идентификатором ID, переданным функции).

Функция O

Основа функции O имеет следующий вид:

```
function O(i)
{
  return document.getElementByld(i)
}
```

Только этот код уже сокращает количество набираемого текста при вызове функции на 22 символа, но я намерен немного расширить функцию, позволяя передавать ей либо ID, либо объект, как показано в полной версии функции, представленной в примере 20.1.

Пример 20.1. Функция O

```
function O(i)
{
  return typeof i == 'object' ? i : document.getElementById(i)
}
```

Если функции передается объект, она просто возвращает его обратно. В противном случае она предполагает, что ей был передан ID, и возвращает объект, на который этот ID ссылается.

Но с какой стати мне захотелось добавить эту первую инструкцию, которая просто возвращает переданный ей объект?

Функция S

Ответ на данный вопрос станет ясным, когда вы посмотрите на вспомогательную функцию, названную S и показанную в примере 20.2, которую я вам предоставляю для упрощения доступа к стилевым свойствам (или CSS) объекта.

Пример 20.2. Функция S

```
function S(i)
{
  return O(i).style
}
```

Для этой функции имя S выбрано потому, что это первая буква слова Style, а функция выполняет задачу возвращения свойства стиля (или подчиненного объекта) того элемента, на который она ссылается. Поскольку встроенная функция O принимает либо ID, либо объект, вы можете передавать функции S как ID, так и объект.

Рассмотрим, что получится, когда мы возьмем -элемент с ID myobj и установим для цвета его текста значение green (зеленый):

```
<div id='myobj'>Some text</div>

<script>
```

```
  O('myobj').style.color = 'green'
</script>
```

Предыдущий код справится с этой задачей, но значительно проще будет вызвать новую функцию S:

```
S('myobj').color = 'green'
```

Теперь рассмотрим случай, при котором объект, возвращенный в результате вызова функции O, сохранен, к примеру, в объекте по имени fred:

```
fred = O('myobj')
```

Благодаря тому способу, который используется в работе функции S, мы можем для изменения цвета на зеленый вызвать и этот объект:

```
S(fred).color = 'green'
```

Это означает, что при желании получить доступ к объекту непосредственно или через его ID вы можете сделать это, передавая его либо функции O, либо функции S в зависимости от того, что вам нужно. Необходимо лишь помнить, что при передаче объекта (а не ID) ни в коем случае не следует брать его имя в кавычки.

Функция C

Вам уже предоставлены две простые функции, упрощающие доступ к любому элементу на веб-странице и любому свойству стиля элемента. Но иногда вам понадобится одновременный доступ более чем к одному элементу. Это можно сделать путем присваивания имени класса CSS каждому такому элементу, как показано в следующем примере, где для каждого элемента применяется класс myclass:

```
<div class='myclass'>Содержимое div-контейнера </div>
<p class='myclass'>Содержимое абзаца</p>
```

Если нужен доступ ко всем элементам страницы, использующим конкретный класс, можно обратиться к функции C (чье имя происходит от первой буквы в слове Class), показанной в примере 20.3. Она вернет массив, состоящий из всех объектов, которые соответствуют предоставленному имени класса.

Пример 20.3. Функция C

```
function C(i)
{
  return document.getElementsByClassName(i)
}
```

Для использования эту функцию следует просто вызвать, как показано далее, сохраняя возвращенный массив, чтобы иметь возможность получить доступ отдельно к каждому нужному элементу или (что чаще всего и бывает) ко всем элементам с помощью цикла:

```
myarray = C('myclass')
```

Теперь можете делать с возвращенными объектами все, что нужно, например установить для их свойства `textDecoration` значение подчеркивания — `'underline'`:

```
for (i = 0 ; i < myarray.length ; ++i)
  S(myarray[i]).textDecoration = 'underline'
```

Этот код осуществляет последовательный перебор объектов в `myarray[]`, а затем использует функцию `S` для ссылки на свойство стиля каждого объекта, задавая для свойства `textDecoration` значение `'underline'`.

Включение функций

Функции `O` и `S` используются во всей оставшейся части главы, поскольку делают код короче и понятнее. Поэтому я сохранил их в файле `OSC.js` (наряду с функцией `C`, поскольку я полагаю, что она принесет вам большую пользу) в папке Chapter 20 в сопутствующем архиве примеров, который вы можете загрузить с сайта http://lpmj.net.

Они могут быть включены в веб-страницу с помощью следующей инструкции. Ее предпочтительнее поместить в блок `<head>` где-нибудь перед любым сценарием, работа которого зависит от вызова этих функций:

```
<script src='OSC.js'></script>
```

Содержимое файла `OSC.js` показано в примере 20.4, где все убрано всего лишь в три строки.

Пример 20.4. Файл OSC.js

```
function O(i) {return typeof i == 'object' ? i : document.getElementById(i) }
function S(i) { return O(i).style                              }
function C(i) { return document.getElementsByClassName(i)   }
```

Обращение к свойствам CSS из JavaScript

Свойство `textDecoration`, использовавшееся в ранее показанном примере, представляет свойство CSS, имя которого в обычном виде содержит дефис: `text-decoration`. Но поскольку в JavaScript дефис зарезервирован для применения в качестве математического оператора, при доступе к свойству CSS, в имени которого используется дефис, этот дефис нужно опустить и перевести в верхний регистр символ, следовавший непосредственно за ним.

Еще одним примером может послужить свойство `font-size`, на которое в JavaScript при помещении после оператора точки ссылаются как на `fontSize`:

```
myobject.fontSize = '16pt'
```

Вместо этого можно предоставить более развернутый код и воспользоваться функцией `setAttribute`, которая поддерживает (и фактически требует) стандартное имя свойства CSS:

```
myobject.setAttribute('style', 'font-size:16pt')
```

Некоторые устаревшие версии Microsoft Internet Explorer в определенных ситуациях слишком разборчивы в применении JavaScript-стиля имен, принадлежащих свойствам CSS. Имеется в виду применение к ним специальных версий правил, в которых используются характерные для браузера префиксы -ms-.

Некоторые общие свойства

С помощью JavaScript вы можете изменить любое свойство любого элемента, имеющегося в веб-документе, примерно так же, как это делается с помощью CSS. Я уже показывал вам, как получить доступ к свойствам CSS, используя либо краткую форму JavaScript, либо функцию `setAttribute` (чтобы применить абсолютно такие же имена свойств, как и в CSS). Поэтому я не стану обременять вас детализацией всех этих сотен свойств. Вместо этого я покажу, как получить доступ к некоторым свойствам CSS, чтобы дать обзорное представление об их возможностях.

Сначала рассмотрим изменение нескольких свойств CSS из JavaScript, используя код примера 20.5, который в первую очередь загружает в себя три ранее упомянутые функции, затем создает `<div>`-элемент и, наконец, запускает инструкции JavaScript, находящиеся внутри блока `<script>` кода HTML с целью изменения некоторых атрибутов элемента `<div>` (рис. 20.1).

Пример 20.5. Обращение к свойствам CSS из JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Обращение к свойствам CSS</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <div id='object'>Div-объект</div>

    <script>
      S('object').border     = 'solid 1px red'
      S('object').width      = '100px'
      S('object').height     = '100px'
      S('object').background = '#eee'
      S('object').color      = 'blue'
      S('object').fontSize   = '15pt'
```

```
      S('object').fontFamily = 'Helvetica'
      S('object').fontStyle  = 'italic'
   </script>
  </body>
</html>
```

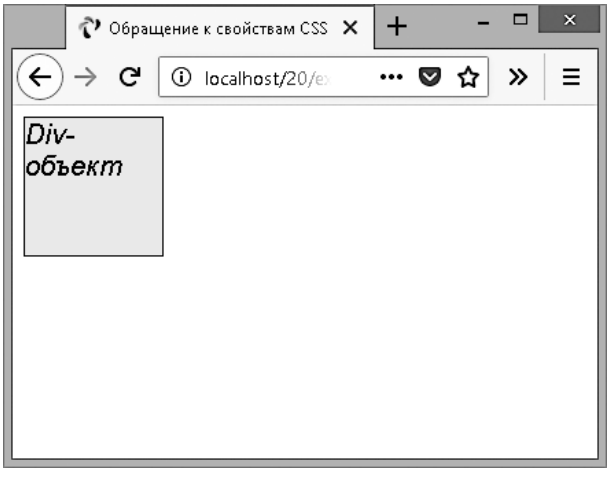

Рис. 20.1. Изменение стилей из JavaScript

От такого изменения свойств нет никакой практической пользы, поскольку можно так же легко включить код CSS непосредственно в атрибуты элемента, но скоро мы будем изменять свойства в ответ на действия пользователя, вот тогда и проявится настоящая эффективность сочетания JavaScript и CSS.

Другие свойства

JavaScript также открывает доступ к очень широкому диапазону других свойств, таких как ширина и высота окна браузера и любых появляющихся или присутствующих в браузере окон или фреймов, и к такой полезной информации, как родительское окно (если таковое имеется) и история URL-адресов, по которым осуществлялись визиты в текущем сеансе.

Все эти свойства доступны из объекта `window` через оператор «точка» (`.`) (например, `window.name`). В табл. 20.1 перечислены все эти свойства с описаниями.

Таблица 20.1. Свойства объекта window

Свойство	Устанавливает и (или) возвращает
closed	Возвращает булево значение, показывающее, было ли закрыто окно
defaultStatus	Устанавливает или возвращает исходный текст панели состояния окна

Свойство	Устанавливает и (или) возвращает
document	Возвращает объект документа для окна
frameElement	Возвращает элемент iframe, в который вставлено текущее окно
frames	Возвращает массив, состоящий из всех фреймов и i-фреймов окна
history	Возвращает для окна объект истории
innerHeight	Устанавливает или возвращает внутреннюю высоту области содержимого окна
innerWidth	Устанавливает или возвращает внутреннюю ширину области содержимого окна
length	Возвращает количество фреймов и i-фреймов окна
localStorage	Позволяет сохранять пары «ключ — значение» в браузере
location	Возвращает местоположение объекта в окне
name	Устанавливает или возвращает имя окна
navigator	Возвращает для окна объект-навигатор
opener	Возвращает ссылку на то окно, из которого было создано данное окно
outerHeight	Устанавливает или возвращает внешнюю высоту окна, включая панель инструментов и полосу прокрутки
outerWidth	Устанавливает или возвращает внешнюю ширину окна, включая панель инструментов и полосу прокрутки
pageXOffset	Возвращает количество пикселов, на которое был горизонтально прокручен документ от левого края окна
pageYOffset	Возвращает количество пикселов, на которое был вертикально прокручен документ от верхнего края окна
parent	Возвращает для окна объект родительского окна
screen	Возвращает для окна объект экрана
screenLeft	Возвращает координату x окна относительно экрана во всех последних браузерах, кроме Mozilla Firefox (для которого нужно использовать screenX)
screenTop	Возвращает координату y окна относительно экрана во всех последних браузерах, кроме Mozilla Firefox (для которого нужно применять screenY)
screenX	Возвращает координату x окна относительно экрана во всех последних браузерах, кроме Opera, который возвращает неправильное значение; не поддерживается в версиях Internet Explorer, предшествующих версии 9
screenY	Возвращает координату y окна относительно экрана во всех последних браузерах, кроме Opera, который возвращает неправильное значение; не поддерживается в версиях Internet Explorer, предшествующих версии 9
sessionStorage	Позволяет сохранять пары «ключ — значение» в веб-браузере
self	Возвращает текущее окно
status	Устанавливает или возвращает текст на панели состояния окна
top	Возвращает верхнее окно браузера

У некоторых из этих свойств есть следующие особенности.

- Свойства `defaultStatus` и `status` могут быть установлены, только если пользователи изменили настройки своих браузеров и разрешили их применение (что маловероятно).
- Содержимое объекта `history` не может быть прочитано (поэтому нельзя посмотреть, какие адреса посещались вашими визитерами), но этот объект поддерживает свойство `length`, чтобы определить длину истории, а также методы `back`, `forward` и `go` для переходов на указанные страницы в истории.
- Когда нужно узнать, какое пространство доступно в текущем окне браузера, следует просто прочитать значения свойств `window.innerHeight` и `window.innerWidth`. Я часто использую эти значения для размещения появляющихся в окне браузера диалоговых окон оповещения и подтверждения по центру.
- Объект `screen` поддерживает свойства, доступные только для чтения, — `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth` и `width`, поэтому отлично подходит для извлечения информации о дисплее пользователя.
- Mozilla Firefox не поддерживает свойства `screenLeft` и `screenTop`. Вместо них для этого браузера следует использовать свойства `screenX` и `screenY`.

Многие из этих свойств могут быть просто бесценными при позиционировании на мобильных телефонах и планшетных устройствах, поскольку дадут точную информацию об экранном пространстве, с которым придется работать, о типе используемого браузера и т. д.

Этого объема информации вполне достаточно для начала работы и для получения представления о многих новых и интересных приемах работы с JavaScript. Разумеется, существует намного больше доступных свойств и методов, которые могли бы быть рассмотрены в данной главе. Но теперь, когда вы знаете о том, как обращаться к свойствам и использовать их, вам нужен лишь информационный ресурс, на котором все они перечислены. Я рекомендую для начала обратиться к сайту http://www.javascriptkit.com/domref/index.shtml.

Встроенный JavaScript

Использование тегов `<script>` не единственный способ выполнения инструкций JavaScript. Получить доступ к JavaScript можно также из тегов HTML, что и делается для повышения динамической интерактивности.

Например, для быстрого добавления эффекта при прохождении указателя мыши над объектом можно воспользоваться таким же кодом, который показан в теге `<img>` в примере 20.6. Там изначально отображается картинка с яблоком, которая

при прохождении над ней указателя мыши заменяется картинкой с апельсином (а при выходе указателя за пределы картинки возвращается картинка с яблоком).

Пример 20.6. Использование встроенного JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Встроенный JavaScript</title>
  </head>
  <body>
    <img src='apple.png'
      onmouseover="this.src='orange.png'"
      onmouseout="this.src='apple.png'">
  </body>
</html>
```

Ключевое слово this

В предыдущем примере вы можете увидеть применение ключевого слова `this`. Оно заставляет JavaScript работать с названным объектом, а именно с тегом `<img>`. Результат можно увидеть на рис. 20.2, где указатель мыши только что прошел над картинкой с яблоком.

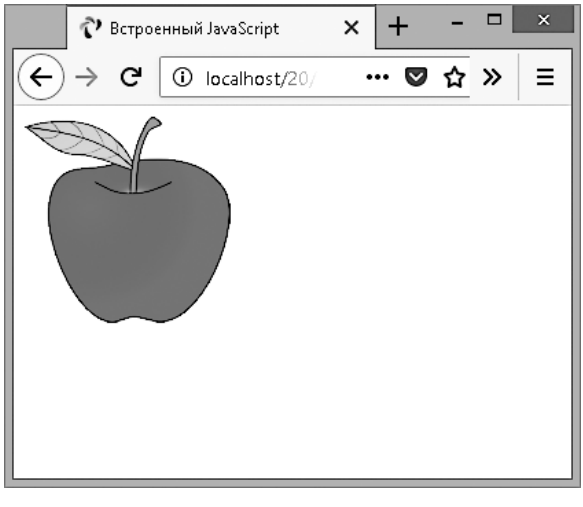

Рис. 20.2. Пример встроенного кода JavaScript, обрабатывающего прохождение указателя мыши над объектом

Когда ключевое слово this находится во встроенном вызове JavaScript, оно представляет вызываемый объект. А при использовании в методах класса оно представляет объект, к которому применяется метод.

Привязка событий к объектам в сценарии

Предыдущий код является эквивалентом предоставления тегу `<img>` идентификатора с последующей привязкой действий к событиям мыши этого тега, как в примере 20.7.

Пример 20.7. Невстроенный JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Невстроенный JavaScript</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <img id='object' src='apple.png'>

    <script>
      O('object').onmouseover = function() { this.src = 'orange.png' }
      O('object').onmouseout  = function() { this.src = 'apple.png' }
    </script>
  </body>
</html>
```

Этот код применяет ID объекта к тегу `<img>` в блоке HTML, а затем продолжает работать с ним отдельно в блоке JavaScript, прикрепив к каждому событию безымянную функцию.

Прикрепление к другим событиям

Какой бы JavaScript ни использовался, встроенный или отдельный, существуют события, к которым вы можете прикрепить действия. И активизировать тем самым множество дополнительных функций, которые можете предоставить своим пользователям. В табл. 20.2 перечислены эти события и указаны условия их наступления.

Таблица 20.2. События и условия их наступления

Событие	Условие его наступления
onabort	Загрузка изображения останавливается до ее завершения
onblur	Элемент теряет фокус*
onchange	Изменяется любая часть формы
onclick	Происходит щелчок кнопкой мыши на объекте
ondblclick	Происходит двойной щелчок кнопкой мыши на объекте
onerror	Обнаруживается ошибка JavaScript
onfocus	Элемент получает фокус

Событие	Условие его наступления
onkeydown	Нажата клавиша (включая Shift, Alt, Ctrl и Esc)
onkeypress	Нажата клавиша (исключая Shift, Alt, Ctrl и Esc)
onkeyup	Клавиша отпущена
onload	Объект загрузился
onmousedown	Над элементом нажата кнопка мыши
onmousemove	Указатель мыши проходит над элементом
onmouseout	Указатель мыши покидает элемент
onmouseover	Указатель мыши заходит на элемент со стороны
onmouseup	Отпускается кнопка мыши
onreset	Сбрасываются данные формы
onresize	Изменяются размеры окна браузера
onscroll	Документ прокручивается
onsubmit	Отправляется форма
onselect	Выделяется какой-нибудь текст
onunload	Удаляется документ

* Элемент с фокусом — тот, который был выбран или иным образом активизирован, например поле ввода.

События нужно прикреплять только к тем объектам, для которых в них имеется смысл. Например, объект, не являющийся формой, не будет реагировать на событие onsubmit.

Добавление новых элементов

Работая с JavaScript, вы можете манипулировать не только элементами и объектами, которые были предоставлены документу его кодом HTML. Вы можете создавать объекты по своему желанию, вставляя их в DOM.

Предположим, к примеру, что вам нужен новый элемент `<div>`. Способ добавления его к веб-странице показан в примере 20.8.

Пример 20.8. Вставка элемента в DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>Добавление элементов</title>
    <script src='OSC.js'></script>
```

```
  </head>
  <body>
    В этом документе содержится только этот текст.<br><br>

    <script>
      alert('Для добавления элемента нажмите кнопку OK')

      newdiv    = document.createElement('div')
      newdiv.id = 'NewDiv'
      document.body.appendChild(newdiv)

      S(newdiv).border = 'solid 1px red'
      S(newdiv).width  = '100px'
      S(newdiv).height = '100px'
      newdiv.innerHTML = "Это новый объект, вставленный в DOM"
      tmp              = newdiv.offsetTop

      alert('Для удаления элемента нажмите кнопку OK')
      pnode = newdiv.parentNode
      pnode.removeChild(newdiv)
      tmp = pnode.offsetTop
    </script>
  </body>
</html>
```

На рис. 20.3 показано, как этот код используется для добавления к веб-документу нового `<div>`-элемента.

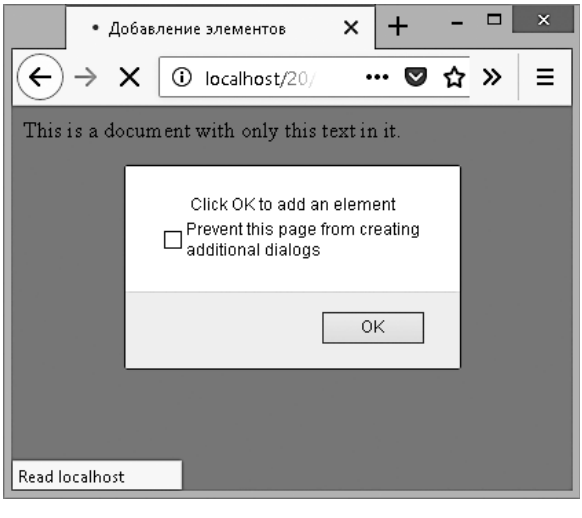

Рис. 20.3. Вставка нового элемента в DOM

Сначала новый элемент создается с помощью функции `createElement`, затем вызывается функция `appendChild` и элемент вставляется в DOM.

После этого элементу присваиваются различные свойства, включая текст для его свойства `innerHTML` (внутреннего HTML). А затем, чтобы обеспечить немедленное отображение нового элемента на экране, значение его свойства `offsetTop` считывается во временную переменную `tmp`. Это заставляет DOM обновиться и вывести элемент на экран в любом браузере, который в противном случае выдержал бы паузу, прежде чем это сделать. В частности, это касается Internet Explorer.

Этот новый созданный элемент точно такой же, как если бы он был включен в исходный HTML, и он открывает доступ к аналогичным свойствам и методам.

Иногда я использую технологию создания новых элементов, когда хочу создать окно, появляющееся в окне браузера, потому что она не зависит от наличия запасных -элементов, доступных в DOM.

Удаление элементов

Вы можете также удалить элементы из DOM, включая те, которые не были вставлены с помощью кода JavaScript. Это даже проще, чем добавить элемент. Если предположить, что удаляется объект `element`, то это делается следующим образом:

```
element.parentNode.removeChild(element)
```

Этот код обращается к объекту `parentNode` элемента, поэтому он может удалить элемент из этого узла. Затем он вызывает метод этого родительского объекта `removeChild`, передавая ему удаляемый объект. Но чтобы обеспечить немедленное обновление DOM во всех браузерах, возможно, будет предпочтительнее заменить предыдущую инструкцию следующим кодом:

```
pnode = element.parentNode
pnode.removeChild(element)
tmp   = pnode.offsetTop
```

Первая инструкция помещает копию `element.parentNode` (родительского элемента объекта) в переменную `pnode`. Третья инструкция (после того как дочерний элемент удаляется второй инструкцией) считывает значение свойства `offsetTop` родительского объекта во временную переменную `tmp`, гарантируя тем самым полное обновление DOM.

Альтернативы добавлению и удалению элементов

Вставка элемента предназначена для добавления к веб-странице абсолютно нового объекта. Но если вы намерены только скрывать и показывать объекты в соответствии с наступлением события `onmouseover` или какого-нибудь другого события, не забудьте, что есть пара свойств CSS, которые могут использоваться для этой цели без принятия таких радикальных мер, как создание и удаление элементов DOM.

Например, когда нужно сделать элемент невидимым, но оставить его на месте (оставляя на своих местах все окружающие его элементы), можно просто установить для свойства `visibility` объекта значение `'hidden'`:

```
myobject.visibility = 'hidden'
```

А для повторного отображения объекта можно воспользоваться следующим кодом:

```
myobject.visibility = 'visible'
```

Можно также свернуть элемент, чтобы он занимал нулевую ширину и высоту (и чтобы все окружающие его объекты заняли освободившееся пространство):

```
myobject.display = 'none'
```

Для последующего восстановления элемента в его исходных размерах можно написать такой код:

```
myobject.display = 'block'
```

И конечно же, в вашем распоряжении всегда есть свойство `innerHTML`, с помощью которого можно изменить код HTML, примененный к элементу. Например:

```
mylement. innerHTML = '<b>Замена HTML</b>'
```

Можно также воспользоваться упомянутой ранее функцией `O`:

```
O('someid').innerHTML = 'Новое содержимое'
```

Можно заставить элемент показаться исчезнувшим:

```
O('someid').innerHTML = ''
```

Не забывайте обо всех других полезных свойствах CSS, к которым можно обратиться из JavaScript. Например, для переключения объекта из видимого в невидимое состояние и обратно можно воспользоваться свойством непрозрачности, а для изменения размеров объекта можно изменить значения свойств width и height. И конечно же, применяя для свойства position значения absolute, static, fixed, sticky или relative, вы можете даже поместить объект в любое место окна браузера (или снаружи).

Использование прерываний

JavaScript предоставляет доступ к *прерываниям*, методу, с помощью которого можно попросить браузер вызвать ваш код после определенного периода времени или даже продолжать вызовы через указанные интервалы времени. Это дает вам средства обработки фоновых задач, таких как обмен данными с помощью AJAX, или даже такие средства, как анимация веб-элементов.

Существует два типа прерываний — `setTimeout` и `setInterval`, сопровождающихся функциями `clearTimeout` и `clearInterval` для их выключения.

Использование функции setTimeout

При вызове функции `setTimeout` передается код JavaScript или имя функции и значение в миллисекундах, отображающее продолжительность задержки запуска кода на выполнение:

```
setTimeout(dothis, 5000)
```

Ваша функция `dothis` может иметь следующий вид:

```
function dothis()
{
  alert('Это ваш будильник!');
}
```

Как ни удивительно, вы не можете просто указать alert() (с круглыми скобками) в качестве функции, вызываемой setTimeout, потому что функция будет тут же выполнена. Передавать имя функции, чтобы код был выполнен только по истечении указанного времени, можно только без круглых скобок, служащих для указания аргументов (например, alert).

Передача строки

Если исполняемой функции нужно передать аргумент, то функции `setTimeout` можно также передать строковое значение, которое не будет выполняться, пока не наступит нужное время. Например:

```
setTimeout("alert('Hello!T\ 5000)
```

Фактически, если после каждой инструкции ставить точку с запятой, можно передать столько строк кода JavaScript, сколько нужно:

```
setTimeout("document.write('Starting'); alert('Hello!')", 5000)
```

Повторение тайм-аутов

Для предоставления повторяющихся прерываний, создаваемых функцией `setTimeout`, некоторые программисты используют технологию вызова функции `setTimeout` из кода, вызываемого этой же функцией, как в следующем примере, который инициирует бесконечный цикл вывода окон предупреждений:

```
setTimeout(dothis, 5000) function dothis()
{
  setTimeout(dothis, 5000) alert('Я вас раздражаю!')
}
```

Теперь окно предупреждения будет появляться каждые 5 с. Я не рекомендую вам запускать этот конкретный пример на выполнение (кроме как в целях

тестирования), поскольку для остановки его работы вам, скорее всего, придется закрывать браузер!

Еще один вариант предполагает использование рассматриваемой далее функции setInterval.

Отмена тайм-аута

После установки тайм-аута вы можете отменить его, если предварительно сохранили значение, возвращенное при начальном вызове функции `setTimeout`:

```
handle = setTimeout(dothis, 5000)
```

Теперь, когда у вас есть это значение в переменной `handle`, вы можете отменить прерывание в любой момент, вплоть до истечения назначенного срока:

```
clearTimeout(handle)
```

В результате этого прерывание полностью забывается и код, назначенный ему для выполнения, никогда не выполняется.

Функция setInterval

Самый простой способ установки регулярных прерываний заключается в использовании функции `setInterval`. Она работает точно так же, как и описанная выше, за исключением того, что, проявив себя после интервала, заданного вами в миллисекундах, она сделает это снова, после того как этот же интервал снова пройдет, и так до бесконечности, пока вы ее не остановите.

В примере 20.9 эта функция используется для вывода в браузере простых часов, показанных на рис. 20.4.

Пример 20.9. Часы, созданные с помощью прерываний

```
<!DOCTYPE html>
<html>
  <head>
    <title>Использование setInterval</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    Текущее время: <span id='time'>00:00:00</span><br>

    <script>
      setInterval("showtime(O('time'))", 1000)

      function showtime(object)
```

```
      {
        var date = new Date()
        object.innerHTML = date.toTimeString().substr(0,8)
      }
    </script>
  </body>
</html>
```

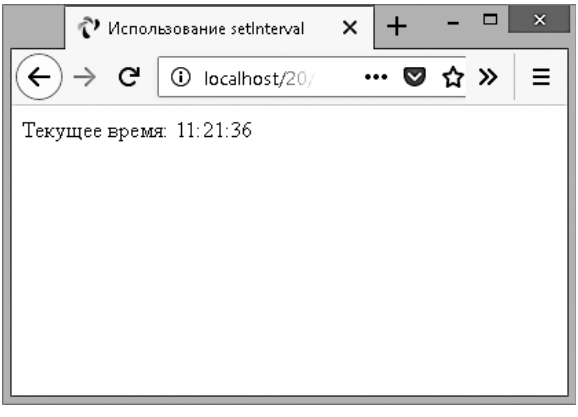

Рис. 20.4. Поддержка показаний правильного времени с помощью прерываний

При каждом вызове функции `ShowTime` она присваивает объекту `date` текущее время и дату с помощью вызова функции `Date`:

```
var date = new Date()
```

Затем свойству `innerHTML` объекта, переданного функции `showtime` (то есть `object`), присваивается значение текущего времени в часах, минутах и секундах, как определено вызовом функции `toTimeString`. В результате возвращается строка `09:57:17 UTC+0530`, которая затем усекается до первых восьми символов с помощью вызова функции `substr`:

```
object.innerHTML = date.toTimeString().substr(0.8)
```

Использование функции

Чтобы воспользоваться этой функцией, сначала нужно создать объект, чье свойство `innerHTML` будет применено для отображения времени, как в следующем коде HTML:

```
Текущее время: <span id='time'>00:00:00</span>
```

Затем в блоке кода `<script>` вызовите функцию `setInterval`:

```
setInterval("showtime(O('time'))", 1000)
```

Затем сценарий передает функции `setInterval` строку, содержащую следующую инструкцию, настроенную на выполнение один раз в секунду (каждые 1000 мс):

```
showtime(O('time'))
```

В том редком случае, когда кто-нибудь отключил в своем браузере JavaScript (что некоторые делают из соображений безопасности), ваш JavaScript не запустится и пользователь просто увидит исходное значение `00:00:00`.

Отмена интервала

Чтобы остановить повторяющийся интервал, при первой установке интервала путем вызова функции `setInterval` вы должны пометить для себя в переменной `handle` дескриптор этого интервала:

```
handle = setInterval("showtime(O('time'))", 1000)
```

Теперь можно остановить часы в любое время, сделав следующий вызов:

```
clearInterval(handle)
```

Можно также настроить таймер на остановку через определенный период времени:

```
setTimeout("clearInterval(handle)", 10000)
```

Эта инструкция выдаст прерывание через 10 с (10 000 мс), которое очистит повторяющиеся интервалы.

Использование прерываний для анимации

Путем сочетания нескольких свойств CSS с повторяющимся прерыванием можно создавать всевозможные анимации и эффекты.

Код в примере 20.10 перемещает прямоугольник по верхней части окна браузера, все время увеличивая его в размерах (рис. 20.5). Когда значение переменной `LEFT` сбрасывается в `0`, анимация начинается снова.

Пример 20.10. Простая анимация

```
<!DOCTYPE html>
<html>
  <head>
    <title>Простая анимация</title>
    <script src='OSC.js'></script>
    <style>
      #box {
        position  :absolute;
        background:orange;
        border    :1px solid red; }
    </style>
  </head>
```

```
  <body>
    <div id='box'></div>

    <script>
      SIZE = LEFT = 0

      setInterval(animate, 30)

      function animate()
      {
        SIZE += 10
        LEFT += 3

        if (SIZE == 200) SIZE = 0
        if (LEFT == 600) LEFT = 0

        S('box').width  = SIZE + 'px'
        S('box').height = SIZE + 'px'
        S('box').left   = LEFT + 'px'
      }
    </script>
  </body>
</html>
```

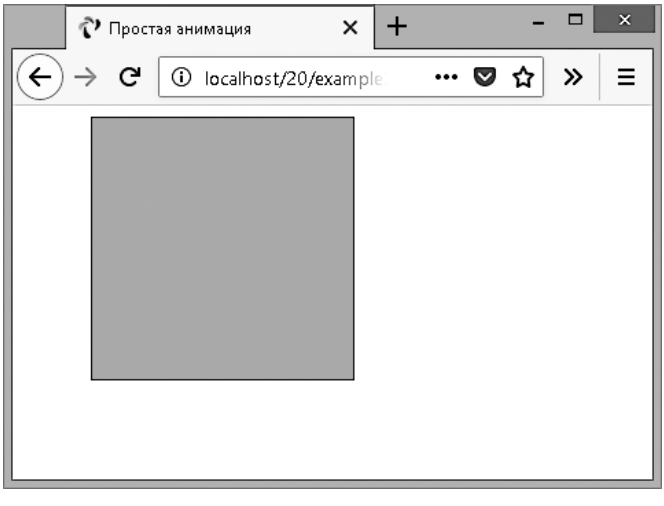

Рис. 20.5. Объект плавно движется слева, одновременно меняя свой размер

В блоке `<head>` документа объекту `box` устанавливается цвет фона `orange` (оранжевый) со значением его границы (`border`) `1px solid red`, а его свойству позиционирования `position` задается значение `absolute`, и следующий далее код анимации может точно задавать ему позицию в соответствии с вашими желаниями.

Затем в функции `animate` происходит постоянное обновление глобальных переменных `SIZE` и `LEFT`, а их значения применяются к атрибутам стиля `width`, `height` и `left`

объекта box (с добавлением после каждого значения строки 'px' для указания, что значение в пикселах), таким образом анимируя объект с частотой один раз каждые 30 мс. Тем самым задается скорость 33,33 кадра в секунду (1000 / 30 мс).

Вопросы

1. Для чего предназначены функции O, S и C?
2. Назовите два способа изменения CSS-атрибута объекта.
3. Какие свойства предоставляют доступную в окне браузера ширину и высоту?
4. Как можно задать какие-нибудь действия при прохождении указателя мыши над объектом, а затем при выходе за границы объекта?
5. Какая функция JavaScript создает новые элементы и какая функция добавляет их к DOM?
6. Как сделать элемент а) невидимым и б) сжатым до нулевых размеров?
7. Какая функция задает одиночное событие в будущем времени?
8. Какая функция устанавливает повторяющиеся события через указанный интервал времени?
9. Как можно освободить элемент от его места на веб-странице, чтобы он мог перемещаться?
10. Какая задержка должна быть установлена между событиями (в миллисекундах) для получения скорости анимации 50 кадров в секунду?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

21 Введение в jQuery

При всей гибкости и эффективности JavaScript, а также при всем изобилии имеющихся в этом языке встроенных функций все же сохраняется потребность в дополнительных уровнях кода, позволяющих упростить, к примеру, получение эффектов анимации, обработку событий и применение технологии асинхронного обмена данными, то есть сделать то, чего нельзя достичь применением обычных средств JavaScript или CSS.

Более того, с годами, вследствие различных браузерных войн, то появлялись, то исчезали досаждающие и раздражающие несовместимости браузеров, ярко проявлявшиеся временами на различных платформах и в программах.

В результате этого гарантировать одинаковый внешний вид веб-страниц во всех устройствах порой можно было, только применяя требовавший утомительной разработки код JavaScript, учитывающий все расхождения всей линейки браузеров и их версий, выпущенных за последние годы. Одним словом, сложилась кошмарная ситуация.

Для заполнения пробелов было разработано множество библиотек функций (многие из которых также предоставляли легкую привязку к DOM), предназначенных для сведения к минимуму различий между браузерами и для содействия асинхронному обмену данными, а также работе с событиями и анимацией. В их число входят такие наиболее предпочтительные библиотеки, как *AngularJS*, *jQuery*, *MooTools*, *Prototype*, *script.aculo.us* и *YUI* (существует и множество других библиотек).

Почему же именно jQuery?

В книге выделено место только для одной библиотеки, поэтому я выбрал наиболее широко используемую библиотеку jQuery, которая сегодня, по сведениям http://w3techs.com, установлена более чем на 73 % сайтов и востребована шире, чем все ее конкуренты, вместе взятые (насколько можно судить по диаграммам источника).

Кстати, если есть желание посмотреть на столбиковую диаграмму, отображающую процентное соотношение востребованности различных библиотек на любой текущий момент времени, введите в поисковой строке сайта similartech.com слово `javascript`.

Используя jQuery, вы получаете не только кросс-браузерную совместимость весьма высокого уровня (включая совместимость с Internet Explorer), но и быстрый и легкий доступ к операциям с HTML и DOM, возможность использования специальных функций для непосредственной работы с CSS, управления событиями, мощные средства для создания профессиональных эффектов и анимации, а также функции для управления асинхронным обменом данными с сервером. Кроме того, jQuery является основой для широкого круга дополнительных модулей и других вспомогательных программ.

Конечно же, использования jQuery вам никто не навязывает, и некоторые борцы за чистоту языка программирования никогда не используют библиотеку, предпочитая создавать собственные специализированные наборы функций (и в этом есть свой резон, например не нужно будет дожидаться, пока другие люди исправят замеченные вами недоделки, можно будет разрабатывать собственные средства безопасности и т. д.). Но библиотека jQuery уже выдержала проверку временем, и если вы захотите с пользой потратить время на ее изучение и получить возможность делать высококачественные веб-страницы в самые короткие сроки, из этой главы вы узнаете, как можно приступить к использованию этой библиотеки.

Включение jQuery

Есть два способа включения jQuery в ваши веб-страницы. Можно перейти на сайт jQuery (https://code.jquery.com/jquery/), выбрать нужную версию, загрузить ее на свой веб-сервер и ссылаться на нее из тега `<script>` в своих HTML-файлах. Или же можно воспользоваться находящейся в свободном доступе сетью доставки контента — Content Delivery Network (CDN) и просто указать ссылку на нужную вам версию.

jQuery выпускается в соответствии с условиями MIT-лицензии, в которой не содержится практически никаких ограничений на ваши дальнейшие действия. Любой проект jQuery можно свободно использовать в любом другом проекте (даже коммерческом) при условии, что заголовок с указанием авторских прав останется нетронутым.

Выбор подходящей версии

Перед тем как решить, стоит ли загружать jQuery и использовать ее функции непосредственно или же воспользоваться CDN, нужно выбрать версию jQuery. В большинстве случаев выбор очевиден, поскольку вы просто отдадите предпо-

чтение наиболее свежей версии. Но если есть намерение использовать конкретные браузеры или же вы поддерживаете устаревший сайт, работа которого зависит от определенной версии jQuery, то последний выпуск этой библиотеки может вам не подойти.

В отличие от большинства других программных средств, для использования которых вы просто загружаете и устанавливаете самую новую из доступных версий, jQuery со временем совершенствовалась с учетом изменения движущих сил на рынке различных версий браузеров с их свойствами и недочетами.

В то же время в jQuery вносили различные усовершенствования, которые могли изменить работу ее новых версий на тех сайтах, которые были специально адаптированы под конкретную версию (и под все сопутствующие ей особенности).

Разумеется, каждая более новая версия является улучшением предыдущей, и вероятность того, что вносимые усовершенствования коснутся всех основ, постоянно возрастает. Но пока вы полностью не протестируете новую версию и не убедитесь в том, что операции, играющие важную роль для вашего сайта, выполняются точно так же, лучше все же продолжать использовать прежнюю версию.

Различные разновидности jQuery

На данный момент существуют три ветви jQuery: 1.x, 2.x и 3.x, каждая из которых разработана для разных сред.

Версия 1.x была первым стабильным выпуском jQuery. Этот выпуск поддерживает устаревшие браузеры, которые даже уже больше не поддерживаются своими разработчиками. Если ожидается большой наплыв посетителей с устаревшими браузерами, то нужно пользоваться именно этой версией (как уже упоминалось, лучшей, наверное, будет версия 1.12).

С целью повышения общей производительности jQuery и уменьшения размера файла библиотеки в версии 2.x была исключена поддержка Internet Explorer 6–8. Эта версия быстрее и меньше версии 1.x, но не поддерживает устаревшие браузеры. Поскольку компания Microsoft прекратила поддержку Windows XP, можно с уверенностью предположить, что ваши посетители будут пользоваться браузером, совместимым с версией 2.x, если вы не будете располагать иными сведениями.

В общих чертах каждая новая версия jQuery поддерживает следующие версии браузеров:

- Chrome: (текущая – 1) и текущая;
- Edge: (текущая – 1) и текущая;
- Firefox: (текущая – 1) и текущая;

- Internet Explorer: 9+;
- Safari: (текущая – 1) и текущая;
- Opera: текущая.

Если нужна поддержка устаревших браузеров вроде Internet Explorer 6–8, Opera 12.1x или Safari 5.1+, разработчики jQuery рекомендуют воспользоваться версией 1.12. Исчерпывающие подробности поддержки различных версий можно найти по адресу http://jquery.com/browser-support/. В данной редакции книги я остановился на версии 3.2.1.

Сжатые или редактируемые

Нужно также решить, какую версию jQuery вам хотелось бы использовать: минимальную по размеру (сжатую, чтобы свести к минимуму требуемую полосу пропускания сети и сократить время загрузки) или несжатую (возможно, по причине того, что вам хочется вносить в нее самостоятельные правки, на что вы имеете полное право). Как правило, наиболее удачным выбором считается минимальная по размеру версия, но большинство веб-серверов поддерживают архиватор gzip, позволяющий выполнять сжатие и распаковку на лету, поэтому данный вопрос теряет свою актуальность (хотя нужно учесть, что из минимизированной версии, кроме всего прочего, удалены все комментарии).

Загрузка

На сайте jquery.com/download каждая последняя выпущенная версия jQuery фигурирует в списке как в сжатой, так и в несжатой форме. Все прошлые выпуски можно также найти на веб-сайте https://code.jquery.com/jquery/. Облегченные версии jQuery, попадающиеся на странице загрузки, в целях экономии пространства исключают функции асинхронного обмена данными, поэтому при необходимости применения данной технологии с использованием jQuery этих версий следует избегать.

Вам остается всего лишь выбрать нужную версию, щелкнуть правой кнопкой мыши на соответствующей ссылке и сохранить версию на своем жестком диске. Оттуда ее можно будет выгрузить на ваш веб-сервер, а затем включить в веб-страницу с помощью `<script>`-тегов примерно таким образом (для минимизированной версии выпуска 3.2.1):

```
<script src='http://myserver.com/jquery-3.2.1.min.js'></script>
```

Если ранее вам не приходилось пользоваться jQuery (и никаких специальных требований на ее счет у вас не имеется), то загружайте минимизированную версию или же установите показанную в слеующем разделе CDN-ссылку на эту библиотеку.

Использование сети доставки контента

Библиотека jQuery поддерживается несколькими сетями доставки контента (CDN). Если вы пользуетесь одной из них, то можете избавить себя от хлопот, связанных с загрузкой новых версий, и выкладывать их на сервер, просто указав прямые ссылки на URL-адреса, поддерживаемые этими сетями.

Ко всему прочему, эти сети предоставляют свои услуги совершенно бесплатно и обычно используют каналы с высокой пропускной способностью, которые, возможно, являются самыми скоростными на свете. Кроме того, CDN-сети обычно хранят свой контент в нескольких различных географических пунктах и предоставляют требуемый файл с ближайшего к вам сервера, гарантируя тем самым наиболее быструю из возможных доставку.

В общем, если вам не нужно вносить изменения в исходный код jQuery (для чего требуется его размещение на ваших собственных веб-серверах) и у ваших пользователей гарантированно имеется живое интернет-соединение, то, скорее всего, наилучшим вариантом будет использование CDN-сетей. Тем более что пользоваться ими довольно просто. Достаточно знать имя нужного файла и используемого для его загрузки корневого каталога CDN. Например, все текущие и предыдущие версии можно получить через CDN-сеть, которая используется библиотекой jQuery, с помощью следующего кода:

```
<script src='http://code.jquery.com/jquery-3.2.1.min.js'></script>
```

Основной каталог доступен по адресу http://code.jquery.com/, и за ним следует просто дописать имя нужного для включения файла (в данном случае это `jquery-3.2.1.min.js`).

Библиотеку jQuery предоставляют в своих сетях как Microsoft, так и Google, поэтому для ее включения можно воспользоваться любым из следующих двух вариантов:

```
<script src='http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.2.1.min.js'></script>
<script src='http://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'>
</script>
```

В случае использования Microsoft CDN (aspnetcdn.com) в URL-адресе сначала нужно указать основной каталог `ajax.aspnetcdn.com/ajax/jQuery/`, а за ним — имя требуемого файла.

Но для Google нужно разбить имя файла (например, `jquery-3.2.1.min.js`) на имя каталога и имя файла таким вот образом: `3.2.1/jquery.min.js`. А перед этим поставить строку `ajax.googleapis.com/ajax/libs/jquery/`.

Дополнительным преимуществом применения CDN-сетей является то, что ими пользуется большинство других сайтов, поэтому библиотека jQuery может уже находиться в кэше пользовательского браузера и ее, может быть, даже не придется доставлять заново. При практически более чем 73%-ной востребованности jQuery другими сайтами тем самым может быть сэкономлен большой объем ценных сетевых ресурсов и времени.

Заказная сборка jQuery

Если есть настоятельная необходимость свести объем данных, загружаемых вебстраницей, к минимуму, то можно воспользоваться jQuery, создав специальную сборку этой библиотеки, включающую только те функции, которые будут использоваться вашим сайтом. При ее доставке полагаться на CDN-сеть нельзя, но при подобных обстоятельствах вы, наверное, все равно не станете планировать использование этой сети.

Для создания собственной заказной сборки jQuery нужно зайти на сайт projects.jga.me/jquery-builder и выставить флажки возле тех модулей, которые вам нужны, сняв их с ненужных модулей. Затем заказная версия jQuery будет загружена в отдельную вкладку или окно, откуда ее можно будет скопировать и вставить в требуемое место.

Синтаксис jQuery

Больше всего людей, ранее незнакомых с jQuery, удивляет символ `$`, который действует как фабричный метод jQuery. Он был выбран из расчета допустимости в JavaScript, краткости и отличия от имен обычной переменной, объекта или функции (метода).

Этим символом обозначается вызов функции jQuery (что также при желании можно сделать). Замысел его использования заключается в сохранении краткости и приятного внешнего вида кода, а также избавлении от излишнего набора текста при каждом обращении к jQuery. Кроме того, при виде этого символа другие, ранее незнакомые с вашим кодом разработчики сразу же понимают, что в коде используется jQuery (или подобная ей библиотека).

Простой пример

В наипростейшем виде обращение к jQuery осуществляется набором символа `$`, за которым следуют заключенный в скобки селектор, точка и метод, применяемый к выбранному элементу (или элементам).

Например, для изменения семейства шрифтов всех абзацев на моноширинное можно воспользоваться следующей инструкцией:

```
$('p').css('font-family', 'monospace')
```

А для добавления границы к элементу `<code>` можно применить такую инструкцию:

```
$('code').css('border', '1px solid #aaa')
```

Взглянем на часть полноценного примера 21.1, где фрагменты, относящиеся к использованию jQuery, выделены полужирным шрифтом.

Пример 21.1. Простой пример применения jQuery

```
<!DOCTYPE html>
<html>
  <head>
    <title>Первый пример jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    В jQuery в качестве имен функции используются
    либо <code>$()</code>, либо <code>jQuery()</code>.
    <script>
      $('code').css('border', '1px solid #aaa')
    </script>
  </body>
</html>
```

После загрузки этого примера в браузер будет получен результат, показанный на рис. 21.1. Разумеется, конкретно эта инструкция просто подменяет собой то, что можно сделать с помощью обычного кода CSS, но я хотел показать синтаксис jQuery, поэтому пока не стал ничего усложнять.

Еще один способ выдачи этой команды заключается в вызове функции jQuery (которая работает точно так же, как и $):

```
jQuery('code').css('border', '1px solid #aaa')
```

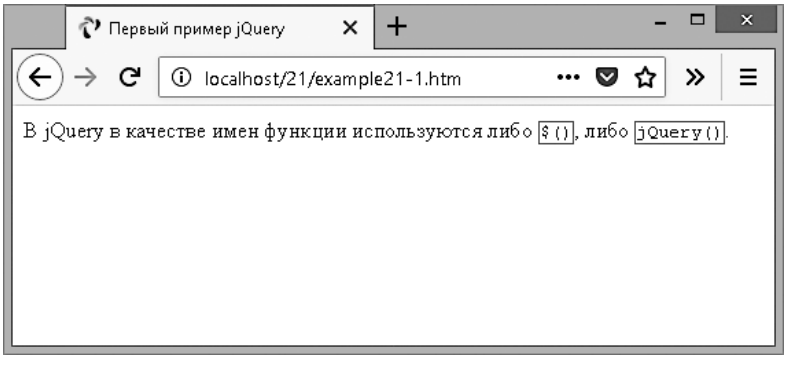

Рис. 21.1. Изменение элементов с помощью jQuery

Как избежать конфликтов библиотек

Если наряду с jQuery используются и другие библиотеки, может оказаться, что в них определены собственные функции `$`. Для решения данной проблемы можно в отношении этого символа вызвать метод `noConflict`, который освобождает

этот символ от управляющей функции, позволяя другим библиотекам воспользоваться им:

```
$.noConflict()
```

После этого для доступа к jQuery следует вызывать функцию `jQuery`. Или же использование символа `$` можно подменить именем объекта по вашему выбору:

```
jq = $.noConflict()
```

Теперь в тех местах, где прежде применялся символ `$`, для вызова jQuery можно воспользоваться ключевым словом `jq`.

Чтобы отличать объекты jQuery и отслеживать их отдельно от объектов стандартных элементов, некоторые разработчики устанавливают символ $ в виде префикса перед любым объектом, созданным с помощью jQuery (что делает их похожими на переменные PHP!).

Селекторы

После того как вы увидели, насколько просто можно включить jQuery в вебстраницу и обратиться к функциям этой библиотеки, перейдем к рассмотрению используемых в ней селекторов, которые (я уверен, что вы будете рады это узнать) работают точно так же, как CSS. По сути, их применение является основой работы большинства функций jQuery.

Вам остается лишь подумать о том, как бы вы оформили стиль одного или нескольких элементов с применением CSS, а затем можете использовать тот же самый селектор (или селекторы) для применения операций jQuery к этим выбранным элементам. Это означает, что вы можете воспользоваться селекторами элементов, селекторами идентификаторов, селекторами классов и любыми их сочетаниями.

Метод css

Чтобы объяснить применение селекторов в jQuery, сначала посмотрим на один из более фундаментальных методов jQuery, `css`, с помощью которого можно динамически менять любое свойство CSS. Этому методу передаются два аргумента: имя свойства, к которому осуществляется обращение, и значение, которое к этому свойству применяется:

```
css('font-family', 'Arial')
```

Как будет показано в следующих разделах, сам по себе этот метод применять невозможно, поскольку его нужно использовать в селекторе jQuery, который выберет

один или несколько элементов, чьи свойства должны быть изменены этим методом. В следующем примере содержимому всех <p>-элементов предписывается отображение с полным выравниванием по ширине:

```
$('p').css('text-align', 'justify')
```

Метод `css` можно также использовать для возвращения (а не для установки) вычисленного значения, для чего ему предоставляется только имя свойства (а второй аргумент опускается). В этом случае возвращается значение первого же соответствующего селектору элемента. Например, выполнение следующего кода приведет к возвращению цвета текста того элемента, чей идентификатор (ID) имеет значение `elem`, и это значение будет в том же формате, в котором цвет задается при применении метода `rgb`:

```
color = $('#elem').css('color')
```

Следует помнить, что возвращаемое значение является вычисленным. Иными словами, jQuery будет вычислять и возвращать значение, используемое браузером на момент вызова метода, а не то исходное значение, которое могло быть присвоено свойству посредством таблицы стилей или любым другим способом.

Следовательно, если текст, к примеру, показан синим цветом, значением, присвоенным переменной `color` в предыдущей инструкции, будет `rgb(0, 0, 255)`, даже если цвет изначально был установлен с использованием имени цвета `blue` или с использованием строк шестнадцатеричных чисел `#00f` или `#0000ff`. Но это вычисленное значение всегда будет в форме, которая может быть снова назначена элементу (или любому другому элементу) при использовании в качестве второго аргумента метода `css`.

К любым вычисленным размерам, возвращаемым этим методом, нужно относиться осмотрительно, поскольку в зависимости от текущих установок свойства box-sizing (см. главу 19) они могут быть, а могут и не быть именно тем, что вы ожидаете получить. Когда нужно получить или установить значения ширины и высоты без учета значения свойства box-sizing, нужно использовать методы width и height (и родственные им), рассматриваемые в разделе «Изменение размеров изображения».

Селектор элемента

Для выбора элемента, обрабатываемого с помощью jQuery, нужно просто указать его имя внутри круглых скобок, следующих за символом `$` (или за именем функции jQuery). Например, если нужно изменить цвет фона всех элементов `<blockquote>`, можно воспользоваться следующей инструкцией:

```
$('blockquote').css('background', 'lime')
```

Селектор идентификатора

Ссылаться на элементы можно также по их идентификаторам (ID), если перед именем идентификатора поместить символ #. Следовательно, чтобы, к примеру, добавить границу к элементу с идентификатором `advert`, можно воспользоваться такой инструкцией:

```
$('#advert').css('border', '3px dashed red')
```

Селектор класса

Можно также воздействовать на группу элементов в соответствии с используемым ею классом. Например, для подчеркивания всех элементов, применяющих класс `new`, можно воспользоваться следующей инструкцией:

```
$('.new').css('text-decoration', 'underline')
```

Сочетание селекторов

Как и при использовании CSS, селекторы можно сочетать друг с другом, составляя единый jQuery-выбор, для чего, как в следующем примере, применяются запятые:

```
$('blockquote, #advert, .new').css('font-weight', 'bold')
```

В примере 21.2 все типы селекторов собраны вместе, а инструкции jQuery выделены полужирным шрифтом. Результат выполнения кода примера показан на рис. 21.2.

Пример 21.2. Использование jQuery с различными селекторами

```
<!DOCTYPE html>
<html>
  <head>
    <title>Второй пример jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <blockquote> При всей гибкости и эффективности JavaScript, а также
      при всем изобилии имеющихся в этом языке встроенных функций все же
      сохраняется потребность в дополнительных уровнях кода, позволяющих
      упростить, к примеру, получение эффектов анимации, обработку событий
      и применение технологии асинхронного обмена данными, то есть сделать
      то, чего нельзя достичь применением обычных средств JavaScript
      или CSS .</blockquote>
    <div id='advert'>Это реклама</div>
    <p>Это мой <span class='new'>новый</span> сайт</p>
    <script>
      $('blockquote').css('background', 'lime')
      $('#advert').css('border', '3px dashed red')
```

```
      $('.new').css('text-decoration', 'underline')
      $('blockquote, #advert, .new').css('font-weight', 'bold')
    </script>
  </body>
</html>
```

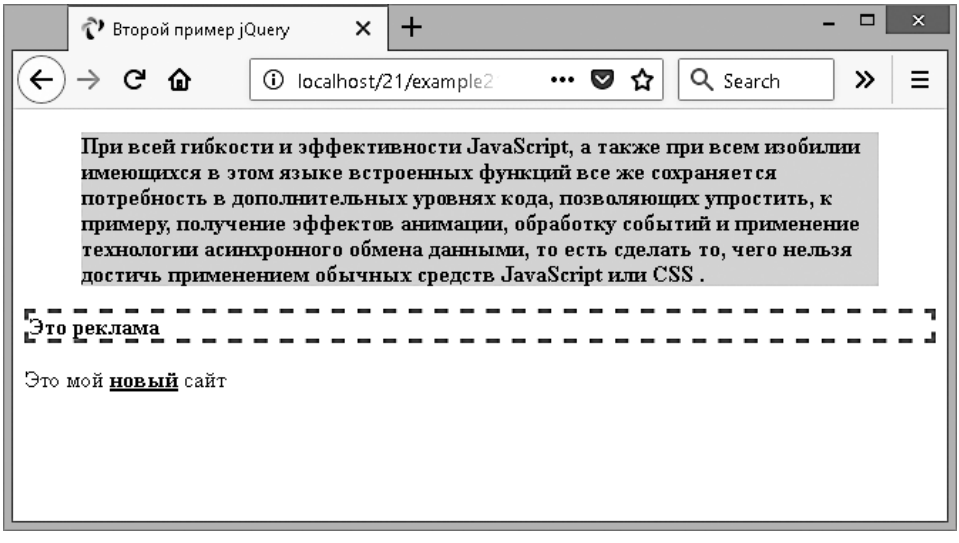

Рис. 21.2. Воздействие сразу на несколько элементов

Обработка событий

Если бы библиотека jQuery умела только подменять CSS-стили, толку от нее было бы маловато, и она, конечно же, способна на гораздо большее. Продолжим исследование и посмотрим, как она обрабатывает события.

Как вы, наверное, помните, большинство событий инициируется действиями пользователя: при прохождении указателя мыши над элементом, щелчке кнопкой мыши или нажатии клавиши. Но существуют и другие события, которые могут инициироваться, к примеру, по завершении загрузки документа.

Прикрепить ваш собственный код к этим событиям с помощью jQuery не составит труда, причем сделано это будет безопасным способом, не блокирующим для другого кода получение такого же доступа к этим событиям. Вот, к примеру, как можно заставить код jQuery откликнуться на щелчок на элементе:

```
$('#clickme').click(function()
{
  $('#result').html('You clicked the button!')
})
```

Когда будет сделан щелчок на элементе с идентификатором `clickme`, свойство `innerHTML` элемента со значением ID, равным `result`, будет обновлено с использованием jQuery-функции `html`.

Объекты jQuery, созданные с помощью метода $ либо метода jQuery, не являются аналогами объектов JavaScript, созданных с помощью getElementById. В обычном коде JavaScript можно использовать такие инструкции, как object = document.getElementById('result'), за которыми, к примеру, следует инструкция object.innerHTML = 'something'. Но в предыдущем примере код $('#result').innerHTML работать не будет, поскольку innerHTML не является свойством объекта jQuery. Следовательно, для достижения требуемого результата нужно использовать jQuery-метод html.

Конкретизация замысла, результат которой можно увидеть на рис. 21.3, показана в примере 21.3.

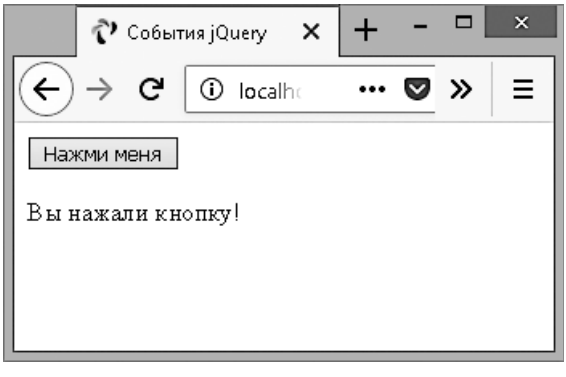

Рис. 21.3. Обработка события click

Пример 21.3. Обработка события

```
<!DOCTYPE html>
<html>
  <head>
    <title>События jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='clickme'>Нажми меня</button>
    <p id='result'>Я - абзац</p>
    <script>
      $('#clickme').click(function()
      {
        $('#result').html('Вы нажали кнопку!')
```

```
      })
    </script>
  </body>
</html>
```

При обращении к событию с помощью jQuery префикс on-, который нужно использовать в стандартном JavaScript, следует опустить. Поэтому, к примеру, название события onmouseover превращается в jQuery в имя функции mouseover, onclick приобретает вид click и т. д.

Ожидание готовности документа

Поскольку тому, что достигается средствами jQuery, мы во многом обязаны весьма тесной связи этой библиотеки с объектной моделью документа — DOM, вам, прежде чем воздействовать на какие-либо части страницы, скорее всего, придется дождаться ее загрузки. Без jQuery это может быть выполнено с помощью события `onload`, но есть более эффективный кросс-браузерный jQuery-метод под названием `ready`, который можно вызвать для включения в работу в самый ранний из возможных моментов времени, даже раньше, чем наступит событие `onload`. Это означает, что jQuery может начать работать на странице намного быстрее и с минимальными задержками для пользователя.

Чтобы воспользоваться этой возможностью, поместите свой код jQuery внутрь следующей структуры:

```
$('document').ready(function()
{
  // Сюда нужно поместить ваш код
})
```

Теперь код будет ждать готовности документа и только после этого будет вызван методом `ready`. На самом деле можно набрать еще меньший объем кода и воспользоваться более краткой версией, показанной в примере 21.4.

Пример 21.4. Наименьший по объему код охватывающей функции, запускаемой по готовности документа (своеобразный аналог метода ready)

```
$(function()
{
  // Сюда нужно поместить ваш код
})
```

Если выработать привычку помещения своих jQuery-инструкций в одну из этих двух структур, то не придется сталкиваться с тем типом ошибок, которые могут выдаваться при попытке слишком раннего обращения к DOM.

Можно использовать альтернативный подход: помещать код JavaScript в конец каждой HTML-страницы, чтобы он выполнялся только после загрузки всего документа. Есть и менее существенное преимущество: поскольку приоритет в загрузке отдается содержимому веб-страницы, у пользователя от работы с такой страницей складывается более благоприятное впечатление.

Единственная ситуация, при которой размещение сценариев в самом конце страницы может вызвать возражение, связана с тем, что документ выглядит как готовый к работе, но фактически он к ней еще не готов, или с тем, что все внешние таблицы стилей еще не загружены (реально определить это можно только тестированием), что вводит пользователей в заблуждение относительно возможности работы с документом до того, как к этому будет готов ваш сценарий. В таких случаях применяйте функцию ready, и все будет в порядке. Если же у вас есть сомнения, поместите свой сценарий в конец страницы и поместите его jQuery-вызовы в функцию ready, и тогда возьмете все самое лучшее от обоих вариантов.

Функции и свойства событий

До сих пор был показан только метод события `ready`, но в jQuery имеется несколько десятков методов событий и связанных с событиями свойств, к которым можно обратиться (их так много, что подробно рассмотреть здесь весь арсенал не представляется возможным). Но рассматриваемые далее функции и свойства относятся к наиболее востребованным и позволят вам начать их использовать в большинстве проектов. Всестороннее описание всех доступных событий можно найти на сайте api.jquery.com/category/events.

События blur и focus

Событие `blur` инициируется, когда фокус убирается с элемента, заставляя этот элемент выглядеть потерявшим фокус, и оно является хорошим партнером для события `focus`. Для добавления обработчика к событию могут использоваться методы `blur` и `focus`. Если же в круглых скобках при вызове метода будут опущены все аргументы, он будут инициировать событие.

В примере 21.5 показаны четыре поля ввода, и первое из них благодаря вызову метода `focus`, применяемого к элементу с идентификатором `first`, сразу же получает фокус. Затем ко всем элементам `input` добавляются два обработчика. Обработчик события `focus` устанавливает для этих элементов желтый фон, когда они получают фокус, а обработчик события `blur` устанавливает для них светло-серый фон, когда фокус с них убирается (у них теряется).

Пример 21.5. Использование событий focus и blur

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>События: blur</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Щелкните в пределах и за пределами этих полей</h2>
    <input id='first'> <input><input> <input>
    <script>
      $('#first').focus()
      $('input').focus(function() { $(this).css('background', '#ff0') } )
      $('input') .blur(function() { $(this).css('background', '#aaa') } )
    </script>
  </body>
</html>
```

Между закрывающей скобкой метода и оператором-точкой, используемым для прикрепления к нему еще одного метода, разрешается включать пробельные символы (и после точки тоже), что я и сделал в предыдущем примере, чтобы выровнять по правому краю имена событий focus и blur, находящиеся друг под другом, чтобы все остальные части инструкций также выстроились в столбец.

На рис. 21.4 показано, как с помощью этого кода любым полям ввода, у которых когда-либо был фокус, придается светло-серый цвет фона. Если у одного из полей в данный момент имеется фокус, цвет его фона становится желтым, а непосещенные поля по-прежнему имеют белый цвет фона.

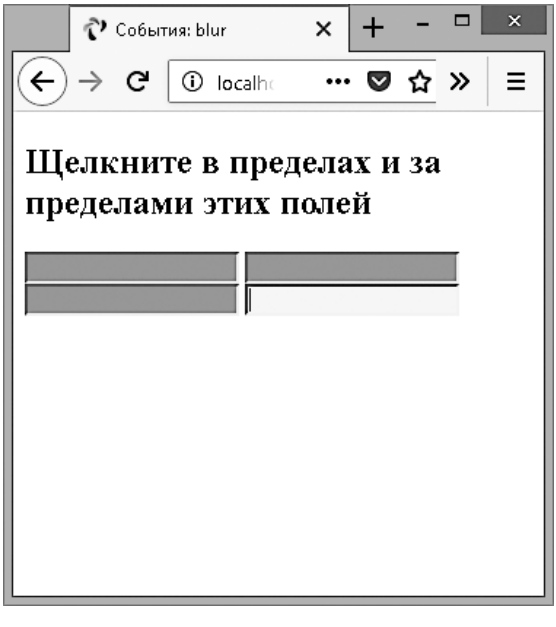

Рис. 21.4. Прикрепление обработчиков к событиям blur и focus

Ключевое слово this

Этот пример также служит иллюстрацией применения ключевого слова `this`. При вызове события объекту `this` передается элемент, в отношении которого это событие было инициировано, и теперь этот объект может быть передан методу `$` для обработки. Или же, поскольку он является стандартным объектом JavaScript (а не объектом jQuery), он может быть использован в качестве такого объекта. Поэтому, если хотите, можете заменить следующий код:

```
$(this).css('background', '#ff0')
```

вот этим кодом:

```
this.style.background = '#ff0'
```

События click и dblclick

Событие `click` ранее уже рассматривалось, но есть также событие, предназначенное для обработки двойных щелчков. Чтобы воспользоваться любым из них, нужно прикрепить метод события к селектору jQuery, а в качестве его аргумента предоставить jQuery-метод, который будет запущен, когда это событие произойдет:

```
$('.myclass')   .click( function() { $(this).slideUp() })
$('.myclass').dblclick( function() { $(this).hide()    })
```

Здесь я решил использовать безымянные функции, но при желании вместо них можно воспользоваться функциями с именами (но не забудьте, что предоставить нужно только имя функции без круглых скобок, в противном случае она будет вызвана несвоевременно). Объекту `this` будет передано то, что и ожидалось, и он станет доступен именованной функции:

```
$('.myclass').click(doslide)

function doslide()
{
  $(this).slideUp()
}
```

Подробное описание методов `slideUp` и `hide` дается в разделе «Специальные эффекты» данной главы. А сейчас просто попробуйте запустить код примера 21.6 и сделайте одинарный либо двойной щелчок на кнопках, чтобы посмотреть, как одни из них исчезают с применением анимации (при использовании `slideUp`), а другие просто исчезают (при использовании `hide`). Результат работы кода показан на рис. 21.5.

Пример 21.6. Прикрепления к событиям click и dblclick

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>События: click и dblclick</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Сделайте на кнопках одинарный и двойной щелчок</h2>
    <button class='myclass'>Кнопка 1</button>
    <button class='myclass'>Кнопка 2</button>
    <button class='myclass'>Кнопка 3</button>
    <button class='myclass'>Кнопка 4</button>
    <button class='myclass'>Кнопка 5</button>
    <script>
      $('.myclass').click( function() { $(this).slideUp() })
      $('.myclass').dblclick( function() { $(this).hide() })
    </script>
  </body>
</html>
```

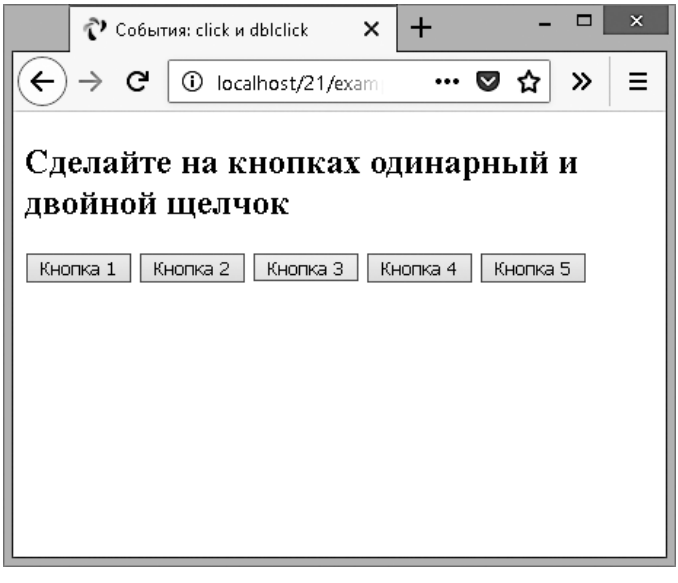

Рис. 21.5. На кнопке 3 был сделан одинарный щелчок, и она ускользнула вверх

Событие keypress

Периодически возникает потребность в более тщательном контроле работы пользователя на клавиатуре, в особенности при обработке сложных форм или написании игр. В таких случаях можно воспользоваться методом `keypress`, который может быть прикреплен к любому элементу, воспринимающему клавиатурный ввод, например к полю ввода или даже самому документу.

В примере 21.7 метод прикреплен к документу, чтобы перехватывать все нажатия клавиш, и результат его запуска показан на рис. 21.6.

Пример 21.7. Перехват нажатия клавиш

```
<!DOCTYPE html>
<html>
  <head>
    <title>События: keypress</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Нажмите какие-нибудь клавиши</h2>
    <div id='result'></div>
    <script>
      $(document).keypress(function(event)
      {
        key = String.fromCharCode(event.which)

        if (key >= 'a' && key <= 'z'  ||
            key >= 'A' && key <= 'Z' ||
            key >= '0' && key <= '9')
        {
          $('#result').html('Вы нажали: ' + key) event.preventDefault()
        }
      })
    </script>
  </body>
</html>
```

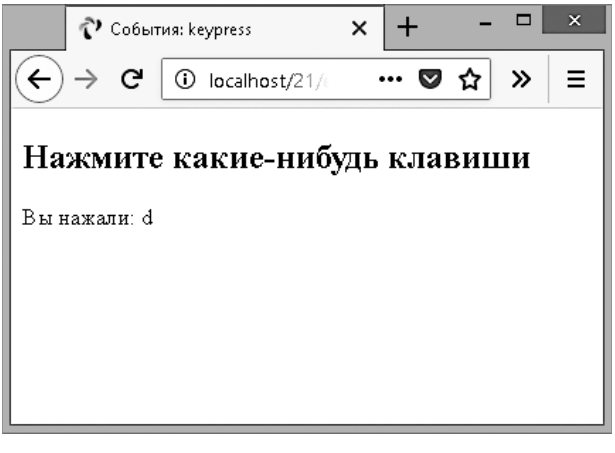

Рис. 21.6. Обработка событий нажатия клавиш

В этом примере следует обратить внимание на ряд особенностей, которые нужно учитывать при написании собственных обработчиков действий на клавиатуре. К примеру, поскольку браузеры возвращают для этого события разные значения, свойство `which` объекта `event` нормализуется библиотекой jQuery, чтобы все браузеры возвращали одни и те же коды символов. Это делается для того, чтобы можно было определить, какая клавиша была нажата.

Но значение в свойстве `which`, будучи кодом символа, является числом, которое можно превратить в отдельную букву, пропустив его через код `String.fromCharCode`. Вам этого делать не нужно, поскольку вы легко можете в своем коде откликаться на ASCII-значения. Но данный метод пригодится, когда нужно будет работать с символами.

Когда в блоке `if` нажатая клавиша будет распознана, код примера вставляет для производства соответствующего эффекта простую инструкцию в свойство `innerHTML` элемента `<div>`, у которого имеется идентификатор (ID) со значением `result`.

Это тот самый пример, в котором не следует использовать функцию document.write, поскольку на момент нажатия клавиши пользователем документ должен быть полностью загружен. Если document.write будет вызван для показа информации до того, как это произойдет, будет стерт весь документ. При таких обстоятельствах, как объяснялось в разделе «О функции document.write» главы 13, лучше ввести запись в HTML-элемент, то есть воспользоваться неразрушающим способом предоставления пользователю обратной связи.

Деликатное программирование

Ожидая пользовательского ввода, нужно решить, на какие значения следует откликаться, после чего игнорировать все остальные значения на тот случай, если к ним должен получить доступ какой-нибудь другой обработчик событий. Это будет примером деликатности, проявляемой по отношению к любой другой полезной программе, которая может находиться в рабочем состоянии (и к самому основному браузеру). Например, в предыдущем примере был выбран прием только символов в диапазонах a–z, A–Z и 0–9, а все остальные символы были проигнорированы.

Есть два способа пропуска прерываний клавиатуры к другим обработчикам (или отказа им в обработке этих прерываний). Во-первых, ничего не делать, тогда при выходе из вашего кода другие обработчики также будут все видеть и смогут реагировать на те же нажатия клавиш. Но это может привести к путанице в том случае, если из-за одного нажатия клавиши произойдет сразу несколько действий.

Альтернативный вариант применяется в том случае, если вам не нужно, чтобы событие инициировало работу других обработчиков, тогда в отношении объекта `event` можно вызвать метод `preventDefault`, который не допустит «всплытия» события на уровень других обработчиков.

Помещая в код вызов метода preventDefault, нужно проявлять особую осмотрительность, поскольку, если этот вызов находится за пределами той части кода, в которой ведется обработка нажатий клавиш, это создаст препятствие для всплытия всех остальных клавиатурных событий и вы можете заблокировать пользователя от использования браузера (или как минимум от использования некоторых его возможностей).

Событие mousemove

Наиболее часто осуществляется перехват событий, связанных с использованием мыши. Щелчки кнопками мыши уже рассматривались, а теперь посмотрим на прикрепление кода к событиям перемещения указателя мыши.

Полагаю, настало время перейти к демонстрации более интересных примеров, и в примере 21.8 я объединил простейшую программу рисования, использующую jQuery, с холстом HTML5. Хотя до главы 24 все особенности холста рассматриваться не будут, волноваться не стоит, поскольку код очень простой.

Пример 21.8. Перехват событий перемещения указателя мыши и нажатия ее кнопок

```
<!DOCTYPE html>
<html>
  <head>
    <title>События: Обработка действий с мышью</title>
    <script src='jquery-3.2.1.min.js'></script>
    <style>
      #pad {
        background:#def;
        border    :1px solid #aaa;
      }
    </style>
  </head>
  <body>
    <canvas id='pad' width='480' height='320'></canvas>
    <script>
      canvas = $('#pad')[0]
      context = canvas.getContext("2d")
      pendown = false

      $('#pad').mousemove(function(event)
      {
        var xpos = event.pageX - canvas.offsetLeft
        var ypos = event.pageY - canvas.offsetTop

        if (pendown) context.lineTo(xpos, ypos)
        else         context.moveTo(xpos, ypos)

        context.stroke()
      })

      $('#pad').mousedown(function() { pendown = true } )
      $('#pad')  .mouseup(function() { pendown = false } )
    </script>
  </body>
</html>
```

На рис. 21.7 показано, как этот очень простой набор инструкций может использоваться для рисования линий (что может пригодиться тем, у кого есть талант

к рисованию). Вот как это работает. Сначала путем ссылки на первый (с нулевым индексом) элемент селектора jQuery программа создает объект `canvas`:

```
canvas = $('#pad')[0]
```

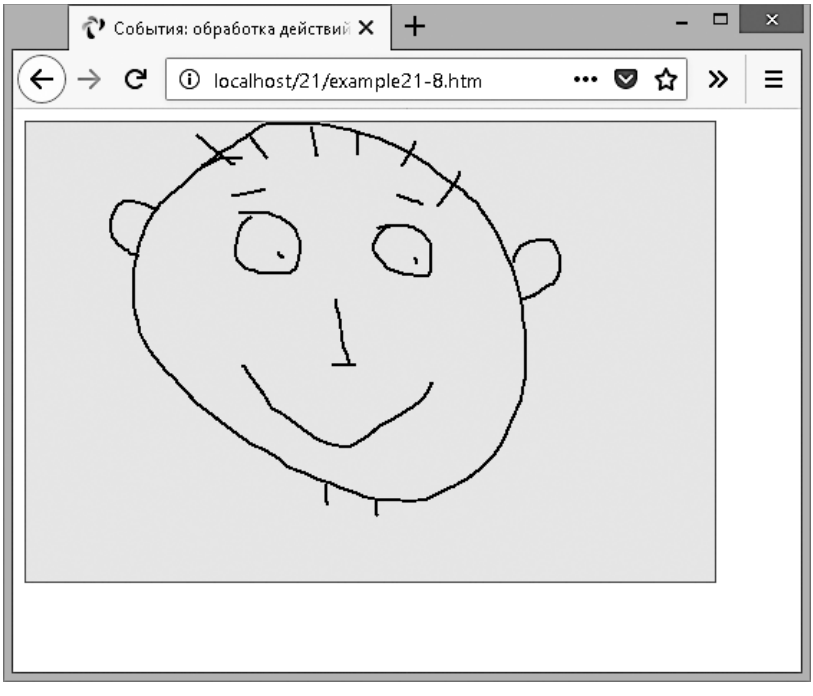

Рис. 21.7. Перехват событий перемещения указателя мыши и нажатия ее кнопок

Это один из способов быстрого получения объекта jQuery и извлечения стандартного объекта элемента JavaScript. Другой способ предусматривает использование метода `get`:

```
canvas = $('#pad').get(0)
```

Оба способа взаимозаменяемы, но при использовании метода `get` есть одно преимущество: если ему не передать аргументы, он вернет все объекты элементов узла из объекта jQuery в виде массива.

В главе 24 будет рассказано, что холст всегда создается для использования специального объекта `context`, который сейчас и будет определен:

```
context = canvas.getContext("2d")
```

Нужно инициализировать еще кое-что, создав булеву переменную под названием `pendown` (перо опущено), которая будет использоваться для отслеживания

состояния кнопки мыши (с исходным значением `false`, поскольку перо пока что поднято):

```
pendown = false
```

После этого холст (с идентификатором `pad`) получает свое событие `mousemove`, перехватываемое показанной далее безымянной функцией, благодаря которой происходят три набора действий:

```
$('#pad').mousemove(function(event)
{
  ...
})
```

Сначала локальным переменным `xpos` и `ypos` (они являются локальными благодаря применению ключевых слов `var`) присваиваются значения, представляющие собой позицию указателя мыши в области холста.

Эти значения берутся из свойств jQuery `pageX` и `pageY`, которые ссылаются на смещение указателя мыши от верхнего левого угла соответствующего документа. Поэтому, так как сам холст немного смещен с этой позиции, значения смещения холста `canvas` (в свойствах `offsetLeft` и `offsetTop`) вычитаются соответственно из `pageX` и `pageY`:

```
var xpos = event.pageX - canvas.offsetLeft
var ypos = event.pageY - canvas.offsetTop
```

Теперь, когда нам известно, где находится указатель мыши по отношению к холсту, в следующих двух строках кода тестируется значение переменной `pendown`. Если оно равно `true`, значит была нажата кнопка мыши, и поэтому вызывается метод `lineTo` для рисования линии в текущей позиции. В противном случае перо поднято, и поэтому вызывается метод `moveTo`, для того чтобы просто обновить значения текущей позиции:

```
if (pendown) context.lineTo(xpos, ypos)
else         context.moveTo(xpos, ypos)
```

Затем вызывается метод `stroke` для применения той команды рисования, которая только что была вызвана по отношению к холсту. Эти пять строк — все, что нужно для управления рисованием, но необходимо по-прежнему отслеживать состояние кнопки мыши, и поэтому завершающие две строки кода перехватывают события `mousedown` и `mouseup`, устанавливая для `pendown` значение `true` при нажатии кнопки мыши и `false` при ее освобождении:

```
$('#pad').mousedown(function() { pendown = true  } )
$('#pad')  .mouseup(function() { pendown = false } )
```

В этом примере показано сочетание работающих вместе трех разных обработчиков событий для создания полезной программы, использующей как локальные переменные для внутренних выражений, так и глобальные переменные, где объект или состояние чего-либо должны быть сделаны доступными нескольким функциям.

Другие события, связанные с мышью

События `mouseenter` и `mouseleave` инициируются при прохождении указателя мыши над элементом или при выходе его за границы элемента. Позиционные значения не предоставляются, поскольку предполагается, что вам просто требуется принять логическое решение о том, что делать при выдаче одного из этих событий.

В примере 21.9 к этим событиям прикреплены две безымянные функции, которые, как показано на рис. 21.8, изменяют соответствующим образом HTML-код элемента.

Пример 21.9. Определения входа указателя в границы элемента и выхода за их пределы

```
<!DOCTYPE html>
<html>
  <head>
    <title>События: Дальнейшая обработка событий мыши</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2 id='test'>Проведи надо мной указателем мыши</h2>
    <script>
      $('#test').mouseenter(function() { $(this).html('Эй, прекрати
        щекотать!') })
      $('#test').mouseleave(function() { $(this).html('Куда же ты
        подевался?') } )
    </script>
  </body>
</html>
```

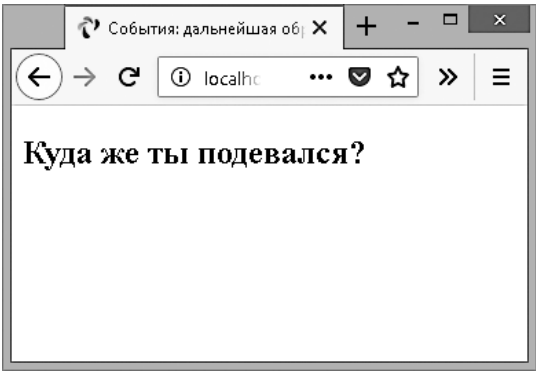

Рис. 21.8. Определение момента входа указателя мыши в границы элемента и выхода за их пределы

Когда указатель мыши входит в границы выбранного элемента, обновляется свойство элемента `innerHTML` (посредством вызова метода `html`). Затем, когда указатель мыши снова оказывается за границами элемента, происходит еще одно обновление HTML-кода элемента.

Альтернативные методы работы с мышью

В jQuery доступны и другие функции, связанные с событиями мыши и охватывающие широкий диапазон возможных обстоятельств. Подробное описание всех этих функций можно найти по адресу api.jquery.com/category/events/mouse-events.

К примеру, для получения таких же результатов, как и при выполнении кода из предыдущего раздела, можно воспользоваться следующими альтернативными методами `mouseover` и `mouseout`:

```
$('#test').mouseover(function() { $(this).html('Cut it out!')         } )
$('#test') .mouseout(function() { $(this).html('Try it this time...') } )
```

Или же, чтобы связать два обработчика с помощью одной функции, можно воспользоваться методом `hover`:

```
$('#test').hover(function() { $(this).html('Cut it out!')         } )
                 function() { $(this).html('Try it this time...') } )
```

Если планируется получение совокупного эффекта от применения `mouseover` и `mouseout`, то вполне логично будет воспользоваться методом `hover`, но есть еще один способ, позволяющий получить такой же результат, который называется *выстраиванием цепочки* (и объясняется чуть позже в пункте «Выстраивание цепочки методов»):

```
$('#test').mouseover(function() { $(this).html('Cut it out!')         } )
           .mouseout(function() { $(this).html('Try it this time...') } )
```

Здесь оператор-точка в начале второй инструкции прикрепляет ее к первой инструкции, создавая тем самым цепочку методов.

В предыдущих примерах показан способ перехвата щелчка кнопкой мыши, перемещения указателя мыши и события клавиатуры, в силу чего они больше всего подходят для сред настольных компьютеров, на которые в первую очередь и нацелено применение библиотеки jQuery. Но существует также версия jQuery для мобильных устройств под названием jQuery Mobile (http://jquerymobile.com/), обеспечивающая управление обработкой всех событий прикосновений, которые вам только могут потребоваться (и многое другое), и доступная по адресу jquerymobile.com.

Событие submit

При отправке формы может понадобиться выполнение различных проверок на наличие ошибок во введенных данных перед отправкой их на сервер. Как показано в примере 21.10, одним из способов получения такой возможности является перехват события `submit`, происходящего в форме. На рис. 21.9 представлен результат

загрузки документа с последующей отправкой формы с одним или двумя незаполненными полями.

Пример 21.10. Перехват события submit, происходящего в форме

```
<!DOCTYPE html>
<html>
  <head>
    <title>События: submit</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <form id='form'>
      Имя:     <input id='fname' type='text' name='fname'><br>
      Фамилия: <input id='lname' type='text' name='lname'><br>
      <input type='submit'>
    </form>
    <script>
    $('#form').submit(function()
    {
      if ($('#fname').val() == '' || $('#lname').val() == '')
      {
        alert('Пожалуйста, введите имя и фамилию')
        return false
      }
    })
    </script>
  </body>
</html>
```

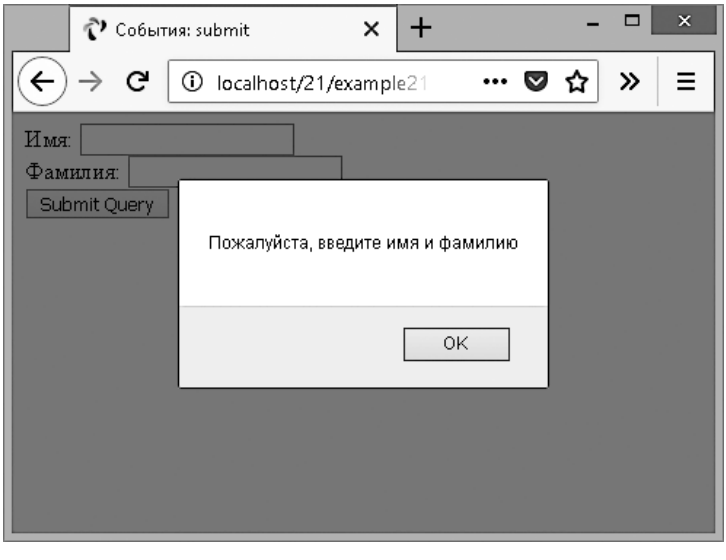

Рис. 21.9. Проверка пользовательского ввода после отправки

Основной частью этого примера является код, в котором к событию прикрепляется безымянная функция:

```
$('#form').submit(function()
```

и где значения двух полей ввода проверяются на их заполнение:

```
if ($('#fname').val() == '' ||
    $('#lname').val() == '')
```

Здесь для получения значения свойства `value` каждого поля используется jQuery-метод `val`. Этот прием выглядит лучше, чем использование `$('#fname')[0]` (как в примере 21.8) для получения доступа к DOM-объекту с последующим добавлением к нему свойства `value` для чтения значения поля:

```
$('#fname')[0].value.
```

В данном примере при возвращении значения `false` в случае незаполненности одного или нескольких полей обычный процесс отправки прекращается. Чтобы разрешить отправку, нужно, чтобы было возвращено значение `true`, или вообще не возвращать никакого значения.

Специальные эффекты

В чем действительно преуспела библиотека jQuery, так это в создании спецэффектов. Можно, конечно, воспользоваться переходами CSS3, но их динамическое управление из JavaScript не будет настолько же простым, а с использованием jQuery все сведется к простому выбору одного или нескольких элементов с последующим применением к ним одного или нескольких эффектов.

Основными доступными эффектами являются исчезновение и появление, постепенное проявление и растворение, скольжение, а также анимация, которые могут использоваться по одному, все вместе согласованно по времени или друг за другом. Поддерживаются также обратные вызовы, представляющие собой функции или методы, вызываемые только один раз по завершении операции.

В следующем разделе дается описание ряда наиболее полезных jQuery-эффектов, каждым из которых поддерживаются три аргумента.

- *Продолжительность (Duration)*. Когда предоставляется это значение, эффект будет наблюдаться в течение назначенного времени, которое может быть задано в миллисекундах или же строками `fast` (быстро) или `slow` (медленно).
- *Изменение скорости выполнения эффекта (Easing)*. В библиотеке jQuery только два варианта изменения скорости: `swing` (с ускорением) и `linear` (линейное изменение). По умолчанию используется вариант `swing`, который задает более естественное изменение скорости эффекта, чем `linear`. Дополнительные варианты изменения скорости выполнения эффекта можно найти в таких до-

полнительных модулях, как jQuery UI, который можно увидеть на сайте http://jqueryui.com/easing/.

- *Функция обратного вызова (Callback)*. Если предоставлена функция обратного вызова, она будет вызвана сразу же после завершения работы метода создания эффекта.

Это означает, что в случае отсутствия предоставляемых аргументов метод вызывается немедленно и в очередь анимации не попадает.

К примеру, метод `hide` можно вызвать несколькими способами:

```
$('#object').hide()
$('#object').hide(1000)
$('#object').hide('fast')
$('#object').hide('linear')
$('#object').hide('slow', 'linear')
$('#object').hide(myfunction)
$('#object').hide(333, myfunction)
$('#object').hide(200, 'linear', function() { alert('Finished!') } )
```

Как будет показано в пункте «Выстраивание цепочки методов», можно прикрепить вызовы функций (с аргументами) друг к другу, а затем они будут анимированы в порядке очереди, как в следующем примере, где элемент сначала исчезнет, а затем появится снова:

```
$('#object').hide(1000).show(1000)
```

Многими этими методами поддерживаются и другие, менее востребованные аргументы, подробное описание которых (а также других поддерживаемых методов создания эффектов) можно найти по адресу http://api.jquery.com/category/effects.

Исчезновение и появление

Наверное, простейшим эффектом можно считать исчезновение и появление элемента в ответ на действия пользователя. В предыдущем разделе говорилось, что методам `hide` и `show` можно вообще не предоставлять никаких аргументов или же предоставлять различные аргументы, а по умолчанию, когда им ничего не предоставлено, результатом станет мгновенное исчезновение или появление элемента.

Когда аргументы предоставляются, эти два метода одновременно изменяют свойства элемента `width`, `height` и `opacity` до тех пор, пока их значения не достигнут нуля при использовании метода `hide` или исходных установок — при использовании метода `show`. После полного исчезновения элемента метод `hide` присваивает его свойству `display` значение `none`, а метод `show` после полного появления элемента снова присваивает этому свойству ранее назначенное ему значение.

Испытать работу методов `hide` и `show` позволит код примера 21.11, а результат можно увидеть на рис. 21.10.

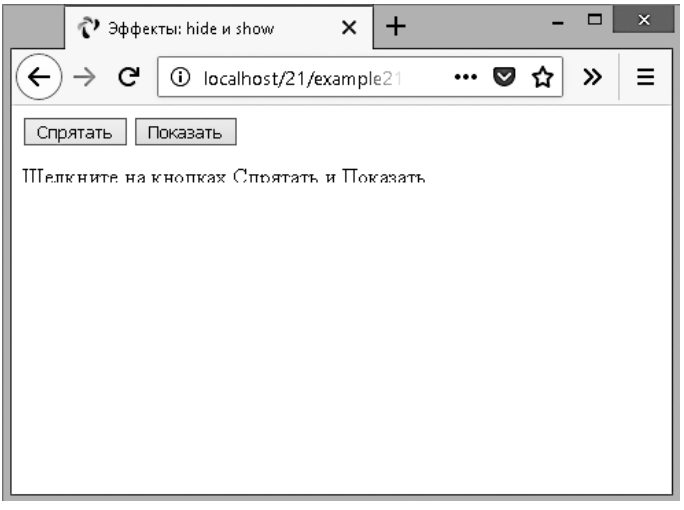

Рис. 21.10. Элемент в процессе появления

Пример 21.11. Исчезновение и появление элемента

```
<!DOCTYPE html>
<html>
  <head>
    <title>Эффекты: hide и show</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='hide'>Спрятать</button>
    <button id='show'>Показать</button>
    <p id='text'>Щелкните на кнопках Спрятать и Показать</p>
    <script>
      $('#hide').click(function() { $('#text').hide('slow', 'linear') })
      $('#show').click(function() { $('#text').show('slow', 'linear') })
    </script>
  </body>
</html>
```

Метод toggle

Альтернативой вызову обоих методов, как `hide`, так и `show`, может стать вызов метода `toggle`, который позволяет заменить предыдущий пример кодом из примера 21.12.

Пример 21.12. Использование метода toggle

```
<!DOCTYPE html>
<html>
  <head>
    <title>Эффекты: toggle</title>
```

```
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='toggle'>Изменить состояние</button>
    <p id='text'>Нажмите кнопку Изменить состояние</p>
    <script>
      $('#toggle').click(function() { $('#text').toggle('slow', 'linear') })
    </script>
  </body>
</html>
```

Методу `toggle` передаются точно такие же аргументы, что и методам `hide` и `show`, но он отслеживает внутреннее состояние элемента, зная таким образом, что нужно делать, заставлять элемент исчезнуть или появиться.

В jQuery имеются четыре основных метода, устанавливающих либо одно, либо другое состояние и предлагающих для упрощения программирования версии переключения. Кроме toggle имеются методы fadeToggle, slideToggle и toggleClass, которые будут рассмотрены в данной главе.

Проявление и растворение

Проявлением и растворением управляют четыре метода: `fadeIn`, `fadeOut`, `fadeToggle` и `fadeTo`. Теперь вы уже имеете представление о том, как работает jQuery, и сможете разобраться в том, что первые три метода похожи, соответственно, на методы `show`, `hide` и `toggle`.

Но последний метод имеет некоторые отличия: ему можно указывать значение непрозрачности (`opacity`) от `0` до `1`, до которого элемент (или элементы) должен проявиться.

В примере 21.13 представлены четыре кнопки, позволяющие проверить в действии каждый из этих методов. Результат показан на рис. 21.11.

Пример 21.13. Четыре метода проявления и растворения

```
<!DOCTYPE html>
<html>
  <head>
    <title>Эффекты: Растворение и проявление</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='fadeout'>Растворить</button>
    <button id='fadein'>Проявить</button>
    <button id='fadetoggle'>Переключить состояние</button>
    <button id='fadeto'>Растворить на 50%</button>
    <p id='text'>Щелкните на кнопках, расположенных вверху</p>
    <script>
```

```
      $('#fadeout')   .click(function() { $('#text').fadeOut(   'slow' )   })
      $('#fadein')    .click(function() { $('#text').fadeIn(    'slow' )   })
      $('#fadetoggle').click(function() { $('#text').fadeToggle('slow' )   })
      $('#fadeto')    .click(function() { $('#text').fadeTo(    'slow', 0.5) })
    </script>
  </body>
</html>
```

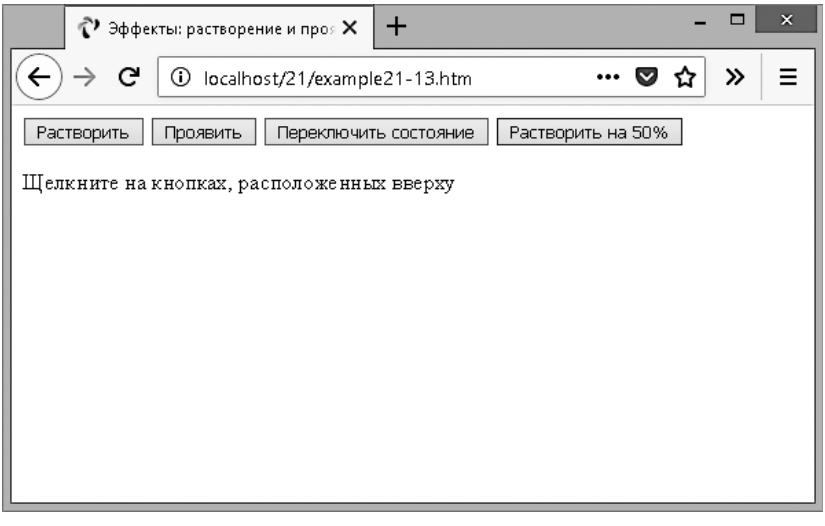

Рис. 21.11. Текст проявился до 50%-ной непрозрачности

Скольжение элементов вверх и вниз

Еще один способ, заставляющий элементы исчезать и появляться снова, заключается в постепенном изменении их высоты с целью имитации ускользания элементов за границу и выскальзывания из-под этой границы. Для достижения этих эффектов доступны три метода jQuery: `slideDown`, `slideUp` и `slideToggle`. Они работают по тем же принципам, что и предыдущие, в чем можно убедиться, запустив на выполнение код примера 21.14. А результат можно увидеть на рис. 21.12.

Пример 21.14. Использование методов ускользания и выскальзывания

```
<!DOCTYPE html>
<html>
  <head>
    <title>Эффекты: Скольжение</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='slideup'>Скольжение вверх</button>
    <button id='slidedown'>Скольжение вниз</button>
    <button id='slidetoggle'>Переключение состояния</button>
```

```
    <div id='para' style='background:#def'>
      <h2>From A Tale of Two Cities - By Charles Dickens</h2>
      <p>It was the best of times, it was the worst of times, it was the age
      of wisdom, it was the age of foolishness, it was the epoch of belief,
      it was the epoch of incredulity, it was the season of Light, it was
      the season of Darkness, it was the spring of hope, it was the winter
      of despair, we had everything before us, we had nothing before us, we
      were all going direct to Heaven, we were all going direct the other
      way - in short, the period was so far like the present period, that
      some of its noisiest authorities insisted on its being received, for
      good or for evil, in the superlative degree of comparison only</p>
    </div>
    <script>
      $('#slideup')    .click(function() { $('#para').slideUp(    'slow') })
      $('#slidedown')  .click(function() { $('#para').slideDown(  'slow') })
      $('#slidetoggle').click(function() { $('#para').slideToggle('slow') })
    </script>
  </body>
</html>
```

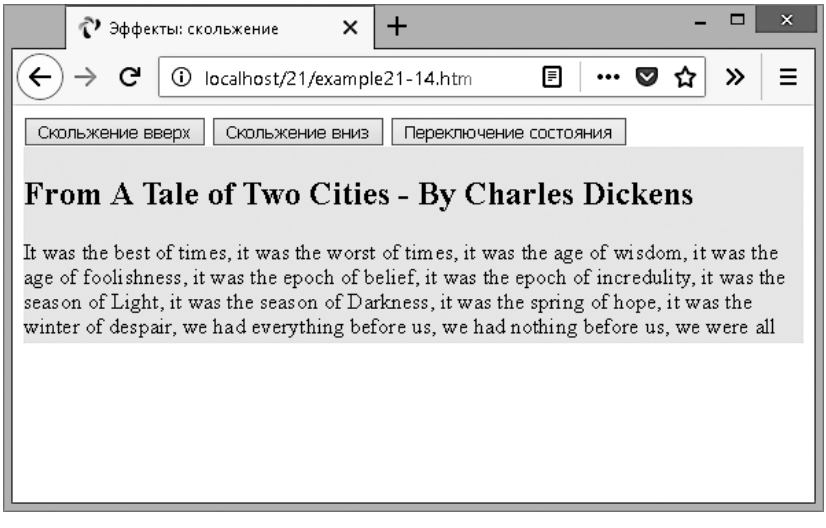

Рис. 21.12. Выдвижение абзаца

Эти методы хорошо подходят для работы с меню и подменю, пункты которых нужно динамически открывать и закрывать в соответствии с тем разделом, на котором пользователь сделал щелчок.

Анимация

А теперь мы можем приступить к весьма забавному занятию — практическому перемещению элементов по окну браузера. Но для этого, поскольку исходное значение свойства `static` не даст нам их перемещать, следует не забыть сначала задать

значения свойствам элементов `position` (позиционирование), исходя из вариантов `relative` (относительное), `fixed` (фиксированное) или `absolute` (абсолютное).

Чтобы применить к элементу эффект анимации, нужно всего лишь предоставить методу `animate` перечень свойств CSS (исключая цвета). В отличие от ранее рассмотренных методов создания эффектов, анимация требует предварительного предоставления перечня свойств, после чего можно предоставить любые аргументы продолжительности, изменения скорости выполнения и обратной функции.

Например, для анимации отскакивающего мячика можно воспользоваться кодом из примера 21.15, результат работы которого показан на рис. 21.13.

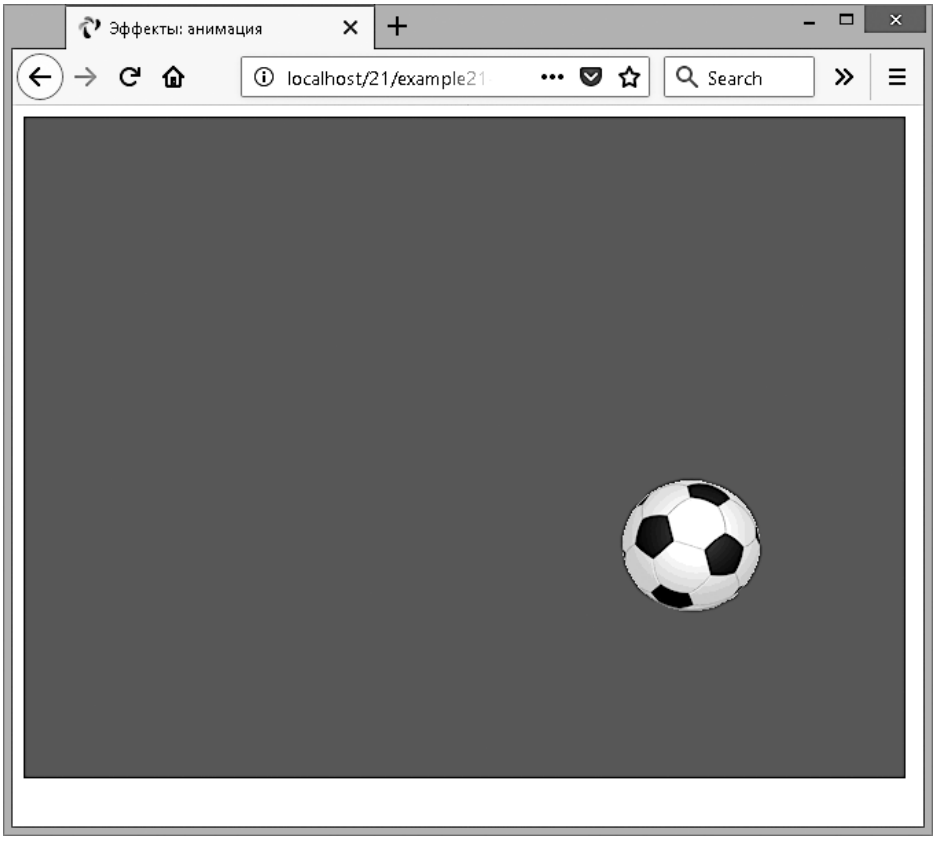

Рис. 21.13. Мячик, отскакивающий от границ окна браузера

Пример 21.15. Создание анимации отскакивающего мячика

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>Эффекты: Анимация</title>
    <script src='jquery-3.2.1.min.js'></script>
    <style>
      #ball {
        position  :relative;
      }
      #box {
        width     :640px;
        height    :480px;
        background:green;
        border    :1px solid #444;
      }
    </style>
  </head>
  <body>
    <div id='box'>
      <img id='ball' src='ball.png'>
    </div>
    <script>
      bounce()

      function bounce()
      {
      $('#ball')
        .animate( { left:'270px', top :'380px' }, 'slow', 'linear')
        .animate( { left:'540px', top :'190px' }, 'slow', 'linear')
        .animate( { left:'270px', top :'0px' },   'slow', 'linear')
        .animate( { left:'0px',   top :'190px' }, 'slow', 'linear')
      }
    </script>
  </body>
</html>
```

В `<style>`-разделе этого примера для свойства `position` мячика установлено значение, позволяющее задавать позиции относительно контейнера, в котором он находится. В роли контейнера выступает `<div>`-элемент, которому заданы граница и зеленый фон.

Затем в `<script>`-раздел помещена функция под названием `bounce`, объединяющая четыре вызова метода `animate`.

Обратите внимание на то, что имена свойств, подвергаемых эффектам анимации (в данном примере это `left` и `top`), предоставляются без кавычек и отделяются от значений, до которых они должны быть изменены (например, `'270px'`), двоеточиями, иными словами, они записываются в форме ассоциативных массивов.

Вместо абсолютных значений можно также задавать относительные, используя для этого операторы `+=` и `-=`. Так, к примеру, следующий код позволит применить к мячику эффект анимации, заключающийся в его перемещении вправо и вверх на 50 пикселов относительно текущей позиции:

```
.animate( { left:'+=50px', top:'-=50px' }, 'slow', 'linear')
```

Для обновления значения свойства можно даже применять строковые значения `hide`, `show` и `toggle`:

```
.animate( { height:'hide', width:'toggle' }, 'slow', 'linear')
```

Если нужно изменить значение каких-либо CSS-свойств, имена которых пишутся через дефис и не передаются в кавычках (как height и width в предыдущем примере), сначала эти имена следует преобразовать в формат слитного написания без дефиса, когда вторая часть имени начинается с прописной буквы (так называемый формат горбатого верблюда — camelCase). Например, для применения эффекта анимации к свойству элемента left-margin следует предоставить имя leftMargin. Но если предоставлять имя с дефисом внутри строки (например: css('font-weight', 'bold'), преобразовывать его в camelCase не нужно.

Выстраивание цепочки методов

Благодаря выстраиванию методов в цепочку при передаче этим jQuery-методам аргументов происходит последовательный запуск методов. Таким образом, каждый из этих методов вызывается только после того, как закончит выполняться эффект анимации предыдущего метода. Но любые методы, вызываемые без аргументов, будут запускаться сразу же, без промедлений и без эффекта анимации.

После загрузки кода примера 21.15 в браузер эффект анимации стартует с однократного вызова функции `bounce`, вызывающей отскок мячика от нижней, правой и верхней границ его контейнера. Затем мячик возвращается в середину левой границы. Если еще раз посмотреть на имеющуюся в этом примере функцию `bounce`, можно увидеть в ней четыре выстроенных в цепочку вызова функции `animate`.

Использование обратного вызова функции

В его нынешнем виде предыдущий пример завершает свою работу после выполнения четырех эффектов анимации, но для многократного запуска эффекта анимации после его завершения можно воспользоваться функцией обратного вызова. Поэтому я решил поместить анимацию в именованную функцию.

При использовании анимации в функции `bounce` остается только лишь указать ее имя, выделенное в примере полужирным шрифтом, в качестве имени функции обратного вызова для четырех анимаций в группе, чтобы заставить эффект анимации повторяться бесконечно:

```
.animate( { left:'0px', top :'190px' }, 'slow', 'linear', bounce)
```

Используя метод `animate`, можно получить эффект анимации многих CSS-свойств с существенным исключением в отношении цветовых решений. Но с применением

дополнительного модуля jQuery UI, который добавляет способность создания очень привлекательных эффектов цветовых изменений (наряду со многими другими интересными эффектами), возможны даже эффекты анимации цвета. Подробности можно найти по адресу http://jqueryui.com.

Остановка анимации

Для остановки еще не завершенной анимации или завершения выполнения цепочки анимации используются несколько способов. Например, с помощью метода `clearQueue` можно очистить все сохраненные в очереди эффекты анимации, с помощью метода `stop` — моментально остановить любую выполняемую в данный момент анимацию, а с помощью метода `finish` — остановить текущую запущенную анимацию и удалить всю анимацию, выстроенную в очередь.

Превратим предыдущий пример в своеобразную игру, предусмотрев возможность обработки щелчка на мяче, чтобы при выдаче события такого щелчка анимация прекращалась. Для этого нужно под функцией `bounce` добавить следующую строку кода:

```
$('#ball').click(function() { $(this).finish() })
```

Если вам удастся щелкнуть на мяче, метод `finish` остановит текущую анимацию, очистит очередь и заставит проигнорировать любые функции обратного вызова, то есть мячик застынет на месте.

Дополнительные сведения об управлении очередями jQuery можно найти по адресу http://api.jquery.com/queue, где также можно будет узнать, как управлять содержимым очередей напрямую для получения именно тех эффектов, которые вам нужны.

Работа с DOM

Поскольку библиотека jQuery слишком тесно привязана к DOM, то в силу необходимости в примерах данной главы уже использовались некоторые имеющиеся в ней методы доступа к DOM-объектам, например `html` и `val`. А теперь подробно рассмотрим все DOM-методы, чтобы выяснить, к чему именно можно получить доступ с помощью jQuery и как это сделать.

В примере 21.3 было показано использование метода `html` для изменения принадлежащего элементу свойства `innerHTML`. Этот метод может использоваться либо для установки кода в HTML-документ, либо для извлечения этого кода из документа. В примере 21.16 (в котором код jQuery выделен полужирным шрифтом) показан способ извлечения HTML-содержимого из элемента. Результат выполнения кода примера показан на рис. 21.14.

Пример 21.16. Вывод в окне оповещения HTML-содержимого элемента

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM: html & text</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Пример документа</h2>
    <p id='intro'>Это пример документа</p>
    <script>
      alert($('#intro').html())
    </script>
  </body>
</html>
```

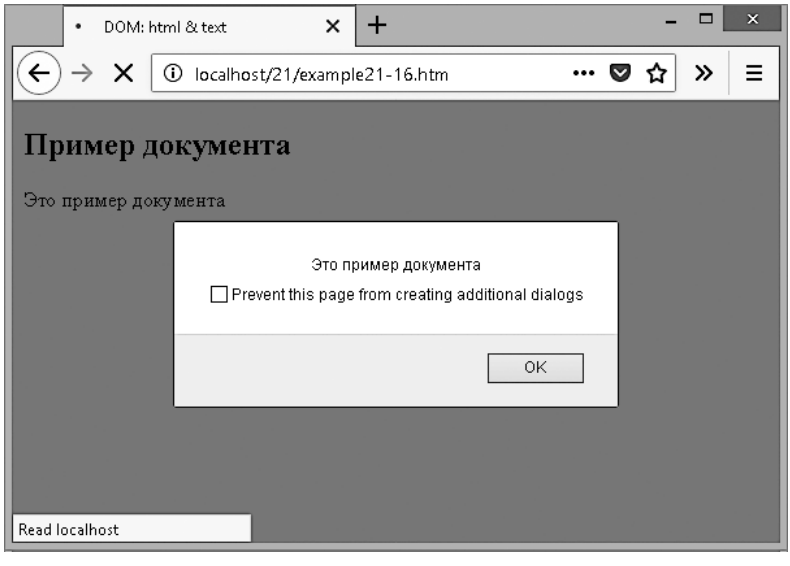

Рис. 21.14. Извлечение и отображение HTML-содержимого элемента

Если при вызове этого метода не указывать никаких аргументов, результатом станет считывание, а не установка HTML-содержимого элемента.

Разница между методами text и html

При работе с XML-документами метод `html` использовать нельзя, поскольку он просто не будет работать (он разработан исключительно для использования с HTML). Но для получения аналогичных результатов (в XML- или HTML-документах) можно воспользоваться методом `text`:

```
text = $('#intro').text()
```

Разница между методами заключается просто в том, что `html` принимает содержимое за HTML, а `text` принимает его за текст. К примеру, предположим, что вам нужно присвоить элементу следующую строку:

```
<a href='http://google.com'>Посетите Google</a>
```

Если присваивать ее HTML-элементу с помощью метода `html`, DOM-модель будет обновлена с получением нового `<a>`-элемента и ссылка станет реагировать на щелчки. Но если сделать то же самое в отношении XML- или HTML-документа с помощью метода `text`, то сначала эта строка будет нейтрализована с превращением кода в текст (например, путем превращения таких HTML-символов, как `<`, в комбинацию символов `<` и т. д.), а затем уже вставлена в элемент, то есть к DOM-модели элементы добавляться не будут.

Методы val и attr

Есть еще два метода для работы с содержимым элементов. Во-первых, как показано в примере 21.10, в котором считывались значения полей имени и фамилии, с помощью метода `val` можно устанавливать и получать значение элемента ввода. Для установки значения нужно просто предоставить его методу в качестве аргумента:

```
$('#password').val('mypass123')
```

С помощью метода `attr`, как показано в примере 21.17, в котором ссылка на сайт Google полностью заменяется ссылкой на сайт Yahoo!, можно получить и установить атрибуты элементов.

Пример 21.17. Изменение атрибутов с помощью метода attr

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM: attr</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Пример документа</h2>
    <p><a id='link' href='http://google.com'
      title='Google'>Посетите Google</a></p>
    <script>
      $('#link').text('Посетите Yahoo!')
      $('#link').attr( { href :'http://yahoo.com', title:'Yahoo!' } )
      alert('Новый код HTML:\n' + $('p').html())
    </script>
  </body>
</html>
```

В первой jQuery-инструкции используется метод `text`, позволяющий изменить текст внутри элемента `<a>`, а вторая инструкция соответствующим образом путем предоставления данных в форме ассоциативного массива изменяет значения

атрибутов `href` и `title`. Третья инструкция с помощью метода `alert` открывает окно оповещения, в которое выводит содержимое измененного элемента, предварительно извлеченное с помощью метода `html` (рис. 21.15).

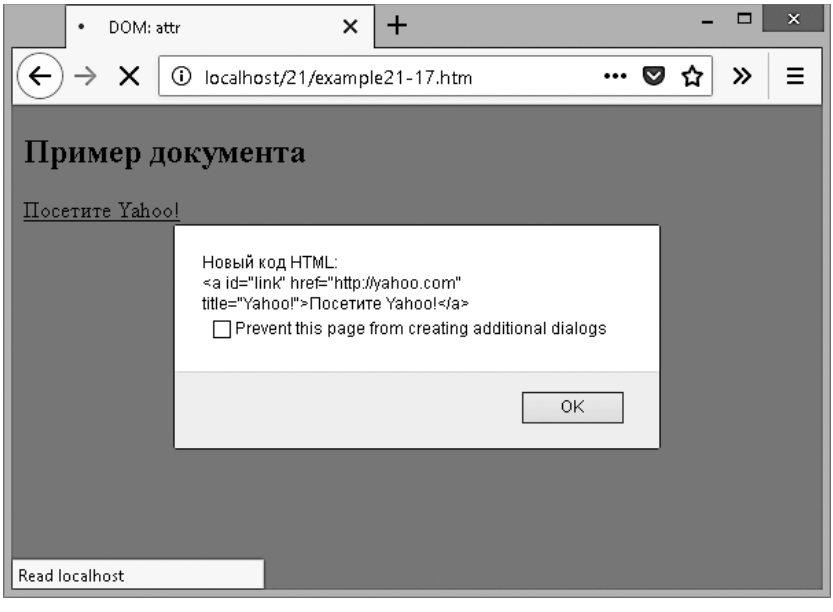

Рис. 21.15. Теперь ссылка полностью изменена

Можно также считать значение атрибута:

```
url = $('#link').attr('href')
```

Добавление и удаление элементов

Метод `html`, конечно, позволяет вставлять элементы в DOM, но он подходит только для создания дочерних элементов отдельно взятого элемента. Поэтому в jQuery предоставляется ряд методов для работы с любой частью DOM.

К таким методам относятся `append`, `prepend`, `after`, `before`, `remove` и `empty`, по одному из вариантов использования которых включает в себя пример 21.18.

Пример 21.18. Добавление и удаление элементов

```
<!DOCTYPE html>
<html>
  <head>
    <title>Изменение DOM</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
```

```
    <h2>Пример документа </h2>
    <a href='http://google.com' title='Google'>Посетите Google</a>
    <code>
      Это раздел кода
    </code>
    <p>
      <button id='a'>Удалить изображение</button>
      <button id='b'>Очистить цитату</button>
    </p>
    <img id='ball' src='ball.png'>
    <blockquote id='quote' style='border:1px dotted #444; height:20px;'>
      тест
    </blockquote>
    <script>
      $('a').prepend('Ссылка: ')
      $("[href^='http']").append(" <img src='link.png'>")
      $('code').before('<hr>').after('<hr>')
      $('#a').click(function() { $('#ball').remove() } )
      $('#b').click(function() { $('#quote').empty() } )
    </script>
  </body>
</html>
```

На рис. 21.16 показан результат применения методов `prepend`, `append`, `before` и `after` к некоторым элементам.

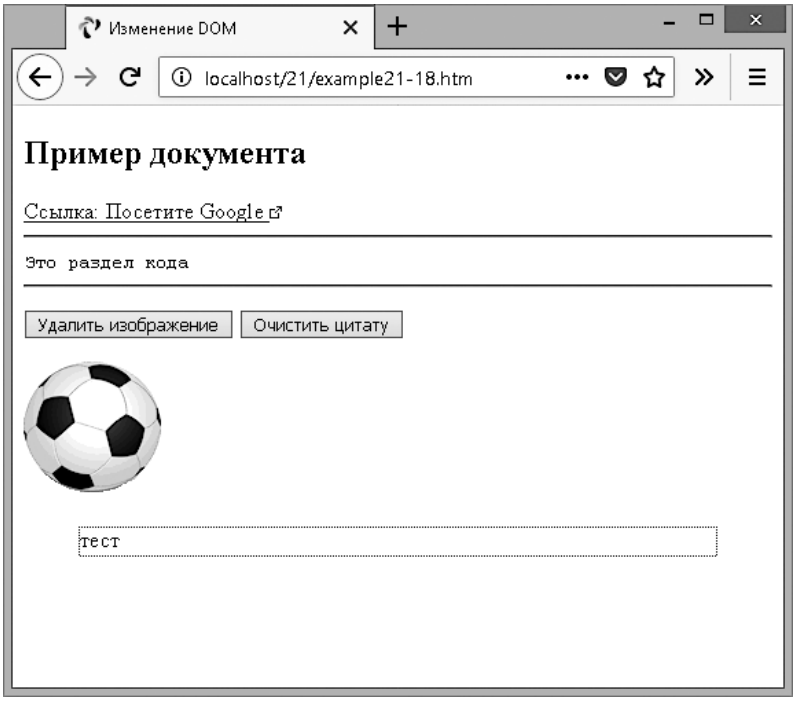

Рис. 21.16. Документ с разнообразными элементами

Метод prepend был использован для вставки строки Link: перед внутренним текстом или HTML-кодом всех <a>-элементов:

```
$('a').prepend('Link: ')
```

Затем для выбора всех элементов, имеющих атрибут href, начинающийся с http, был использован селектор атрибутов. Строка http, появляющаяся в начале URL (поскольку используется оператор ^=), обозначает ссылки, не являющиеся относительными, то есть относящиеся к абсолютным. В данном случае к концу внутреннего текста или HTML-кода всех соответствующих элементов добавляется значок внешней ссылки:

```
$("[href^='http']").append(" <img src='link.png'>")
```

Оператор ^= задает соответствие лишь с началом строки. Если бы использовался только один оператор =, выбирались бы только целые строки с соответствующим значением. Более подробно селекторы CSS рассматриваются в главах 18 и 19.

Далее для помещения одноуровневых элементов (имеющих общего родителя) либо перед заданным элементом, либо после него используются выстроенные в цепочку методы before и after. В данном случае мой выбор пал на помещение элемента <hr> как до, так и после элементов <code>:

```
$('code').before('<hr>').after('<hr>')
```

Затем к паре кнопок я добавил небольшую реакцию на действия пользователя. При щелчке на первой кнопке с помощью метода remove удаляется элемент <img>, в котором содержится изображение мячика:

```
$('#a').click(function() { $('#ball').remove() } )
```

Теперь изображения в DOM больше нет, в чем можно убедиться, если выделить содержимое окна браузера, щелкнуть на нем правой кнопкой мыши и выбрать в большинстве основных браузеров пункт контекстного меню Просмотр кода элемента (Inspect Element) или, при работе с Internet Explorer, нажать клавишу F12.

И наконец, при нажатии второй кнопки к элементу <blockquote> применяется метод empty, который просто-напросто опустошает содержимое элемента в DOM:

```
$('#b').click(function() { $('#quote').empty() } )
```

Динамическое применение классов

Иногда было бы неплохо изменить класс, применяемый к элементу, или, может быть, просто добавить класс к элементу или удалить его из того или иного элемента. Предположим, к примеру, что у нас есть класс под названием `read`, который используется для придания стиля прочитанным постам блога. Добавить класс к посту можно просто с помощью метода `addClass`:

```
$('#post23').addClass('read')
```

За один вызов можно добавить сразу несколько классов, разделив их названия пробелами:

```
$('#post23').addClass('read liked')
```

А что делать, если пользователь решил снова пометить пост как непрочитанный, возможно, чтобы не забыть чуть позже прочитать его еще раз? В таком случае придется воспользоваться методом `removeClass`:

```
$('#post23').removeClass('read')
```

При этом на все остальные классы, использующиеся этим постом, не будет оказано никакого влияния.

При поддержке возможности постоянного добавления или удаления класса проще, наверное, будет воспользоваться методом `toggleClass`:

```
$('#post23').toggleClass('read')
```

Тогда, если пост еще не использовал класс, он будет добавлен, а если использовал — удален.

Работа с размерами

Работа с размерами всегда считалась в веб-разработке далеко не самой легкой задачей, поскольку различные браузеры склонны к использованию несколько различающихся значений. Одной из весьма сильных сторон библиотеки jQuery является приложение больших усилий к нормализации этих типов значений, чтобы во всех основных браузерах ваши страницы выглядели полностью соответствующими вашим замыслам.

Размеры бывают трех типов: ширина и высота элемента, внутренняя ширина и высота, внешняя ширина и высота. Рассмотрим их по очереди.

Методы width и height

Оба метода, как `width`, так и `height`, могут получать ширину или высоту первого элемента, соответствующего селектору, или устанавливать ширину или высоту всех соответствующих селектору элементов. Например, для получения ширины элемента с идентификатором `elem` можно воспользоваться следующей инструкцией:

```
width = $('#elem').width()
```

Значение, возвращаемое переменной `width`, будет числовым, что отличается от CSS-значения, возвращаемого после вызова метода `css`, как в следующей инструкции, в результате выполнения которой возвращается (к примеру) значение `230px`, а не просто число `230`:

```
width = $('#elem').css('width')
```

Можно получить ширину как текущего окна, так и документа:

```
width = $(window).width()
width = $(document).width()
```

Когда библиотеке jQuery передаются объекты окна или документа, получить их ширину или высоту с помощью метода css невозможно. Вместо него нужно воспользоваться методами width или height.

Возвращаемое значение не зависит от установок свойства `box-sizing` (см. главу 19). Если нужно взять в расчет значение свойства `box-sizing`, следует, как показано в следующей инструкции, воспользоваться вместо этого методом `css` с аргументом `width` (но если вы собираетесь продолжить работу с возвращенными значениями, не забудьте удалить символы `px`, которые будут добавлены после числовой части):

```
width = $('#elem,).css('width')
```

Точно так же легко можно устанавливать значения. Например, для установки размеров всех элементов, задействующих класс `box` 100 × 100 пикселов, можно воспользоваться следующей инструкцией:

```
$('.box').width(100).height(100)
```

В примере 21.19 эти действия объединены в одну программу, результат выполнения которой показан на рис. 21.17.

Пример 21.19. Получение и установка размеров элементов

```
<!DOCTYPE html>
<html>
  <head>
    <title>Работа с размерами</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
```

```
  <body>
    <p>
      <button id='getdoc'>Получить ширину документа </button>
      <button id='getwin'>Получить ширину окна</button>
      <button id='getdiv'>Получить ширину div-элемента</button>
      <button id='setdiv'>Установить ширину div-элемента равной
        150 пикселам</button>
    </p>
    <div id='result' style='width:300px; height:50px; background:#def;'>
    </div>
    <script>
      $('#getdoc').click(function()
      {
        $('#result').html('Ширина документа: ' + $(document).width())
      } ) $('#getwin').click(function()
      {
        $('#result').html('Ширина окна: ' + $(window).width())
      } )
      $('#getdiv').click(function()
      {
        $('#result').html('Ширина div-элемента: ' + $('#result').width())
      } )
      $('#setdiv').click(function()
      {
        $('#result').width(150)
        $('#result').html('Ширина div-элемента: ' + $('#result').width())
      } )
    </script>
  </body>
</html>
```

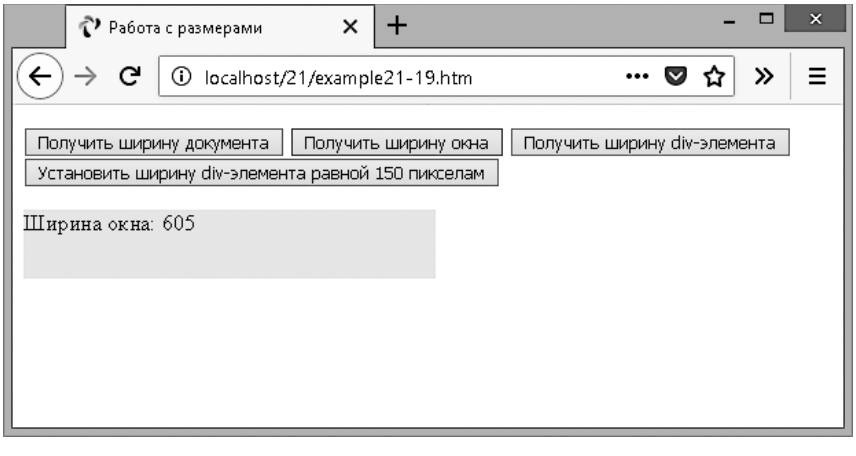

Рис. 21.17. Получение и установка размеров элементов

В начале раздела `<body>` объявляются четыре кнопки: три для получения отчета о ширине документа, окна и элемента `<div>`, который появляется под кнопками, и одна для установки нового значения ширины `div`-контейнера. В разделе `<script>`

находятся четыре jQuery-инструкции, первые три из которых просто извлекают ширину заданных объектов, а затем сообщают о полученных значениях, записывая их в HTML-код `div`-контейнера.

Последняя инструкция состоит из двух частей: первая часть сокращает ширину `<div>`-элемента до 150 пикселов, а затем вторая часть выводит новое значение ширины `div`-контейнера, извлекая его с помощью метода `width`, чтобы гарантировать отображение вычисленного значения.

При изменении масштаба страницы (его увеличении или уменьшении) пользователем это событие в любом основном браузере ничем не отмечается, то есть никакого способа, позволяющего его достоверно обнаружить кодом JavaScript, не существует. Поэтому jQuery не может взять масштабирование в расчет при применении или возвращении значений размеров. Следовательно, в подобной ситуации можно получить непредсказуемые результаты.

Методы innerWidth и innerHeight

Зачастую возникает необходимость взять в расчет также границы, отступы и другие свойства, работающие с размерами, и поэтому вы можете воспользоваться для возвращения ширины и высоты первого элемента, соответствующего селектору, *включая* отступы, но *не включая* границы, методами `innerWidth` и `innerHeight`.

Например, следующая инструкция вернет значение свойства `innerWidth`, включающего отступы, имеющиеся у элемента с идентификатором `elem`:

```
iwidth = $('#elem').innerWidth()
```

Методы outerWidth и outerHeight

Для возвращения размеров элемента, включая не только отступы, но и границы, можно вызвать методы `outerWidth` и `outerHeight`:

```
owidth = $('#elem').outerWidth()
```

Если в возвращаемое значение нужно *также* включить еще и поля, то при вызове любого из этих методов можно передать ему значение `true`:

```
owidth = $('#elem').outerWidth(true)
```

Значения, возвращенные любыми методами inner... или outer..., не обязательно должны быть целыми числами и иногда могут иметь дробную составляющую. Изменение масштаба страницы пользователем этими методами не обнаруживается, и их нельзя использовать в отношении объектов окон или документов, для которых нужно применять методы width или height.

Обход объектов DOM

Если обратиться к разделу главы 13, рассказывающему об объектной модели документа — Document Object Model (DOM), можно вспомнить, что все веб-страницы во многом конструктивно напоминают большие семьи. В них имеются родительские и дочерние объекты, одноуровневые объекты, у которых общий родительский объект, объекты-предки, которые старше родительских объектов, и объекты-потомки, которые младше дочерних объектов, и даже элементы, чьи родственные отношения могут рассматриваться в качестве двоюродных сестер, тетей и т. д. Например, в следующем фрагменте кода `<li>`-элементы являются дочерними по отношению к `<ul>`-элементу, который, в свою очередь, является родительским для `<li>`-элементов:

```
<u1>
  <li>Элемент 1</li>
  <li>Элемент 2</li>
  <li>Элемент 3</li>
</u1>
```

И так же, как и в семьях, существует множество способов ссылки на HTML-элементы, например с помощью абсолютного указания, или путем указания, начинающегося с уровня окна и идущего далее вниз (по принципу, известному как *обход объектов DOM*). Кроме того, для ссылок на элементы можно воспользоваться родственными отношениями между одним и другим элементами. На самом деле весь вопрос заключается в том, что будет разумнее конкретно для вашего проекта.

Например, если нужно добиться наибольшей автономности веб-страницы, чтобы получить больше шансов на возможность вырезать и вставлять элементы в другие веб-документы без необходимости изменения вставляемого HTML, разумнее будет ссылаться на близлежащие элементы, применяя относительную адресацию. Но какой бы из способов вы ни выбрали, jQuery предлагает широкий диапазон функций, помогающих осуществлять конкретную адресацию элементов.

Родительские элементы

Чтобы обратиться к непосредственному родителю элемента, нужно воспользоваться методом `parent`:

```
my_parent = $('#elem').parent()
```

Независимо от типа элемента `elem` объект `my_parent` теперь будет содержать jQuery-объект, ссылающийся на его родительский элемент. На самом же деле, поскольку селектор может ссылаться на несколько элементов, этот вызов приведет к возвращению объекта, ссылающегося на список родительских элементов (хотя в списке может быть всего одна запись), по одной ссылке для каждого соответствующего элемента.

Так как у родителя может быть много детей, вы можете поинтересоваться, будет ли этим методом возвращено больше элементов, чем имеется родителей. Возьмем предыдущий фрагмент кода с тремя -элементами. Если применить следующий код:

```
my_parent = $('li').parent()
```

возникнет вопрос: будут ли возвращены три родительских элемента (поскольку будет найдено три соответствия), даже если имеется всего один родительский -элемент? Ответ будет отрицательным, поскольку библиотека jQuery достаточно интеллектуальна, чтобы распознать все дубликаты и провести фильтрацию. Если для проверки запросить количество возвращенных элементов, будет возвращен результат 1:

```
alert($('li').parent().length)
```

Инициируем какие-либо изменения при нахождении соответствий селектору, например изменим значение свойства font-weight родительского элемента в предыдущем фрагменте кода на bold:

```
$('li').parent().css('font-weight', 'bold')
```

Использование фильтра

Дополнительно методу parent может быть передан селектор, чтобы отфильтровать те родительские элементы, к которым должно быть применено желаемое изменение. В качестве иллюстрации в примере 21.20 имеются три небольших списка и две jQuery-инструкции.

Пример 21.20. Обращение к родительским элементам

```
<!DOCTYPE html>
<html>
  <head>
    <title>Обход DOM: Parent</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <ul>
      <li>Элемент 1</li>
      <li>Элемент 2</li>
      <li>Элемент 3</li>
    </ul>
    <ul class='memo'>
      <li>Элемент 1</li>
      <li>Элемент 2</li>
      <li>Элемент 3</li>
    </ul>
    <ul>
      <li>Элемент 1</li>
      <li>Элемент 2</li>
      <li>Элемент 3</li>
    </ul>
```

```
    <script>
      $('li').parent()       .css('font-weight',     'bold')
      $('li').parent('.memo').css('list-style-type', 'circle')
    </script>
  </body>
</html>
```

Все три списка совершенно одинаковы, за исключением того, что у среднего в элементе `<ul>` используется класс `memo`. В разделе `<script>` первая инструкция применяет значение `bold` к свойству `font-weight` всех элементов, являющихся родительскими для элементов `<li>`. В данном случае ее выполнение приводит к тому, что все `<ul>`-элементы отображаются полужирным шрифтом.

Вторая инструкция похожа на первую, но в ней методу `parent` вдобавок ко всему передается имя класса `memo`, поэтому будет выбран только тот родительский элемент, у которого есть такой класс. Затем вызывается метод `css`, чтобы установить для свойства `list-style-type` выбранного списка значение `circle`. Результат выполнения этих двух инструкций показан на рис. 21.18.

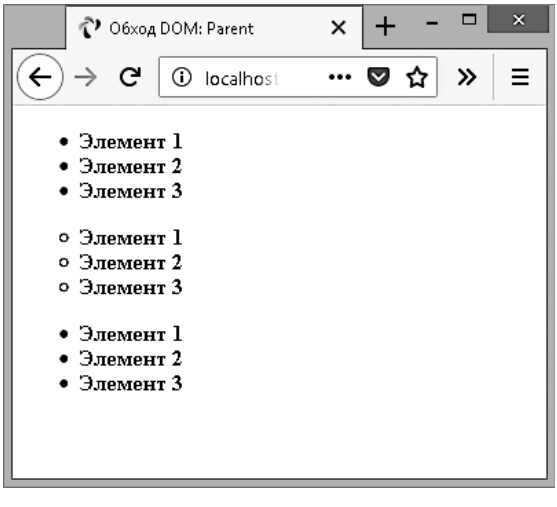

Рис. 21.18. Обращение к родительским элементам с фильтром и без фильтра

Выбор всех элементов-прародителей

Только что были рассмотрены способы выбора непосредственных родительских элементов, но с помощью метода `parents` можно выбрать и предков вплоть до корневого элемента `<html>`. Но зачем это может понадобиться? К примеру, чтобы обратиться к первому `<div>`-элементу, расположенному вверх по цепочке прародителей, с целью придания ему стилевого оформления в соответствии с каким-либо развитием событий, произошедшим в отношении какого-либо элемента где-нибудь ниже по цепочке.

Тип выбора может быть несколько изощреннее, чем тот, который имело бы смысл применить в обычных условиях, но вас он вполне устроит, когда возникнет подобная необходимость. Вот как можно было бы продолжить реализацию данного замысла:

```
$('#elem').parents('div').css('background', 'yellow')
```

Вообще-то результат может быть не совсем тот, которого вы добивались, поскольку выбраны будут все `<div>`-элементы в цепочке прародителей, а в ней могут быть те, которые располагаются еще выше, изменять стиль которых вам совсем не хотелось. Для подобных случаев можно применить дополнительную фильтрацию выбора, воспользовавшись вместо предыдущего метода методом `parentsUntil`.

Метод `parentsUntil` совершает обход вверх по цепочке прародителей точно так же, как и метод `parents`, но останавливается на первом элементе, который соответствует фильтру выбора (в данном случае это `<div>`-элемент). То есть этот метод можно использовать точно так же, как и метод в предыдущем примере, будучи уверенными в том, что будет выбран наиболее соответствующий вашему желанию элемент:

```
$('#elem').parentsUntil('div').css('background', 'yellow')
```

Чтобы понять разницу между этими двумя методами, посмотрите код примера 21.21, в котором имеются два набора вложенных элементов, у каждого из которых один родительский `<div>`-элемент. Затем в разделе `<script>` находятся по одному примеру вызова методов `parents` и `parentsUntil`.

Пример 21.21. Использование методов parents и parentsUntil

```
<!DOCTYPE html>
<html>
  <head>
    <title>Обход DOM: Parents</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <div>
      <div>
        <section>
          <blockquote>
            <ul>
              <li>Элемент 1</li>
              <li id='elem'>Элемент 2</li>
              <li>Элемент 3</li>
            </ul>
          </blockquote>
        </section>
      </div>
      <div>
        <section>
          <blockquote>
            <ul>
```

```
            <li>Элемент 1</li>
            <li>Элемент 2</li>
            <li>Элемент 3</li>
          </ul>
        </blockquote>
      </section>
    </div>
  </div>
  <script>
    $('#elem').parents('div') .css('background', 'yellow')
    $('#elem').parentsUntil('div').css('text-decoration', 'underline')
  </script>
 </body>
</html>
```

На рис. 21.19 можно увидеть, что первая jQuery-инструкция установила желтый цвет фона для всего содержимого. Дело в том, что при использовании метода `parents` дерево прародителей было пройдено вверх вплоть до элемента `<html>` и были выбраны оба встреченных на пути `<div>`-элемента (тот, что содержит список `<li>`-элементов, выделенный полужирным шрифтом, и имеет идентификатор `elem`, и его родительский `<div>`-элемент, в котором содержатся оба набора вложенных элементов).

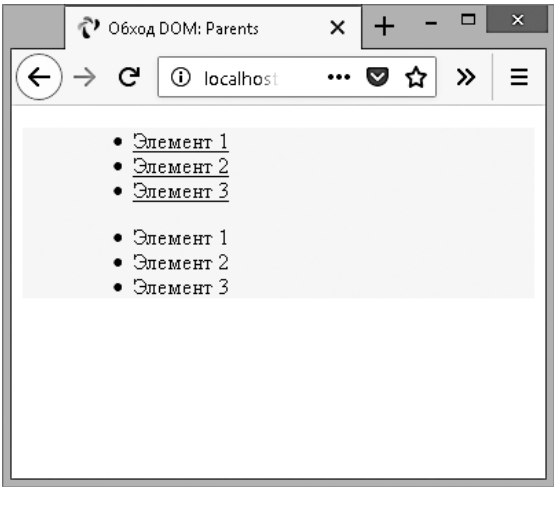

Рис. 21.19. Сравнение методов parents и parentsUntil

А во второй инструкции используется метод `parentsUntil`, поэтому выбор останавливается на первом же встреченном `<div>`-элементе. Это означает, что применение стиля подчеркивания касается только ближайшего родительского `<div>`-элемента, в котором содержится `<li>`-элемент с идентификатором `elem`. До внешнего `<div>`-элемента установка не доходит, и, поскольку он не получает данного стилевого оформления, второй список с подчеркиванием не выводится.

Дочерние элементы

Для обращения к дочерним элементам используется метод `children`:

```
my_children = $('#elem').children()
```

Как и у метода `parent`, его действие распространяется только на один уровень вниз, и в результате возвращается список, либо не содержащий ссылок на объекты, либо содержащий одну соответствующую ссылку или более. Методу можно также передавать аргумент фильтра для выбора среди дочерних элементов:

```
li_children = $('#elem').children('li')
```

В результате выполнения этого кода будут выбраны только те дочерние элементы, которые являются `<li>`-элементами.

Чтобы углубиться в поколения, нужно воспользоваться методом `find`, являющимся противоположностью метода `parents`:

```
li_descendants = $('#elem').find('li')
```

Но, в отличие от `parents`, методу `find` нужно предоставить селектор, используемый в качестве фильтра, если же нужно выбрать всех потомков, можно воспользоваться универсальным селектором:

```
all_descendants = $('#elem').find('*')
```

Одноуровневые элементы

Для выбора одноуровневых элементов доступен более широкий диапазон методов, начинающийся с метода `siblings`.

Метод `siblings` возвращает все соответствующие элементы, являющиеся дочерними по отношению к одному и тому же родителю, за исключением элемента, используемого для выбора. Таким образом, применительно к следующему фрагменту кода, если ведется поиск одноуровневых элементов для `<li>`-элемента с идентификатором `two`, будут возвращены только первый и третий `<li>`-элементы:

```
<ul>
  <li>Item 1</li>
  <li id='two'>Item 2</li>
  <li>Item 3</li>
</ul>
```

Например, следующая инструкция приведет к выводу на экран первого и третьего из одноуровневых элементов полужирным шрифтом:

```
$('#two').siblings().css('font-weight', 'bold')
```

Чтобы сузить круг возвращаемых одноуровневых элементов, с методом `siblings` можно применять фильтр. Например, для выбора только тех одноуровневых

элементов, которые используют класс new, можно воспользоваться следующей инструкцией:

```
$('#two').siblings('.new').css('font-weight', 'bold')
```

В примере 21.22 (где щедро расставлены пробелы, чтобы выровнять атрибуты по вертикали) показывается неупорядоченный список из семи элементов, у четырех из которых имеется класс new, а у второго элемента — идентификатор two.

Пример 21.22. Выбор и фильтрация одноуровневых элементов

```
<!DOCTYPE html>
<html>
  <head>
    <title>Обход DOM: Siblings</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <ul>
      <li          class='new'>Элемент 1</li>
      <li id='two' class='new'>Элемент 2</li>
      <li                     >Элемент 3</li>
      <li          class='new'>Элемент 4</li>
      <li          class='new'>Элемент 5</li>
      <li                     >Элемент 6</li>
      <li                     >Элемент 7</li>
    </ul>
    <script>
      $('#two').siblings('.new').css('font-weight', 'bold')
    </script>
  </body>
</html>
```

Результат применения инструкции jQuery при загрузке кода в браузер показан на рис. 21.20, где полужирным шрифтом выделены только элементы Элемент 1, Элемент 4 и Элемент 5, даже притом, что Элемент 2 также использует класс new (поскольку метод вызывается в отношении именно этого элемента, он исключается из выбора).

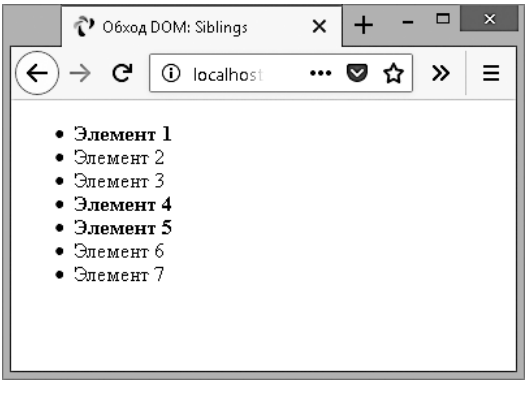

Рис. 21.20. Выбор и фильтрация одноуровневых элементов

Поскольку элемент, в отношении которого был вызван метод siblings (и который я буду называть элементом вызова), опускается, этот метод не может использоваться для выбора всех дочерних элементов родительского элемента. Но для достижения этой цели в предыдущем примере можно воспользоваться следующей инструкцией, которая, к примеру, вернет все одноуровневые элементы (включая элемент вызова), имеющие класс new:

```
$('#two').parent0.children('.new')
.css('font-weight', 'bold')
```

Для достижения такого же результата для выбора можно воспользоваться также методом andBack:

```
$('#two').siblings('.new').andBack()
.css('font-weight', 'bold')
```

Выбор следующих и предыдущих элементов

Когда нужен более жесткий контроль над выбором одноуровневых элементов, можно с помощью методов `next` и `prev` и их расширенных версий сузить круг возвращаемых элементов еще больше. Например, для ссылки на элемент, непосредственно следующий за выбранным элементом, можно воспользоваться такой инструкцией (которая установит для соответствующих элементов или элемента отображение полужирным шрифтом):

```
$('#new').next().css('font-weight', 'bold')
```

Применительно к следующему фрагменту кода со щедро расставленными пробелами идентификатор `new` имеет третий элемент, и поэтому возвращается четвертый элемент:

```
<ul>
  <li         >Элемент 1</li>
  <li         >Элемент 2</li>
  <li id='new'>Элемент 3</li>
  <li         >Элемент 4</li>
  <li         >Элемент 5</li>
</ul>
```

Пока ничего сложного нам не попадалось. А что делать, если нужно получить ссылку на *все* одноуровневые элементы, следующие за конкретно заданным элементом? Это можно сделать с помощью метода `nextAll` (который применительно к предыдущему фрагменту кода придаст стилевое оформление последним двум элементам):

```
$('#new').nextAll().css('font-weight', 'bold')
```

При вызове метода `nextAll` ему также можно предоставлять фильтр, чтобы выбирать соответствующие элементы, как в следующей инструкции, которая задаст стилевое оформление только тем одноуровневым элементам, которые применяют класс `info` (но в предыдущем фрагменте кода нет элементов, имеющих такой класс,

поэтому инструкция, запущенная в отношении этого фрагмента, ни к какому результату не приведет):

```
$('#new').nextAll('.info').css('font-weight', 'bold')
```

Или же рассмотрим случай применения следующего фрагмента кода, в котором у одного элемента имеется идентификатор `new`, а у другого — `old`:

```
<ul>
   <li          >Элемент 1</li>
   <li id='new' >Элемент 2</li>
   <li          >Элемент 3</li>
   <li id='old' >Элемент 4</li>
   <li          >Элемент 5</li>
</ul>
```

Теперь есть возможность выбрать только одноуровневые элементы, следующие за тем элементом, у которого имеется идентификатор `new`, и доходящие до элемента с идентификатором `old`, исключая сам этот элемент (в данном случае стилевое оформление получит третий элемент):

```
$('#new').nextUntil('#old').css('font-weight', 'bold')
```

Если методу `nextUntil` аргументы не передаются, он ведет себя точно так же, как и метод `nextAll`, возвращая все следующие одноуровневые элементы. Методу `nextUntil` можно передать и второй аргумент, чтобы он действовал в качестве фильтра для выбора из элементов, соответствующих указанному селектору:

```
$('#new').nextUntil('#old', '.info').css('font-weight', 'bold')
```

В этой инструкции стилевое оформление получат только те элементы, которые используют класс `info`, а таких элементов в предыдущем фрагменте кода нет, следовательно, никаких действий предпринято не будет.

Абсолютно то же самое можно сделать в группе одноуровневых элементов в обратном направлении, используя методы `prev`, `prevAll` и `prevUntil`.

Обход элементов, выбранных с помощью методов jQuery

Так же, как и при обходе объектов DOM, можно обойти и возвращенный набор элементов, выбранных с помощью jQuery, чтобы выбрать те из них, в отношении которых нужно выполнить определенные действия.

Например, для придания стиля только первым элементам, возвращенным после выбора, можно воспользоваться методом `first` (чтобы первый элемент списка в первом неупорядоченном списке выводился на экран подчеркнутым):

```
$('ul>li').first().css('text-decoration', 'underline')
```

Или же, воспользовавшись методом `last`, можно выбрать для стилевого оформления только самый последний элемент:

```
$('ul>li').last().css('font-style', 'italic')
```

А для обращения к элементу по индексу (нумерация индексов начинается с нуля) можно воспользоваться методом `eq` (при этом стилевое оформление получит второй элемент в списке, поскольку отсчет начинается с нуля):

```
$('ul>li').eq(1).css('font-weight', 'bold')
```

Можно также применить к выбранным элементам фильтр, воспользовавшись для этого методом `filter` (при этом изменится цвет фона каждого второго элемента, начиная с первого элемента, имеющего нулевой индекс):

```
$('ul>li').filter(':even').css('background', 'cyan')
```

Следует помнить, что при индексации элементов, выбранных средствами jQuery, первым всегда является элемент с нулевым индексом. Поэтому, к примеру, при использовании подобным образом селектора :even будут выбраны элементы 1, 3, 5 и т. д. (а не 2, 4, 6...).

Чтобы исключить один или несколько элементов, можно применить метод `not` (при этом те элементы, которые не используют идентификатор `new`, будут выведены синим цветом):

```
$('ul>li').not('#new').css('color', 'blue')
```

Элемент можно выбрать также в зависимости от того, какие у него потомки. К примеру, для выбора только тех элементов, у которых в качестве потомков имеются элементы `<ol>`, чтобы их перечеркнуть, можно воспользоваться следующей инструкцией:

```
$('ul>li').has('ol').css('text-decorationI, 'line-through')
```

В примере 21.23, где задается стилевое оформление неупорядоченного списка и один из элементов которого представляет собой упорядоченный список, все эти приемы собраны воедино.

Пример 21.23. Обход элементов, выбранных с помощью методов jQuery

```
<!DOCTYPE html>
<html>
  <head>
    <title>Обход выбранных элементов</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <ul>
      <li>Элемент 1</li>
```

```
      <li>Элемент 2</li>
      <li id='new'>Элемент 3</li>
      <li>Элемент 4
        <ol type='a'>
          <li>Элемент 4a</li>
          <li>Элемент 4b</li>
        </ol></li>
      <li>Элемент 5</li>
    </ul>
    <script>
      $('ul>li').first()        .css('text-decoration', 'underline')
      $('ul>li').last()         .css('font-style',      'italic')
      $('ul>li').eq(1)          .css('font-weight',     'bold')
      $('ul>li').filter(':even').css('background',      'cyan')
      $('ul>li').not('#new')    .css('color',           'blue')
      $('ul>li').has('ol')      .css('text-decoration', 'line-through')
    </script>
  </body>
</html>
```

Как показано на рис. 21.21, каждый элемент в каждом списке получил стилевое оформление за счет применения одной или нескольких jQuery-инструкций.

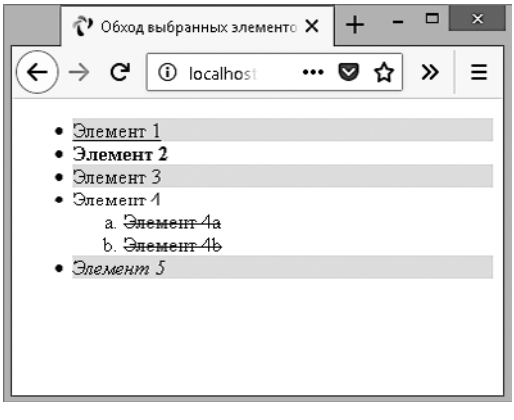

Рис. 21.21. Конкретная адресация элементов из тех, что были выбраны методами jQuery

Метод is

Метод `is` заставляет jQuery-селектор вернуть булево значение, которое затем можно использовать в обычном коде JavaScript. В отличие от других имеющихся в jQuery методов фильтрации, показанных в предыдущих разделах, этот метод не создает новый объект jQuery, к которому затем можно применить другие методы или который можно подвергнуть дальнейшей фильтрации.

Вместо этого возвращается значение `true` или `false`, что делает метод наиболее подходящим для использования в условных инструкциях. В примере 21.24 метод `is`

прикреплен для вызова родительского элемента в обработчике события для набора кнопок. Обработчик вызывается при нажатии любой кнопки, а метод `is` может вернуть значение `true` или `false` при выяснении вопроса, является родительский элемент `<div>`-элементом или нет (рис. 21.22).

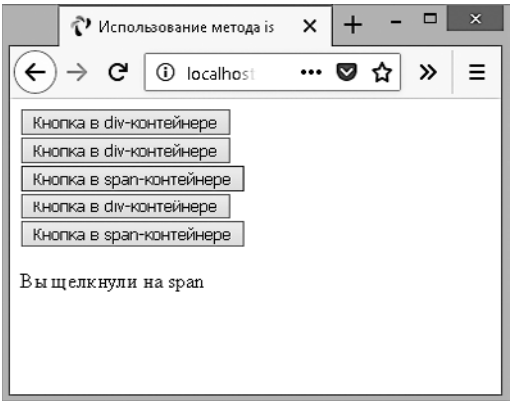

Рис. 21.22. Использование метода is для получения сведений о том, чем именно является родительский элемент

Пример 21.24. Получение с помощью метода is сведений о том, чем именно является родительский элемент

```
<!DOCTYPE html>
<html>
  <head>
    <title>Использование метода is</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <div><button>Кнопка в div-контейнере</button></div>
    <div><button>Кнопка в div-контейнере</button></div>
    <span><button>Кнопка в span-контейнере</button></span>
    <div><button>Кнопка в div-контейнере</button></div>
    <span><button>Кнопка в span-контейнере</button></span>
    <p id='info'></p>
    <script>
      $('button').click(function()
      {
        var elem = ''

        if ($(this).parent0.is('div')) elem = 'div'
        else                           elem = 'span'

        $('#info').html('Вы щелкнули на ' + elem)
      })
    </script>
  </body>
</html>
```

Использование jQuery без селекторов

Существуют также два jQuery-метода, предназначенных для использования со стандартными объектами JavaScript и существенно упрощающих работу с ними. Это похожие друг на друга, но все же имеющие небольшие различия методы `$.each` и `$.map`.

Метод $.each

Используя метод `$.each`, можно осуществить последовательный перебор элементов массивов или подобных массивам объектов путем простого прикрепления к функции, вызываемой для каждой итерации. В примере 21.25 показан массив из имен и видов домашних животных (названный `pets`), из которого должен быть извлечен другой массив (названный `guineapigs`), содержащий только имена морских свинок.

Пример 21.25. Вызов метода $.each

```
<!DOCTYPE html>
<html>
  <head>
    <title>Использование метода each</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <div id='info'></div>
    <script>
      pets =
      {
        Scratchy : 'Guinea Pig',
        Squeeky  : 'Guinea Pig',
        Fluffy   : 'Rabbit',
        Thumper  : 'Rabbit',
        Snoopy   : 'Dog',
        Tiddles  : 'Cat'
      }

      guineapigs = []

      $.each(pets, function(name, type)
      {
        if (type == 'Guinea Pig') guineapigs.push(name)
      })

      $('#info').html('Имена морских свинок: ' + guineapigs.join(' & '))
    </script>
  </body>
</html>
```

Для выполнения поставленной задачи методу `$.each` передаются массив и безымянная функция для его обработки. Функция получает два аргумента, индекс в массиве (называемый `name`) и содержимое каждого элемента (называемое `type`).

Затем значение, имеющееся в `type`, проверяется на предмет того, не содержит ли оно строку `Guinea Pig`, и если содержит, значение, имеющееся в `name`, вставляется в массив `guineapigs`. По завершении содержимое массива `guineapigs` выводится на экран путем записи его в `<div>`-элемент с идентификатором `info`. Чтобы отделить элементы массива друг от друга, используется JavaScript-метод `join` с символом `&` в качестве разделителя. В результате загрузки примера в браузер на экране будет просто отображен текст `Имена морских свинок: Scratchy & Squeeky`.

Метод $.map

То же самое можно сделать и с помощью метода `$.map`, который возвращает все значения, возвращенные функцией, в виде массива, освобождая вас от необходимости создания массива с последующим помещением в него соответствующих значений, что приходилось делать в предыдущем примере.

Вместо этого создать и наполнить массив можно, одновременно присвоив ему тот массив, который будет возвращен методом `$.map` (итоговый результат будет тем же, но для его получения используется меньший объем кода):

```
guineapigs = $.map(pets, function(type, name)
{
  if (type == 'Guinea Pig') return name
})
```

Будьте внимательны, используя методы $.each и $.map один вместо другого, поскольку метод $.each передает аргументы функции в последовательности «индекс, значение», а в map используется последовательность «значение, индекс». Именно поэтому в предыдущем примере использования метода $.map аргументы поменялись местами.

Использование асинхронного обмена данными

В главе 17 был подробно показан порядок реализации асинхронного обмена данными между кодом JavaScript в браузере и кодом PHP, запущенным на веб-сервере. Кроме того, там был представлен ряд удобных и компактных функций, которые можно будет вызывать для упрощения процесса.

Но если у вас загружена библиотека jQuery, можно вместо них отдать предпочтение использованию асинхронных функций, имеющихся в этой библиотеке, которые работают очень похоже вне зависимости от того, какой запрос был вами выбран, POST или GET, и взять их из этой библиотеки.

Метод POST

В примере 21.26 показан непосредственный jQuery-эквивалент примера 17.2 (загружающий сайт Amazon Mobile в `<div>`-контейнер), но, поскольку весь код, занимающийся обработкой асинхронного обмена данными, убран в файл библиотеки jQuery, пример получился гораздо короче. Ему требуется один-единственный вызов метода `$.post`, которому передаются следующие три аргумента:

- URL-адрес PHP-программы на сервере, к которой нужно обратиться;
- данные для передачи по этому URL-адресу;
- безымянная функция для обработки возвращаемых данных.

Пример 21.26. Отправка асинхронного POST-запроса

```
<!DOCTYPE html>
<html> <!-- jqueryasuncpost.htm -->
  <head>
    <title>jQuery асинхронный POST-запрос</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Загрузка веб-страницы в DIV</h1>
    <div id='info'>Это предложение будет заменено</div>

    <script>
      $.post('urlpost.php', { url : 'amazon.com/gp/aw' }, function(data)
      {
        $('#info').html(data)
      } )
    </script>
  </body>
</html>
```

Программа `urlpost.php` остается такой же, какой она была в примере 17.3, поскольку оба примера взаимозаменяемы.

Метод GET

Асинхронный обмен с данными использованием GET-метода осуществляется практически так же легко и требует только двух следующих аргументов:

- URL-адрес PHP-программы на сервере, к которой нужно обратиться (включающий строку запроса, содержащую передаваемые этой программе данные);
- безымянная функция для обработки возвращаемых данных.

Пример 21.27 является jQuery-эквивалентом примера 17.4.

Пример 21.27. Отправка асинхронного GET-запроса

```
<!DOCTYPE html>
<html> <!-- jqueryasuncget.htm -->
```

```
  <head>
    <title>jQuery асинхронный GET-запрос</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Загрузка веб-страницы в DIV</h1>
    <div id='info'>Это предложение будет заменено</div>

    <script>
      $.get('urlget.php?url=amazon.com/gp/aw', function(data)
      {
        $('#info').html(data) } )
    </script>
  </body>
</html>
```

Программа `urlget.php` остается такой же, какой она была в примере 17.5, поскольку оба примера взаимозаменяемы.

Следует помнить, что ограничительные меры безопасности асинхронного обмена данными требуют, чтобы этот обмен велся с тем же сервером, с которого был предоставлен основной веб-документ. Для асинхронного обмена данными нужно также использовать веб-сервер, а не локальную файловую систему. Поэтому данные примеры лучше всего тестировать, используя, согласно описаниям, приведенным в главе 2, производственный или разработочный сервер.

Дополнительные модули (плагины)

В данной главе места хватило только для описания основной библиотеки jQuery, и, хотя начинающим этого более чем достаточно для того, чтобы освоиться с новыми возможностями, настанет время, когда потребуются новые функциональные возможности. К счастью, здесь вам смогут помочь другие проекты jQuery, поскольку круг доступных в настоящее время официальных и неофициальных дополнительных модулей может предоставить вам любые функции, которые только можно себе представить.

Пользовательский интерфейс jQuery

В первую очередь следует упомянуть библиотеку дополнительных модулей пользовательского интерфейса jQuery (известную как jQuery UI, http://jqueryui.com), которую можно взять там же, где и саму библиотеку jQuery. Применяя эти дополнения, можно добавлять к своим веб-страницам методы перетаскивания, измене-

ния размеров и сортировки, а также использовать на них еще больше эффектов, в том числе анимационных, применяя их также к цветовым переходам, получать расширенные возможности изменения скорости выполнения эффектов, а также применять пакеты виджетов для создания меню и других визуальных объектов, таких как разъезжающиеся вкладки, кнопки, календари для выбора дат, индикаторы хода выполнения, ползунки, пошаговые переключатели, вкладки, подсказки и многое другое.

Если перед тем, как принять решение о загрузке какого-либо дополнительного модуля, есть желание просмотреть демонстрацию его работы, зайдите на страницу сайта по адресу http://jqueryui.com/demos.

Весь пакет занимает чуть меньше 400 Кбайт сжатого кода, и его можно абсолютно свободно использовать практически без всяких ограничений (по весьма щедрой MIT-лицензии).

Другие дополнительные модули

В реестре jQuery Plugin Registry (http://plugins.jquery.com) содержится подборка широкого разнообразия бесплатных и уже готовых к работе дополнительных модулей к jQuery от многих разработчиков. В их числе модули для расширенной обработки и проверки форм, для демонстрации слайдов, для живых откликов на действия пользователей, для работы с изображениями, для получения еще большего количества эффектов анимации и для многого другого.

Если разработка ведется для браузеров мобильных устройств, то вам непременно захочется посмотреть на вариант jQuery, предназначенный для работы на этих устройствах, jQuery Mobile (см. главу 22), предоставляющий специализированные, оптимизированные для использования сенсорных экранов способы навигации с применением широкого диапазона различных типов мобильного оборудования и программных средств с целью обеспечения наиболее выгодного восприятия пользователями.

При чтении данной главы вами была проделала немалая работа, вы изучили материал, который порой становится темой отдельных книг. Но я надеюсь, что вам удалось во всем разобраться, поскольку jQuery очень легко поддается изучению и использованию. А теперь, пожалуйста, уделите еще немного времени и внимательно прочитайте приложение Д, в котором перечислены основные объекты, события и методы jQuery и которое должно послужить вам весьма удобным справочником. Если нужна другая информация, обратитесь, пожалуйста, на сайт jQuery по адресу http://jquery.com.

Вопросы

1. Какой символ обычно используется в качестве фабричного метода для создания объектов jQuery и какое имя применяется альтернативным методом?
2. Как создать ссылку на минимизированную версию выпуска 3.2.1 jQuery, получаемую из Google CDN?
3. Какие типы аргументов приемлемы для фабричного метода jQuery?
4. С помощью какого метода jQuery можно получить или установить значение CSS-свойства?
5. Какую инструкцию нужно использовать для прикрепления метода к событию щелчка на элементе с идентификатором `elem`, чтобы этот элемент медленно исчез?
6. Какое свойство элемента нужно изменить в целях применения к нему эффекта анимации и какими могут быть приемлемые значения?
7. Как можно добиться одновременного (или последовательного в случае применения анимации) запуска сразу нескольких методов?
8. Как извлечь узловой объект элемента из объекта, выбранного средствами jQuery?
9. Какая инструкция установит отображение полужирным шрифтом одноуровневого элемента, который следует непосредственно перед элементом с идентификатором `news`?
10. С применением какого метода средствами jQuery делается асинхронный GET-запрос?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

22 Введение в jQuery Mobile

После изучения главы 21 и осознания того, что использование jQuery существенно экономит время и повышает эффективность разработки программ, вам, видимо, будет приятно узнать о дальнейшем расширении возможностей за счет применения библиотеки jQuery Mobile.

Созданная в качестве дополнения к jQuery, библиотека jQuery Mobile требует включения в веб-страницу обеих этих библиотек (а также использования файла CSS и дополнительных изображений). Это позволяет при просмотре страницы на смартфонах и других мобильных устройствах воспользоваться полностью интерактивным интерфейсом.

Библиотека jQuery Mobile позволяет переделывать обычные веб-страницы, превращая их в страницы для мобильных устройств с помощью технологии под названием «постепенное улучшение» (при использовании которой сначала для получения наилучшего отображения применяются основные функции браузера, а затем происходит добавление функциональной насыщенности по мере раскрытия возможностей имеющегося на устройстве браузера). В библиотеке применяются также функции так называемого адаптивного веб-дизайна (благодаря чему достигается весьма качественное отображение веб-страниц на широком диапазоне устройств при различных размерах окон и экранов).

Основная цель данной главы не заключается в том, чтобы научить вас абсолютно всему, что нужно знать о jQuery Mobile, для чего может понадобиться целая книга! Мне хочется снабдить вас информацией, достаточной для преобразования любого не слишком большого объема веб-страниц в выстраиваемое в логическую последовательность быстрое и красивое веб-приложение со всеми слайд-показами страниц и другими переходами, ожидаемыми от современного сенсорного устройства, а также с более крупными и удобными в использовании значками, полями ввода и другими усовершенствованиями ввода и навигации.

С этой целью будет рассмотрен лишь небольшой объем основных функций jQuery Mobile, позволяющий приступить к разработке вполне понятных и работоспособных решений, которые могут запускаться как на настольных компьютерах, так и на мобильных платформах. Наряду с этим будут рассмотрены сложности, с которыми можно столкнуться на пути предлагаемой адаптации веб-страниц под мобильные устройства, и способы их преодоления. После освоения работы с библиотекой jQuery Mobile, станет проще ориентироваться в онлайн-документации при поиске функциональных свойств, необходимых при разработке ваших собственных проектов.

Кроме постепенного улучшения способа отображения вашего кода HTML в зависимости от возможностей браузера, на котором он обнаружит себя запущенным, библиотека jQuery Mobile также постепенно улучшает обычную HTML-разметку на основе используемых тегов и набора пользовательских атрибутов данных. Некоторые элементы подвергаются автоматическому улучшению, не нуждаясь при этом в каких-либо атрибутах данных (например, элементы выбора проходят автоматическое преобразование в меню), а некоторые требуют для усовершенствования наличия атрибутов данных. Полный перечень поддерживаемых атрибутов данных можно найти в документации по API.

Включение jQuery Mobile

Включить jQuery Mobile в ваши веб-страницы можно двумя способами. Можно перейти на страницу загрузки http://jquerymobile.com/download/, выбрать нужную версию, загрузить файлы на ваш веб-сервер (включая таблицы стилей и сопутствующие изображения, поставляемые вместе с библиотекой) и пользоваться всем этим отсюда.

Например, если в корневой каталог документов вашего сервера загружены библиотека jQuery Mobile 1.4.5 (это последняя версия на момент написания книги) и ее CSS-файл, можно включать в веб-страницу их и сопутствующую им JavaScript-библиотеку jQuery, у которой, применительно ко времени написания книги, *должна* быть версия 2.2.4, поскольку jQuery Mobile тогда еще не поддерживала версию 3 (но планировалось, что jQuery Mobile 1.5 станет поддерживать jQuery 3). Для этого воспользуйтесь следующим кодом:

```
<link href="http://myserver.com/jquery.mobile-1.4.5.min.css" rel="stylesheet">
<script src='http://myserver.com/jquery-2.2.4.min.js'></script>
<script src='http://myserver.com/jquery.mobile-1.4.5.min.js'></script>
```

Или же, как и в случае применения jQuery, можно воспользоваться предлагаемой в свободном доступе сетью доставки контента — content delivery network (CDN) и просто сослаться на требуемую вам версию (или версии). Свой выбор можно остановить на одной из трех основных CDN-сетей (Max CDN, Google

CDN и Microsoft CDN), нужные файлы из которых извлекаются следующими способами:

```
<!-- Retrieving jQuery & Mobile via Max CDN -->
<link rel="stylesheet"
  href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
<script src="http://code.jquery.com/jquery-2.2.4.min.js"></script>
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>

<!-- Retrieving jQuery & Mobile via Google CDN -->
<link rel="stylesheet" href=
  "http://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.css">
<script src=
  "http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script src=
  "http://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.js">
</script>

<!-- Retrieving jQuery & Mobile via Microsoft CDN -->
<link rel="stylesheet" href=
  "http://ajax.aspnetcdn.com/ajax/jquery.mobile/1.4.5/jquery.mobile-1.4.5.min.css">
<script src=
  "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.2.4.min.js"></script>
<script src=
  "http://ajax.aspnetcdn.com/ajax/jquery.mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>
```

Вероятно, один из наборов этих инструкций вам захочется разместить в <head>-разделе страницы.

Чтобы предоставить возможность использования приводимых примеров в автономном режиме, я загрузил все требуемые файлы jQuery и включил их с архивом файлов примеров в набор, который можно свободно загрузить с сайта, сопровождающего эту книгу (http://lpmj.net/). Поэтому во всех примерах показаны файлы, обслуживаемые в режиме локальной работы.

Начало работы

Давайте сразу углубимся в тему, наблюдая за общим внешним видом веб-страницы jQuery Mobile, воспользовавшись кодом, показанным в примере 22.1. По сути, ничего сложного в этом не будет, и такое беглое ознакомление поможет быстро разложить по своим местам весь остальной материал данной главы.

Пример 22.1. Одностраничный шаблон jQuery Mobile

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Одиночная страница</title>
    <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css">
    <script src="jquery-2.2.4.min.js"></script>
    <script src="jquery.mobile-1.4.5.min.js"></script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>Одиночная страница</h1>
      </div>
      <div data-role="content">
        <p>Это одностраничный шаблон страницы</p>
      </div>
      <div data-role="footer">
        <h4>Содержимое колонтитула</h4>
      </div>
    </div>
  </body>
</html>
```

При изучении предыдущего примера видно, что он начинается со стандартных и вполне ожидаемых элементов HTML5. Первым необычным элементом, который может броситься в глаза в разделе `<head>`, станет настройка окна просмотра `viewport`, осуществляемая внутри тега `<meta>`. Эта строка кода указывает мобильным браузерам на необходимость установки ширины показываемого документа в ширину окна браузера, и на начало работы с документом без увеличения или уменьшения его изображения. Как выглядит страница в браузере, показано на рис. 22.1.

После обозначения заголовка страницы загружается CSS для jQuery Mobile, затем загружается библиотека jQuery 2.2.4 и библиотека jQuery Mobile 1.4.5. Если хотите, можете загрузить их из сети CDN, о чем уже говорилось в разделе «Включение jQuery Mobile».

В примерах, сопровождающих эту главу, есть папка по имени images, содержащая значки и другие изображения, востребованные таблицами CSS. Если для загрузки CSS и файлов библиотек JavaScript используется сеть CDN, необходимость включения этой папки в ваш собственный проект может отпасть, поскольку вместо нее нужно будет воспользоваться папкой изображений, принадлежащей сети CDN.

Переместившись к разделу `<body>`, следует заметить, что главная веб-страница помещена в `<div>`-элемент. Он определяет значение свойства характера данных на странице `data-role` jQuery Mobile и содержит еще три `<div>`-элемента для заголовка, содержания и нижнего колонтитула страницы, у каждого из которых имеется соответствующее свойство `data-role`.

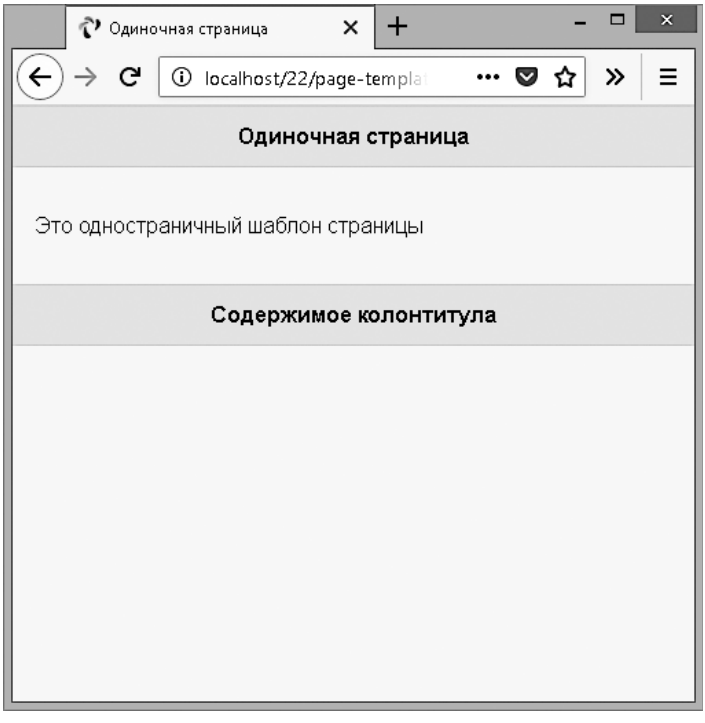

Рис. 22.1. Отображение одностраничного шаблона jQuery Mobile в окне браузера

Из этого складывается основная структура веб-страницы jQuery Mobile. Связывание страниц предполагает загрузку новых страниц и добавление их в DOM-модель с использованием асинхронного режима обмена данными. После загрузки страница может выводиться на экран множеством способов, включая мгновенную замену прежней страницы, плавное выведение новой страницы, постепенное превращение старой страницы в новую, наплыв новой страницы на старую и т. д.

Из-за асинхронной загрузки веб-страниц тестирование вашего кода jQuery Mobile всегда нужно проводить на веб-сервере, а не в локальной файловой системе. Это связано с тем, что именно веб-сервер обеспечивает асинхронную загрузку веб-страниц, необходимую для правильного обмена данными.

Связывание страниц

Библиотека jQuery Mobile позволяет связывать страницы обычным способом и выполнять (по возможности) асинхронную обработку страничных запросов, чтобы обеспечить применение любых выбранных вами переходов.

Это позволяет сосредоточиться просто на создании веб-страниц, возложив на средства библиотеки jQuery Mobile обязанности по формированию их удобного внешнего вида и быстрому их выводу на экран в профессиональном качестве.

Чтобы включить анимированные переходы от страницы к странице, все ссылки, указывающие на внешние страницы, приведут к загрузке страниц в асинхронном режиме. В jQuery Mobile это достигается за счет превращения всех ссылок вида `<a href...>` в асинхронные запросы на обмен данными (также известные как Ajax) с последующим выводом на экран кругового индикатора загрузки на время выполнения запроса.

Смена страниц после щелчка на ссылке выполняется в библиотеке jQuery Mobile за счет «перехвата» щелчков при доступе к событию event.preventDefault и последующего предоставления собственного особого кода jQuery Mobile.

Если запрос завершен успешно, к DOM-модели добавляется содержимое новой страницы, а затем происходит визуализация выбранной новой страницы с использованием перехода либо по умолчанию, либо по вашему выбору.

При неудачном завершении запроса ненадолго будет выведено небольшое и ненавязчивое уведомительное сообщение об ошибке, которое не помешает навигации по страницам.

Синхронная связанность

Данные по ссылкам, указывающим на другие домены или имеющим атрибуты `rel="external"`, `data-ajax="false"` или `target`, будут загружаться в синхронном режиме, что приведет к полному обновлению страницы без анимированных переходов.

Атрибуты `rel="external"` и `data-ajax="false"` приводят к одинаковому эффекту, но первый из них предназначен для ссылок на другой сайт или домен, а второй пригодится для предотвращений асинхронного режима загрузки любой страницы.

Из-за ограничений, накладываемых мерами безопасности, jQuery Mobile загружает страницы со всех внешних доменов в синхронном режиме.

При использовании выгрузки HTML-файла асинхронную загрузку страниц придется отключить, поскольку этот способ извлечения веб-страницы конфликтует с заложенной в jQuery Mobile возможностью получения выгружаемого файла. Для данного конкретного случая наилучшим вариантом, наверное, станет помещение атрибута data-ajax='false' в элемент <form>:

```
<form data-ajax='false' method='post'
  action='dest_file' enctype='multipart/form-data'>
```

Связанность внутри многостраничного документа

В одном HTML-документе может содержаться одна или несколько страниц. Последний вариант предполагает формирование структуры из `<div>`-элементов с указанием характера имеющихся на странице данных — `data-role`. Это позволяет создать в рамках *одного* HTML-документа небольшой сайт или приложение; библиотека jQuery Mobile просто отобразит на экране первую страницу, которая будет найдена в исходном порядке загрузки страниц.

Если ссылка в многостраничном документе указывает на гипертекстовый элемент (например, `#page2`), среда станет искать охватывающий страницу `<div>`-элемент с атрибутом `data-role`, имеющим значение `page`, и с заданным идентификатором `id` (`id="page2"`). Если такая страница будет найдена, произойдет ее вывод на экран.

Библиотека jQuery Mobile позволяет пользователям беспрепятственно перемещаться между всеми типами веб-страниц (внутренними, локальными или внешними). Для конечного пользователя все страницы будут выглядеть одинаково, за исключением внешних страниц, при вызове которых будет выводиться круговой индикатор загрузки Ajax. Во всех ситуациях jQuery Mobile обновляет страничный URL-хеш с целью поддержки функционирования кнопки возврата на предыдущую страницу. Это также означает, что страницы jQuery Mobile могут индексироваться поисковыми механизмами и не имеют никаких барьеров где-нибудь в своем родном приложении.

При связывании из страницы, отображенной в формате для мобильных устройств и загруженной в асинхронном режиме в документ, в котором содержатся несколько внутренних страниц, к ссылке нужно добавить либо rel="external", либо data-ajax="false", чтобы принудительно вызвать полную перезагрузку страницы, вычищая асинхронный хеш из URL. В страницах, загруженных в асинхронном режиме, знак решетки (#) используется для отслеживания их истории, в то время как во множестве внутренних страниц этот знак используется в качестве признака внутренних страниц.

Смена страниц

С помощью CSS-переходов jQuery Mobile может при условии использования асинхронной навигации (включаемой по умолчанию) применять эффекты к любой ссылке на страницу или к отправке формы.

Чтобы воспользоваться переходом, внутри тега `<a>` или `&lt;form&gt;` применяется атрибут `data-transition`:

```
<a data-transition="slide" href="destination.html">Нажми меня</a>
```

Этот атрибут поддерживает значения `fade` (начиная с версии 1.1 используется по умолчанию), `pop`, `flip`, `turn`, `flow`, `slidefade`, `slide` (использовалось по умолчанию до версии 1.1), `slideup`, `slidedown` и `none`.

Например, значение `slide` вызывает наплыв новой страницы справа при одновременном сдвиге текущей страницы влево. Эффект, вызываемый другими значениями, обусловливается их названиями.

Загрузка страницы, используемой в качестве диалогового окна

Новую страницу можно отобразить на экране в качестве диалогового окна, воспользовавшись атрибутом `data-rel` со значением `dialog`:

```
<a data-rel="dialog" href="dialog.html">Открыть диалоговое окно</a>
```

Применение различных страничных переходов как к загрузке страниц, так и к диалогам при локальной (не по сети CDN) загрузке библиотек jQuery показано в примере 22.2. Он состоит из простой таблицы в два столбца, первый из которых предназначен для загрузки диалога, а второй — для загрузки новой страницы. Перечислен каждый из доступных типов переходов. Чтобы ссылки выглядели как кнопки, каждая из них снабжена атрибутом `data-role` со значением `button` (кнопки рассматриваются в разделе «Стильные кнопки»).

Пример 22.2. Страничные переходы в jQuery Mobile

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Страничные переходы jQuery Mobile</title>
    <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css">
    <script src="jquery-2.2.4.min.js"></script>
    <script src="jquery.mobile-1.4.5.min.js"></script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>Страничные переходы jQuery Mobile</h1>
      </div>
      <div data-role="content"><table>
        <tr><th><h3>fade</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="fade" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="fade"
           data-role='button'>page</a></td>
```

```
        </tr><tr><th><h3>pop</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="pop" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="pop"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>flip</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="flip" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="flip"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>turn</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="turn" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="turn"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>flow</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="flow" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="flow"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>slidefade</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="slidefade" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="slidefade"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>slide</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="slide" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="slide"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>slideup</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="slideup" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="slideup"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>slidedown</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="slidedown" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="slidedown"
           data-role='button'>page</a></td>
        </tr><tr><th><h3>none</h3></th>
          <td><a href="page-template.html" data-rel="dialog"
           data-transition="none" data-role='button'>dialog</a></td>
          <td><a href="page-template.html" data-transition="none"
           data-role='button'>page</a></td></tr></table>
      </div>
      <div data-role="footer">
        <h4><a href="http://tinyurl.com/jqm-trans">Официальное демо</a></h4>
      </div>
    </div>
  </body>
</html>
```

На рис. 22.2 представлены результаты загрузки этого примера (сохраненного в файле `pagetemplate.html`) в браузер, а на рис. 22.3 показывается в действии переход в виде переворота (flip). Кстати, переход по ссылке http://demos.jquerymobile.com/1.4.4/transitions/, указанной в самом конце примера, приведет к сайту официальной демонстрации, где можно будет ознакомиться с этими эффектами более подробно.

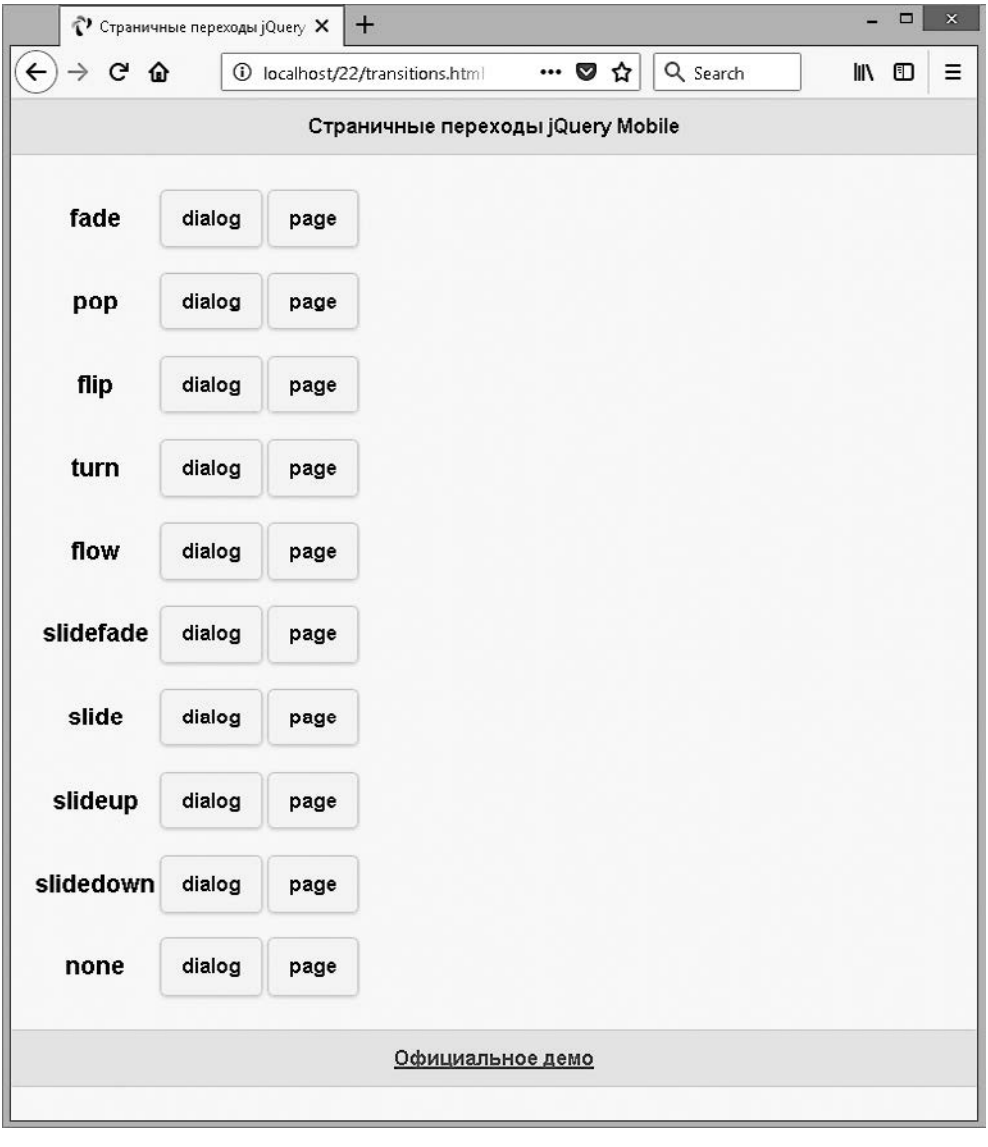

Рис. 22.2. Применение переходов к страницам и диалогам

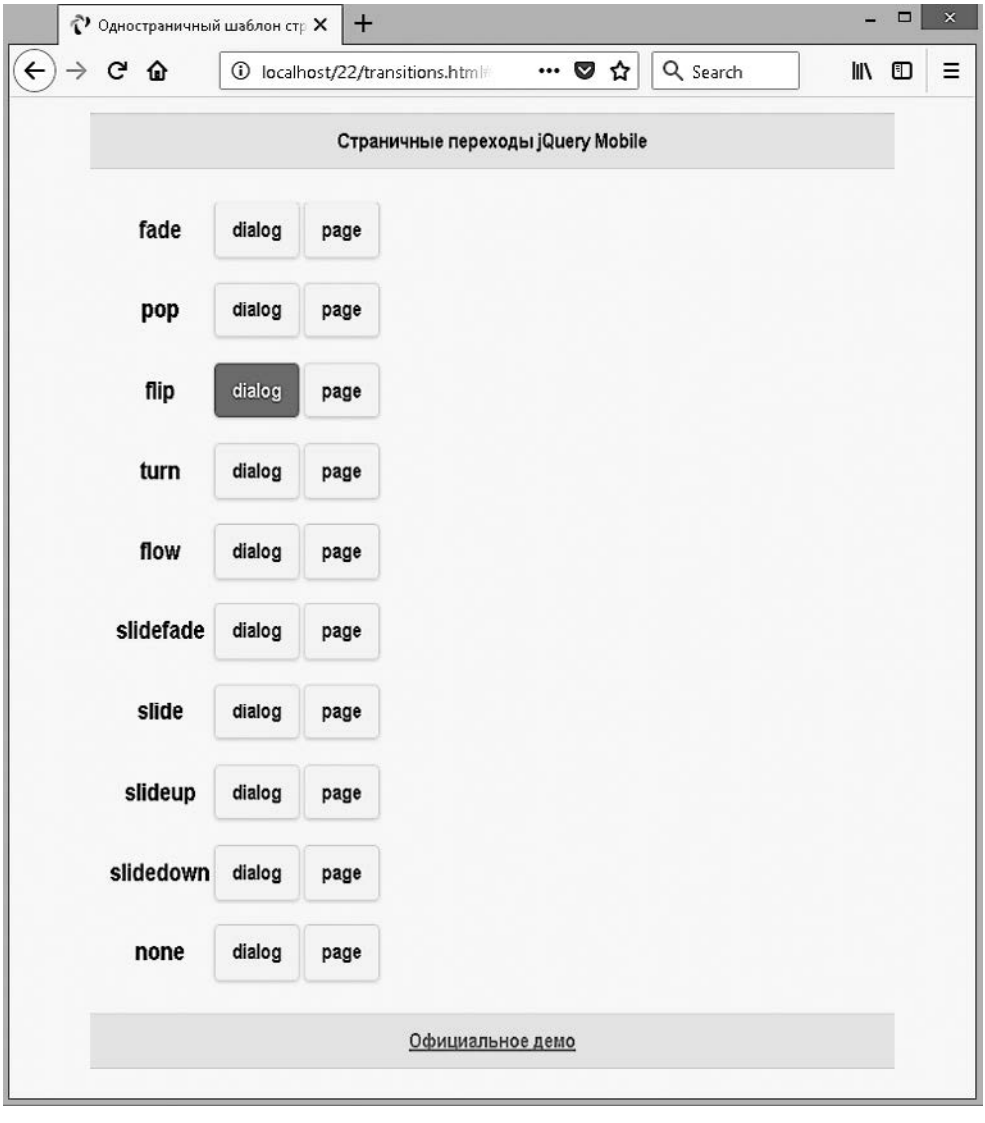

Рис. 22.3. Flip-переход в действии

Стильные кнопки

Простую ссылку совсем нетрудно показать в виде кнопки, не добавляя для этого свой собственный код CSS. Нужно лишь предоставить значение `button` используемому в элементе атрибуту `data-role`:

```
<a data-role="button" href="news.html">Последние новости</a>
```

Можно решать, будет ли эта кнопка шириной во все окно (по умолчанию), подобно `<div>`-элементу, или же показана как составляющая линейной структуры, подобно `<span>`-элементу. Для второго варианта атрибуту `data-inline` нужно присвоить значение `true`:

```
<a data-role="button" data-inline="true" href="news.html">Последние новости</a>
```

Изменить способ отображения можно для любой кнопки, как созданной из ссылки, так и использованной из формы, для чего нужно сделать выбор между скругленными углами (по умолчанию) или прямыми углами и придать кнопке тень (по умолчанию) либо показать ее без тени. Функции, используемые по умолчанию, можно выключить, присвоив соответствующим атрибутам `data-corners` и `data-shadow` значение `false`:

```
<a data-role="button" data-inline="true" data-corners="false"
   data-shadow="false" href="news.html">Последние новости</a>
```

Более того, воспользовавшись атрибутом `data-icon`, можно даже добавить к кнопкам значки:

```
<a data-role="button" data-inline="true" data-icon="home"
   href="home.html">Домашняя страница</a>
```

Существует на выбор более 50 готовых к применению значков. Все они созданы с использованием довольно мощного графического языка под названием Scalable Vector Graphics (SVG), то есть масштабируемая векторная графика, а для устройств, не поддерживающих SVG, используется формат PNG, придающий значкам особую привлекательность на дисплеях с технологией Retina. Доступные значки можно увидеть на демонстрационной странице по адресу http://tinyurl.com/jqmicons.

По умолчанию значки появляются слева от текста кнопки, но можно выбрать их размещение справа, выше или ниже текста или же удалить любой текст, применив, соответственно, значения `right`, `top`, `bottom` или `notext` к атрибуту `data-iconpos`:

```
<a data-role="button" data-inline="true" data-icon="home"
   data-iconpos="right" href="home.html">Домашняя страница</a>
```

Если выбрать вариант без текста, то по умолчанию значок будет показан на круглой кнопке.

И наконец, в завершение этого краткого обзора по стилизации кнопок рассмотрим вариант выбора уменьшенных кнопок (включая и их текст) путем присваивания атрибуту `data-mini` значения `true`:

```
<a data-role="button" data-inline="true" data-icon="home"
   data-mini="true" href="home.html">Домашняя страница</a>
```

В примере 22.3 представлена подборка кнопок, созданных с применением различных стилей (без указания для краткости атрибутов `href`). Результат выполнения кода примера показан на рис. 22.4.

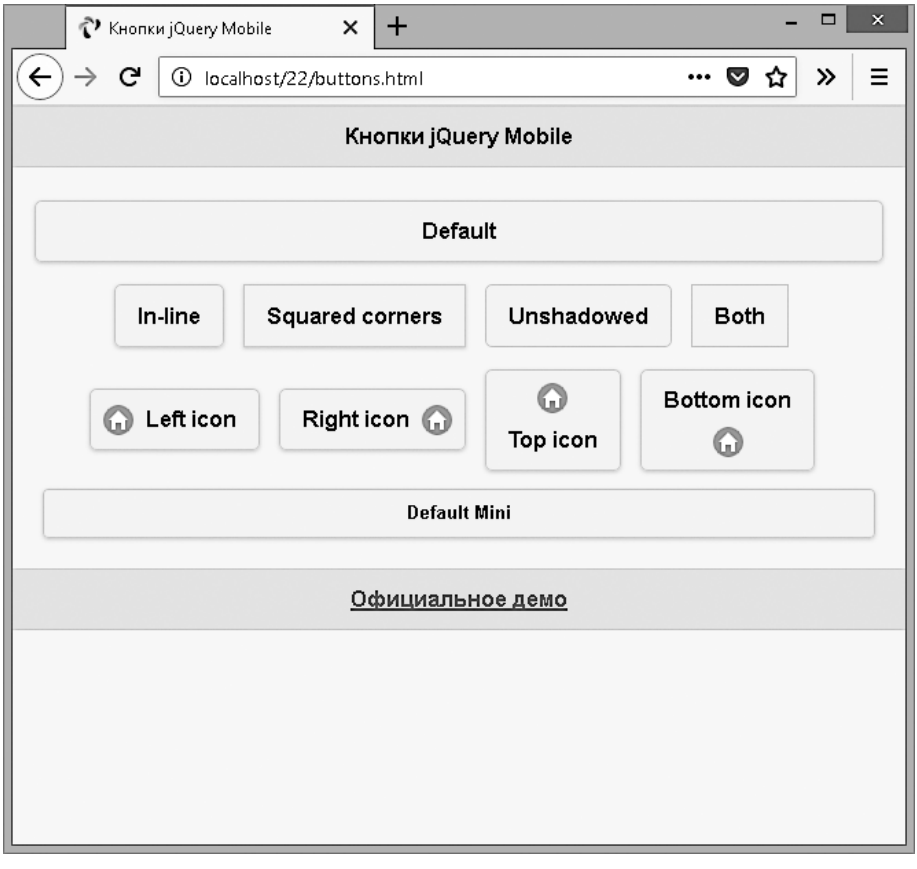

Рис. 22.4. Ассортимент кнопочных стилей

Пример 22.3. Разнообразные кнопочные элементы

```
<a data-role="button">Default</a>
<a data-role="button" data-inline="true">In-line</a>
<a data-role="button" data-inline="true"
                      data-corners="false">Squared corners</a>
<a data-role="button" data-inline="true"
                      data-shadow="false">Unshadowed</a>
<a data-role="button" data-inline="true" data-corners="false"
                      data-shadow="false">Both</a><br>
<a data-role="button" data-inline="true"
                      data-icon="home">Left icon</a>
<a data-role="button" data-inline="true" data-icon="home"
                      data-iconpos="right">Right icon</a>
<a data-role="button" data-inline="true" data-icon="home"
                      data-iconpos="top">Top icon</a>
<a data-role="button" data-inline="true" data-icon="home"
                      data-iconpos="bottom">Bottom icon</a><br>
<a data-role="button" data-mini="true">Default Mini</a>
```

На самом деле выбор стилевого оформления кнопок намного шире, и все необходимые вам подробности можно получить на сайте демонстрации кнопок по адресу http://tinyurl.com/jqmbuttons. А пока вполне сгодится и эта вводная информация.

Обработка списков

Что касается обработки списков, jQuery Mobile предоставляет широкий спектр простых в применении свойств, позволяющих существенно упростить работу. Все эти свойства доступны за счет присваивания значения `listview` атрибуту `data-role`, принадлежащему элементу `<ul>` или `<ol>`.

К примеру, для создания простого неупорядоченного списка можно воспользоваться следующим кодом:

```
<ul data-role="listview">
  <li>Брокколи</li>
  <li>Морковь</li>
  <li>Салат</li>
</ul>
```

Для получения упорядоченного списка нужно просто заменить открывающие и закрывающие теги `<ul>` тегами `<ol>`, и список станет пронумерованным.

Любые ссылки внутри списка автоматически получат значок стрелки, встраиваемый и изображаемый в виде кнопки. Можно также вставить список, смешав его с другим содержимым страницы, для чего нужно присвоить атрибуту `data-inset` значение `true`.

В примере 22.4 показывается, как эти разнообразные свойства работают на практике. Результат выполнения кода представлен на рис. 22.5.

Пример 22.4. Подборка списков

```
<ul data-role="listview">
  <li>Это</li>
  <li>Немаркированный</li>
  <li>Список</li>
</ul><br><br>

<ol data-role="listview">
  <li>Это</li>
  <li>Маркированный</li>
  <li>Список</li>
</ol><br>

<ul data-role="listview" data-inset="true">
  <li>Это</li>
  <li>Вложенный немаркированный</li>
  <li>Список</li>
```

```
</ul>

<ul data-role="listview" data-inset="true">
  <li><a href='#'>Это</a></li>
  <li><a href='#'>Вложенный немаркированный</a></li>
  <li><a href='#'>Список ссылок</a></li>
</ul>
```

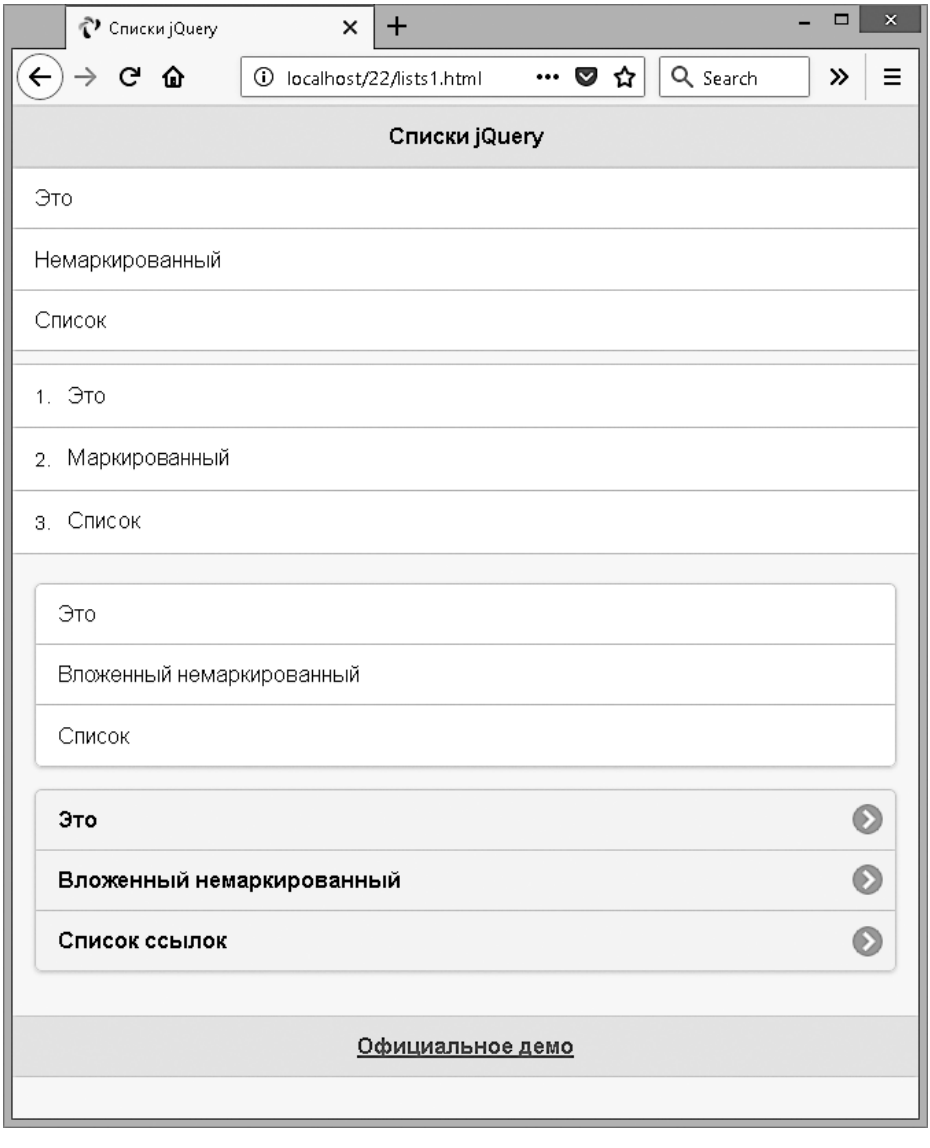

Рис. 22.5. Упорядоченные и неупорядоченные, а также простые и вставленные списки

Фильтруемые списки

Списки можно сделать фильтруемыми, присвоив атрибуту `data-filter` значение `true`, что приведет к появлению над списком поля поиска, по мере ввода в которое аргумента поиска из отображения списка будут автоматически удаляться не соответствующие этому аргументу элементы списка. Можно также присвоить значение `true` аргументу `data-filter-reveal`, в результате чего не будут показаны никакие элементы, пока пользователь не наберет в поле фильтра хотя бы одну букву, а затем будут отображаться только те элементы, которые соответствуют символам, набранным в поле фильтра.

В примере 22.5 показывается использование этих двух типов фильтруемых списков, отличающихся только тем, что к последнему добавлен код `data-filter-reveal="true"`.

Пример 22.5. Фильтруемый список и список с появлением элементов по мере ввода значения фильтра

```
<ul data-role="listview" data-filter="true"
    data-filter-placeholder="Поиск больших кошек..." data-inset="true">
  <li>Гепард</li>
  <li>Пума</li>
  <li>Ягуар</li>
  <li>Леопард</li>
  <li>Лев</li>
  <li>Белый леопард</li>
  <li>Тигр</li>
</ul>

<ul data-role="listview" data-filter="true" data-filter-reveal="true"
    data-filter-placeholder="Поиск больших кошек..." data-inset="true">
  <li>Гепард</li>
  <li>Пума</li>
  <li>Ягуар</li>
  <li>Леопард</li>
  <li>Лев</li>
  <li>Белый леопард</li>
  <li>Тигр</li>
</ul>
```

Обратите внимание на использование атрибута `data-filter-placeholder`, предлагающего приглашение пользователю в пустом поле ввода.

На рис. 22.6 можно увидеть, как первый вариант списка реагирует на ввод буквы в поле поиска, оставляя на экране только те элементы списка, в которых имеется буква «а», и как во втором варианте списка не показывается ни один элемент, поскольку в поле фильтра не введено пока ни одной буквы.

Рис. 22.6. Фильтруемый список и список с появлением элементов по мере ввода значения фильтра

Разделители списков

Чтобы улучшить внешний вид, в списках можно применить разделители, вставляемые вручную или в автоматическом режиме. Вручную разделители списка создаются путем добавления к элементу списка атрибута `data-role` с присвоенным значением `list-divider`. Результат выполнения кода примера 22.6, где применен данный прием, показан на рис. 22.7.

Пример 22.6. Расстановка разделителей списка вручную

```
<ul data-role="listview" data-inset="true">
  <li data-role="list-divider">Большие кошки</li>
  <li>Гепард</li>
  <li>Пума</li>
```

```
  <li>Ягуар</li>
  <li>Лев</li>
  <li>Белый леопард</li>
  <li data-role="list-divider">Большие собаки</li>
  <li>Бладхаунд</li>
  <li>Doberman Pinscher</li>
  <li>Доберман-пинчер</li>
  <li>Мастиф</li>
  <li>Ротвейлер</li>
</ul>
```

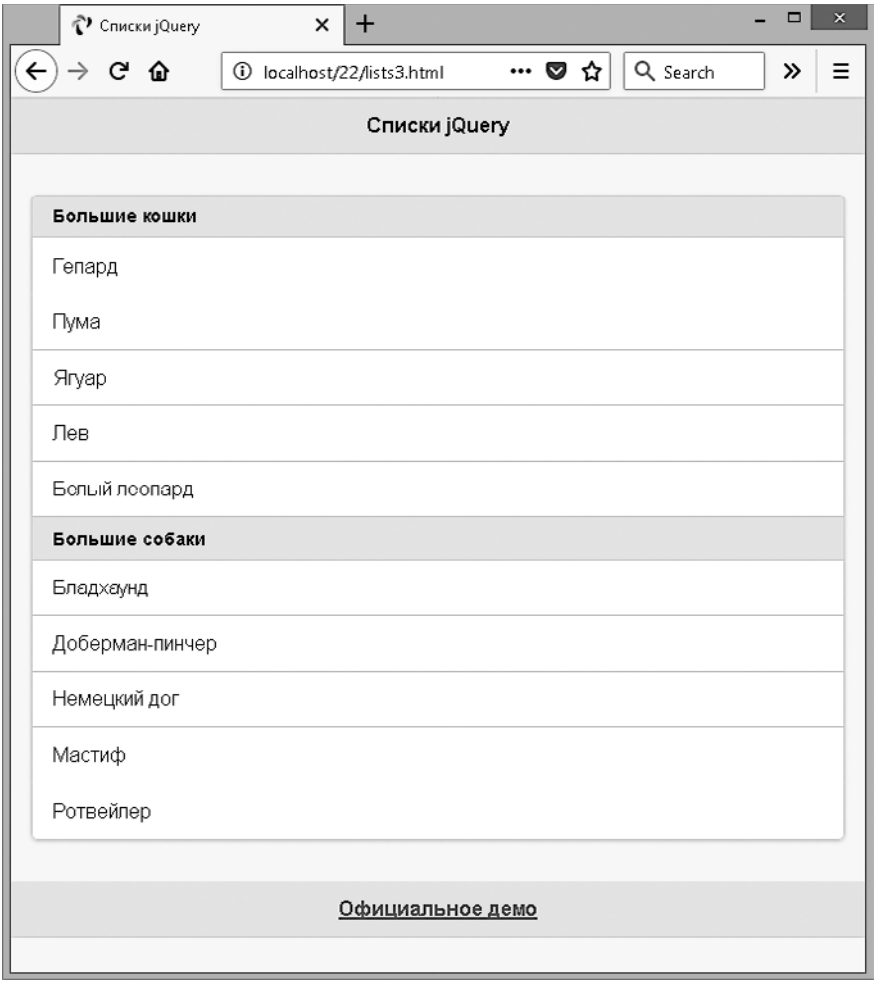

Рис. 22.7. Список, разделенный на категории

Чтобы позволить jQuery Mobile определить расстановку разделителей удобным для вас образом, можно присвоить значение `true` атрибуту `data-autodividers`, соз-

дав код, показанный в примере 22.7, выполнение которого приведет к разделению элементов по алфавиту (рис. 22.8).

Пример 22.7. Использование авторазделителей

```
<ul data-role="listview" data-inset="true" data-autodividers="true">
  <li>Гепард</li>
  <li>Пума</li>
  <li>Ягуар</li>
  <li>Леопард</li>
  <li>Лев</li>
  <li>Белый леопард</li>
  <li>Тигр</li>
</ul>
```

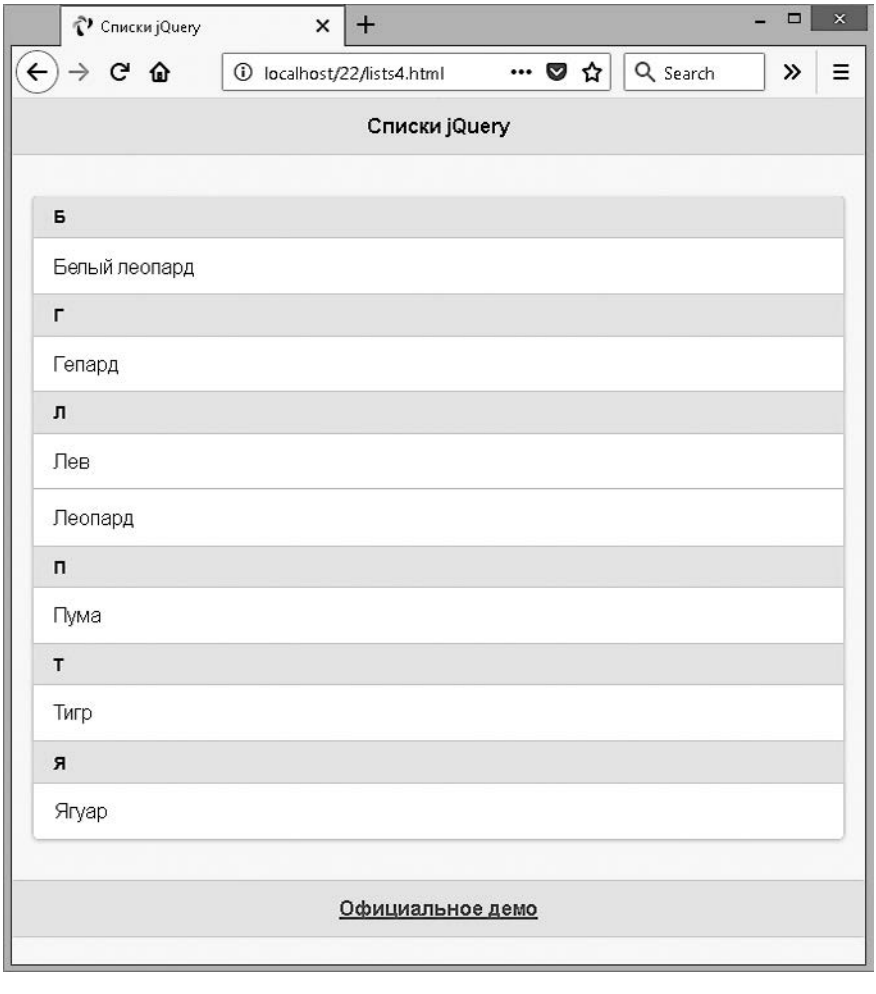

Рис. 22.8. Автоматическое разделение элементов списка по алфавиту

Как и кнопки (см. раздел «Стильные кнопки»), значки можно добавлять к ссылочным элементам списка, используя для этого атрибут `data-icon` со значением, представляющим название показываемого значка:

```
<li data-icon="gear"><a href="settings.html">Настройки</a></li>
```

В этом примере используемая по умолчанию в ссылочном элементе списка правосторонняя угловая скобка будет заменена выбранным вами значком (в данном случае значком шестеренки).

Вдобавок ко всем этим полезным свойствам в элементы списка можно добавлять значки и пиктограммы, и их масштаб будет подбираться для достижения наилучшего внешнего вида. Подробности практических приемов, позволяющих это сделать, и многие другие особенности оформления списков можно найти в официальной документации: http://tinyurl.com/jqmlists.

А что же дальше?

В начале главы уже говорилось, что она предназначена в качестве стартового пособия, позволяющего вам быстро освоиться с применением jQuery Mobile, чтобы можно было легко переделать сайты в веб-приложения, правильно отображаемые на всех устройствах, будь то настольный компьютер или мобильное приспособление.

Именно поэтому были представлены только самые яркие и наиболее важные свойства jQuery Mobile и затронут только самый верхний слой всех возможностей, предоставляемых этой библиотекой. Например, существует огромное множество способов усовершенствования форм и улучшения условий работы с ними на мобильных устройствах. Можно создавать адаптивные таблицы, свертываемое содержимое, вызывать раскрывающиеся окна, проектировать свои собственные темы и делать многое другое.

Вас может заинтересовать изучение возможностей использования jQuery Mobile в сочетании с продуктом фирмы Adobe под названием PhoneGap (https://phonegap.com/) с целью создания автономных приложений для Android и iOS. Там не все так просто, и рассмотрение этого вопроса выходит за рамки данной книги, но вся самая тяжелая работа уже проделана за вас командами разработчиков jQuery и PhoneGap. Дополнительные сведения по разработке PhoneGap-приложений можно найти в руководстве по jQuery Mobile (http://bit.ly/2HHx1zR).

После того как вы освоите все, что изложено в данной главе, и у вас возникнет желание узнать о том, что еще можно сделать с jQuery Mobile, я рекомендую обратиться к официальным демонстрационным страницам и документации, имеющейся на сайте этой библиотеки: http://demos.jquerymobile.com/.

Кроме того, многие из упомянутых здесь свойств применяются в приложении социальной сети, рассматриваемом в главе 27, где приводится довольно близкий к реальности сценарий и предоставляется отличная возможность посмотреть на способы адаптации ваших веб-страниц для работы на мобильных устройствах. Но прежде чем добраться до рассмотрения этих вопросов, в следующих главах мы обратимся к тем полезным свойствам, которые предоставляет нам HTML5.

Вопросы

1. Назовите два основных преимущества и один недостаток использования сети CDN для доставки кода jQuery Mobile в браузер.
2. Каким HTML вы воспользуетесь, чтобы определить страницу содержимого для jQuery Mobile?
3. Каковы три основные части, из которых складывается страница jQuery, и как они обозначаются?
4. Как в HTML-документ поместить более одной страницы jQuery Mobile?
5. Как предотвратить асинхронную загрузку веб-страницы?
6. Как настроить страничный переход по гипертекстовой ссылке на применение эффекта `flip`, а не на используемый по умолчанию эффект `fade`?
7. Как загрузить страницу таким образом, чтобы она отображались в виде диалогового окна, а не в виде веб-страницы?
8. Как можно придать обычной гипертекстовой ссылке внешний вид кнопки?
9. Как добиться линейного показа элемента jQuery Mobile, подобного отображению `<span>`-элемента, не раскрывая его на всю ширину окна, как при использовании элемента `<div>`?
10. Как добавить к кнопке значок?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

23 Введение в HTML5

Язык HTML5 стал существенным шагом вперед в веб-дизайне, разметке и удобстве применения. Он предоставляет простой способ работы с графикой в браузере без обращения к дополнительным модулям, например к Flash, а также предлагает методы вставки аудио и видео в веб-страницы (опять-таки без дополнительных модулей) и сглаживает некоторые досадные несоответствия, вкравшиеся в HTML по мере его развития.

Кроме того, HTML5 включает множество других усовершенствований, например обработку местоположения пользователя, рабочие веб-процессы (web workers), управляющие выполнением фоновых задач, улучшенную обработку форм и доступ к пакетам локального хранилища (значительно превышающим ограниченные возможности cookie).

Но в отношении HTML5 следует отметить один курьезный факт: поскольку этот язык находился в постоянном развитии, присущими ему свойствами различные браузеры обзаводились в разное время. К счастью, все наиболее значимые и популярные добавления, пришедшие с HTML5, теперь уже пользуются поддержкой всех основных браузеров (занимающих долю на рынке, превышающую 1 % или около того, включая такие браузеры, как Chrome, Internet Explorer, Edge, Firefox, Safari и Opera, а также браузеры Android и iOS).

Холст

Первоначально представленный компанией Apple для движка визуализации WebKit (который сам произошел от движка разметки KDE HTML) для своего браузера Safari, элемент *canvas (холст)*, теперь уже реализованный и в iOS, Android, Kindle, Chrome, BlackBerry, Opera и Tizen, позволяет вам рисовать графические элементы на веб-странице независимо от таких дополнительных модулей, как Java или Flash. После стандартизации холст был принят всеми браузерами и сейчас стал популярным средством современной веб-разработки.

Как и другие HTML-элементы, холст — это просто элемент в составе веб-страницы с определенными размерами, внутри которого можно использовать JavaScript для вставки содержимого, в данном случае для рисования графики. Холст создается с помощью тега `<canvas>`, которому также нужно присвоить идентификатор, чтобы в коде JavaScript было понятно, к какому именно холсту идет обращение (ведь на странице может быть несколько холстов).

В примере 23.1 создан элемент `canvas` с идентификатором `mycanvas`, содержащий текст, выводимый только теми браузерами, которые не поддерживают холсты. Далее следует раздел JavaScript, рисующий на холсте японский флаг (рис. 23.1).

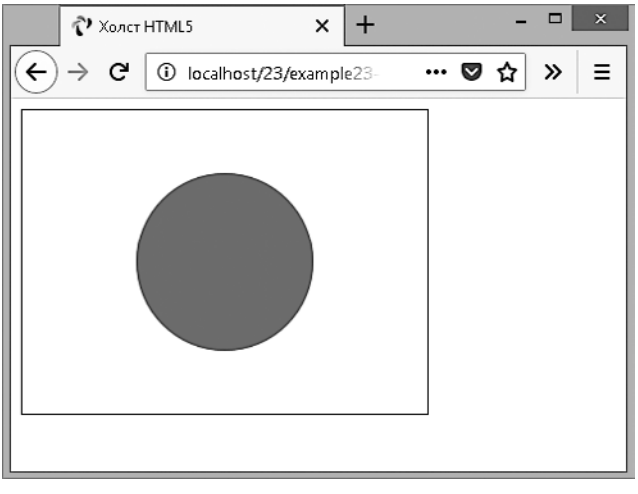

Рис. 23.1. Рисование японского флага на холсте HTML5

Пример 23.1. Использование имеющегося в HTML5 элемента canvas

```
<!DOCTYPE html>
<html>
  <head>
    <title>Холст HTML5</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='240'>
      Это элемент canvas с идентификатором <i>mycanvas</i>
      Этот текст виден только в браузерах, не поддерживающих HTML5
    </canvas>

    <script>
      canvas            = O('mycanvas')
      context           = canvas.getContext('2d')
      context.fillStyle = 'red'
      S(canvas).border  = '1px solid black'

      context.beginPath()
      context.moveTo(160, 120)
```

```
      context.arc(160, 120, 70, 0, Math.PI * 2, false)
      context.closePath()
      context.fill()
    </script>
  </body>
</html>
```

Рассматривать подробности происходящего на данный момент не имеет смысла, поскольку все будет объяснено в главе 24, но уже сейчас вы должны понять, что использовать холст совсем не трудно, хотя для этого потребуется изучить несколько новых функций JavaScript. Чтобы сохранить ясность и компактность кода, в этом примере при рисовании используется набор функций `OSC.js` из главы 20.

Геолокация

При использовании *геолокации* ваш браузер может вернуть веб-серверу информацию о вашем местонахождении. Эта информация может поступать из микросхемы GPS, имеющейся в используемом компьютере или мобильном устройстве, с вашего IP-адреса или путем анализа ближайших точек доступа к Wi-Fi. В целях безопасности пользователь всегда контролирует ситуацию и может отказать в предоставлении этой информации на разовой основе или включить настройки либо на постоянное блокирование, либо на предоставление доступа к этим данным с одного или со всех сайтов.

Эта технология имеет множество вариантов применения, включая пошаговую навигацию, предоставление местных карт, уведомление о ближайших ресторанах, о точках доступа Wi-Fi или о других местах, уведомление о тех друзьях, которые находятся рядом, указание направления на ближайшую заправку и многое другое.

В примере 23.2 будет отображена карта Google с учетом местоположения пользователя при условии, что браузер поддерживает геолокацию и пользователь предоставил доступ к своему местоположению (как показано на рис. 23.2). В противном случае будет выведено сообщение об ошибке.

Пример 23.2. Вывод карты с учетом местоположения пользователя

```
<!DOCTYPE html>
<html>
  <head>
    <title>Пример геолокации</title>
  </head>
  <body>
   <script>
      if (typeof navigator.geolocation == 'undefined')
        alert("Геолокация не поддерживается.")
     else
        navigator.geolocation.getCurrentPosition(granted, denied)

      function granted(position)
      {
       var lat      = position.coords.latitude
```

```
        var long   = position.coords.longitude

        alert("Разрешение дано. Ваше местоположение:\n\n"
         + lat + ", " + lon +
         "\n\nЩелкните на кнопке 'OK' для загрузки Google Maps
         с вашим местоположением ")

        window.location.replace("https://www.google.com/maps/@"
          + lat + "," + lon + ",14z")
      }

      function denied(error)
      {
        var message

        switch(error.code)
        {
          case 1: message = 'Доступ запрещен'; break;
          case 2: message = 'Позиция недоступна'; break;
          case 3: message = 'Время ожидания операции истекло'; break;
          case 4: message = 'Неизвестная ошибка'; break;
        }

        alert("Ошибка геолокации: " + message)
      }
    </script>
  </body>
</html>
```

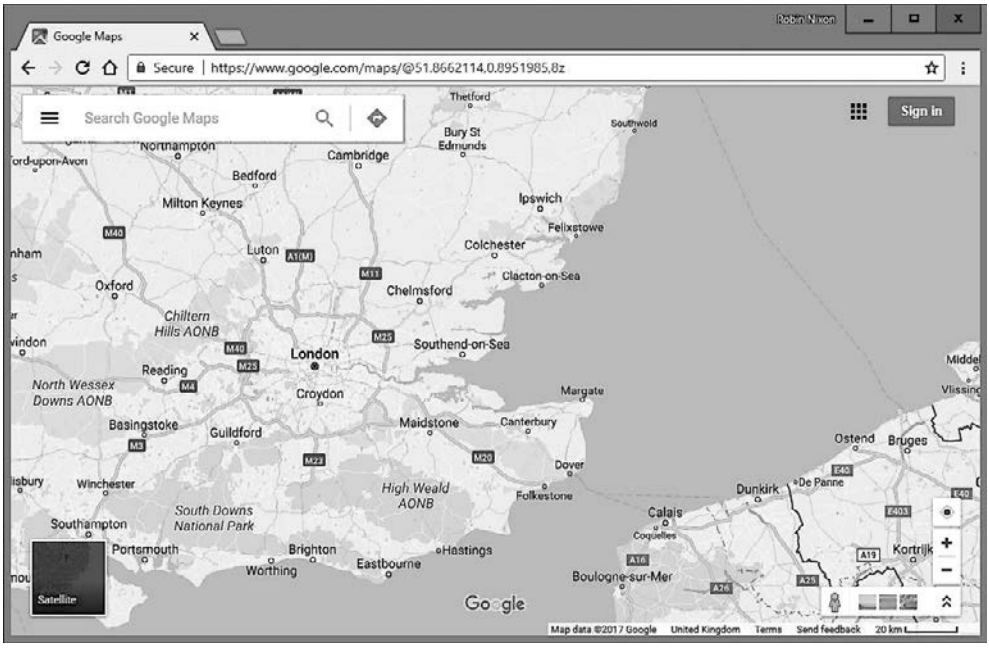

Рис. 23.2. Местоположение пользователя использовалось для отображения карты

Здесь опять нет смысла давать описание того, как все это работает, поскольку подробное рассмотрение данного вопроса предстоит в главе 26. А сейчас этот пример нужен, чтобы показать вам, насколько просто можно управлять геолокацией.

Аудио и видео

Еще одним существенным дополнением, появившимся в HTML5, стала поддержка аудио и видео в самом браузере. Хотя проигрывание этого типа аудиовизуальных данных может представлять некоторые сложности из-за разнообразия типов кодирования и лицензий, элементы `<audio>` и `<video>` дают всю необходимую гибкость для отображения доступных вам типов аудиовизуальной информации.

В примере 23.3 один и тот же видеофайл был закодирован в различных форматах, чтобы были учтены возможности всех основных браузеров. Браузеры просто выбирают первый из опознанных ими типов и проигрывают его (рис. 23.3).

Рис. 23.3. Показ видео с помощью HTML5

Пример 23.3. Проигрывание видео с помощью HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <title>Видео в HTML5</title>
  </head>
  <body>
    <video width='560' height='320' controls>
      <source src='movie.mp4'  type='video/mp4'>
```

```
      <source src='movie.webm' type='video/webm'>
      <source src='movie.ogv'  type='video/ogg'>
    </video>
  </body>
</html>
```

Вопрос о том, насколько просто аудио вставляется в веб-страницу, будет рассмотрен в главе 25.

Формы

Как вы уже видели в главе 11, формы в HTML5 находятся в процессе усовершенствования, при этом поддержка новых свойств во всех браузерах носит фрагментарный характер. Все, что сегодня можно свободно использовать, было подробно рассмотрено в главе 11.

Локальное хранилище

При использовании локального хранилища объем и сложность данных, сохраняемых на локальном устройстве, существенно возрастают по сравнению со скудными объемами, предоставлявшимися cookie. Тем самым открываются возможности использования веб-приложений для работы с документами в автономном режиме с их последующей синхронизацией с веб-сервером, когда интернет-подключение станет доступным. Кроме того, открываются перспективы локального сохранения небольших баз данных для доступа к ним с помощью WebSQL, возможно, с целью сохранения копии вашей музыкальной коллекции или всей вашей персональной статистики, к примеру, в виде части плана диеты или похудания. Как задействовать большинство этих возможностей в ваших веб-проектах, будет показано в главе 26.

Рабочие веб-процессы

Возможность запускать с помощью JavaScript в фоновом режиме приложения, управляемые прерываниями, существует уже многие годы, но реализуется только лишь посредством довольно грубого и неэффективного процесса. Было бы намного разумнее позволить технологии, положенной в основу браузера, запускать от вашего имени фоновые задачи, которые она смогла бы выполнять гораздо быстрее, чем при ваших постоянных прерываниях работы с целью проверки текущего состояния дел.

Вместо этого можно все настроить в виде *рабочих веб-процессов* и передать ваш код браузеру, который затем его запустит. Когда произойдет что-либо значимое, ваш код просто должен будет уведомить браузер, который затем отрапортует все это назад вашему основному коду. В то же самое время ваша веб-страница может пребывать в бездействии или выполнять ряд других задач и при этом напрочь забыть о фоновой задаче, пока та сама о себе не напомнит.

В главе 26 будет показано, как можно воспользоваться рабочими веб-процессами для создания простых часов и вычисления простых чисел.

Микроданные

В главе 26 также будет показано, как можно пометить свой код с помощью *микроданных*, чтобы он стал абсолютно понятен любому браузеру или другой технологии, которой нужно получить к нему доступ.

Важность микроданных, несомненно, будет возрастать и для оптимизации поисковых машин, поэтому вам нужно приступить к их внедрению или по крайней мере понять, какую информацию они могут предоставить о ваших сайтах.

Как видите, в HTML5 появилось немало новых свойств, которые многими ожидались на протяжении довольно долгого времени и вот наконец-то стали доступны. В следующих нескольких главах эти свойства будут рассмотрены во всех замечательных подробностях, чтобы вы могли ими воспользоваться и улучшить качество своих сайтов в самые кратчайшие сроки.

Вопросы

1. Какие новые элементы, появившиеся в HTML5, позволяют рисовать графические изображения на веб-странице?
2. Какой язык программирования требуется для доступа ко многим улучшенным свойствам HTML5?
3. Какими тегами вы воспользуетесь для внедрения аудио и видео в веб-страницу?
4. Какое новое свойство, предлагающее более высокие возможности по сравнению с cookie, введено в HTML5?
5. Какая имеющаяся в HTML5 технология поддерживает запуск фоновых задач на JavaScript?

Ответы на эти вопросы можно найти в приложении A, в соответствующем разделе.

24 Холсты в HTML5

Хотя новым веб-технологиям в *HTML5* даются обобщенные названия, все они не являются простыми HTML-тегами и свойствами. Это также относится к элементам `canvas` (холст). Да, холст создается с помощью тега `<canvas>`, и ему могут быть предоставлены ширина и высота, и их можно немного изменять с помощью CSS, но для записи в холст (или для чтения из него) нужно использовать JavaScript.

Хорошо, что для этого нужны лишь минимальные знания JavaScript и реализовать задуманное весьма просто. Кроме того, в главе 20 я уже предоставлял вам набор из трех готовых к использованию функций (в файле `OSC.js`), с помощью которых доступ к таким объектам, как холсты, существенно упрощается. Итак, углубимся в эту тему и приступим к использованию нового тега `<canvas>`.

Создание холста и доступ к нему

В главе 23 показывалось, как нарисовать простую окружность для отображения японского флага (пример 24.1). Теперь посмотрим, что именно происходит.

Пример 24.1. Отображение японского флага с помощью холста

```
<!DOCTYPE html>
<html>
  <head>
    <title>Холст HTML5</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='240'>
      Это элемент canvas с идентификатором <i>mycanvas</i>
      Этот текст виден только в браузерах, не поддерживающих HTML5 </canvas>
    <script>
      canvas            = O('mycanvas')
      context           = canvas.getContext('2d')
```

```
      context.fillStyle = 'red'
      S(canvas).border  = '1px solid black'

      context.beginPath() context.moveTo(160, 120)
      context.arc(160, 120, 70, 0, Math.PI * 2, false)
      context.closePath()
      context.fill()
    </script>
  </body>
</html>
```

Сначала помещается объявление `<! DOCTYPE html>`, чтобы сообщить браузеру о том, что документ будет использовать HTML5. После этого выводится заголовок и загружаются три функции, находящиеся в файле `OSC.js`.

В теле документа определяется элемент `canvas`, которому дается идентификатор `mycanvas`, и задаются ширина и высота 320 × 240 пикселов. Текст холста, как уже говорилось в предыдущей главе, не будет появляться в браузерах, поддерживающих элемент холста, он будет виден в окнах устаревших браузеров, не поддерживающих холсты.

Затем следует раздел JavaScript, в котором задаются стили и прорисовки холста. Начало создания объекта `canvas` обозначено вызовом функции `O` в отношении элемента `canvas`. Как вы помните, в результате этого вызывается функция `document.getElementById`, и, следовательно, это лишь более короткий способ ссылки на элемент.

Все это вы уже видели раньше, а затем следует нечто новое:

```
context = canvas.getContext('2d')
```

Команда вызывает метод `getContext` нового, только что созданного объекта `canvas`, запрашивая двумерный доступ к холсту путем передачи значения `2d`.

Если нужно отобразить на холсте трехмерную графику, можно либо провести математические вычисления самостоятельно и «подделать» ее в режиме отображения двумерной графики, либо воспользоваться библиотекой WebGL (основанной на технологии OpenGL ES), и в этом случае нужно будет для нее создать переменную context путем вызова метода canvas.getContext('webgl'). Для более расширенного рассмотрения данного вопроса у нас здесь нет места, но отличное руководство по WebGL можно найти на веб-сайте по адресу http://www.webgltutorials.org. Кроме того, можно обратиться к JavaScript-библиотеке 3D-функций Three.js (https://threejs.org), в которой также используется WebGL.

Вооружившись данным содержимым в контексте объекта, мы начнем выдавать последующие команды рисования, установив для свойства `fillStyle` этого контекста значение `red`:

```
context.fillStyle = 'red'
```

Затем вызывается функция S для установки свойству border объекта canvas значения однопиксельной сплошной черной линии для очертания изображения флага:

```
S(canvas).border = '1px solid black'
```

После того как все будет готово, открывается путь контекста и позиция рисования перемещается к точке с координатами (160; 120):

```
context.beginPath()
context.moveTo(160, 120)
```

Затем рисуется дуга с центром, имеющим данные координаты. Ее радиус — 70 пикселов, она начинается с угла 0° (что является правым краем окружности, если смотреть прямо на нее) и продолжается по всей окружности в радианах, определяемых значением 2ϖ:

```
context.arc(160, 120, 70, 0, Math.PI * 2, false)
```

Последнее значение false служит признаком направления рисования дуги по часовой стрелке; а значение true было бы признаком рисования, осуществляемого против часовой стрелки.

И наконец, мы закрываем путь и осуществляем его заливку, используя предопределенное значение в свойстве fillStyle, которое мы несколькими строками ранее установили в red:

```
context.closePath() context.fill()
```

Результат загрузки этого документа в браузер показан в предыдущей главе на рис. 23.1.

Функция toDataURL

При создании изображения на холсте иногда требуется получить его копию, возможно, для его повторения на веб-странице, для сохранения в локальном хранилище или для выкладки на веб-сервер. В частности, удобство копирования предопределено невозможностью сохранения пользователем изображения с холста путем его перетаскивания.

Чтобы проиллюстрировать, как это делается, к предыдущему примеру было добавлено несколько строк кода (выделенных полужирным шрифтом в примере 24.2). Здесь создается новый элемент <img> с идентификатором myimage, ему дается сплошная черная граница, а затем в элемент <img> копируется изображение холста (рис. 24.1).

Пример 24.2. Копирование изображения холста

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>Копирование холста </title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='240'>
      Это элемент canvas с идентификатором <i>mycanvas</i>
      Этот текст виден только в браузерах, не поддерживающих HTML5
    </canvas>

    <img id='myimage'>

    <script>
      canvas            = O('mycanvas')
      context           = canvas.getContext('2d')
      context.fillStyle = 'red'
      S(canvas).border  = '1px solid black'

      context.beginPath() context.moveTo(160, 120)
      context.arc(160, 120, 70, 0, Math.PI * 2, false)
      context.closePath()
      context.fill()
      S('myimage').border = '1px solid black'
      O('myimage').src    = canvas.toDataURL()
    </script>
  </body>
</html>
```

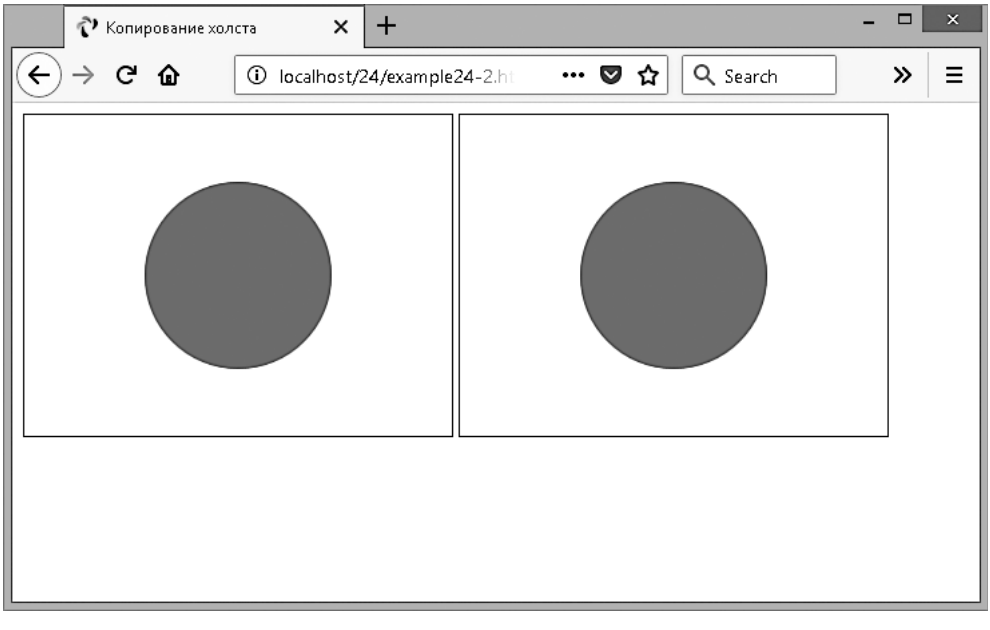

Рис. 24.1. Изображение справа скопировано с холста, показанного слева

Если вы проверите этот код в работе, то заметите, что при отсутствии возможности перетаскивания изображения холста, показанного слева, правостороннее изображение можно перетаскивать. Его также можно сохранить в локальном хранилище или выложить на веб-сервер, используя соответствующий код JavaScript (а с серверной стороны — код PHP).

Указание типа изображения

При создании изображения из холста можно указать тип того изображения, которое требуется вывести за пределы холста, выбирая из JPEG (файлы с расширениями `.jpg` или `.jpeg`) или PNG (файлы с расширениями `.png`). По умолчанию используется тип PNG (`'image/png'`), но если отдается предпочтение JPEG, нужно изменить вызов `toDataURL`. Одновременно с этим можно указать степень сжатия в диапазоне от `0` (для самого низкого качества) до `1` (для самого высокого качества). В следующем коде используется степень сжатия `0.4`, позволяющая сгенерировать подходящее изображение, которое будет неплохо выглядеть при небольшом размере файла:

```
O('myimage').src = canvas.toDataURL('image/jpeg', 0.4)
```

Следует учесть, что метод toDataURL применяется к объекту холста, а не к любому контексту, созданному из этого объекта.

Теперь, когда вы знаете, как создаются изображения холста с их последующим копированием или использованием в каком-нибудь другом качестве, настало время посмотреть на отдельные доступные команды рисования, начиная с команд, относящихся к прямоугольникам.

Метод fillRect

Существует два разных метода, вызываемых для рисования прямоугольников, первым из которых является `fillRect`. Чтобы им воспользоваться, нужно просто предоставить левую верхнюю координату вашего прямоугольника, а за ней ширину и высоту в пикселах:

```
context.fillRect(20, 20, 600, 200)
```

По умолчанию прямоугольник будет залит черным цветом, но можно воспользоваться любым другим оттенком, выдав команду, похожую на следующую, где аргументом может послужить любой приемлемый CSS-цвет, указанный по названию или по значению:

```
context.fillStyle = 'blue'
```

Метод clearRect

Можно также нарисовать прямоугольник, в котором все его цветовые значения (красная, зеленая, синяя составляющие и альфа-прозрачность) установлены в `0`, как в следующем примере кода, где используется тот же порядок координат и аргументы ширины и высоты:

```
context.clearRect(40, 40, 560, 160)
```

После применения метода `clearRect` новый прозрачный прямоугольник удалит все цвета из закрываемой им области, оставив только любой исходный CSS-цвет, который был применен к элементу холста.

Метод strokeRect

Когда нужен лишь контур прямоугольника, можно обратиться к команде, похожей на показанную ниже, в которой будет использована установка по умолчанию на черный цвет или же выбранный цвет обводки:

```
context.strokeRect(60, 60, 520, 120)
```

Для смены цвета можно сначала выдать команду, подобную следующей, предоставив ей в качестве аргумента любой приемлемый CSS-цвет:

```
context.strokeStyle = 'green'
```

Сочетание всех этих команд

В примере 24.3 предыдущие команды рисования прямоугольников объединены для вывода изображения, приведенного на рис. 24.2.

Пример 24.3. Рисование нескольких прямоугольников

```
<!DOCTYPE html>
<html>
  <head>
    <title>Рисование прямоугольников</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='640' height='240'></canvas>

    <script>
      canvas               = O('mycanvas')
      context              = canvas.getContext('2d')
      S(canvas).background = 'lightblue'
      context.fillStyle    = 'blue'
      context.strokeStyle  = 'green'

      context.fillRect(  20, 20, 600, 200)
      context.clearRect( 40, 40, 560, 160)
      context.strokeRect(60, 60, 520, 120)
```

```
    </script>
  </body>
</html>
```

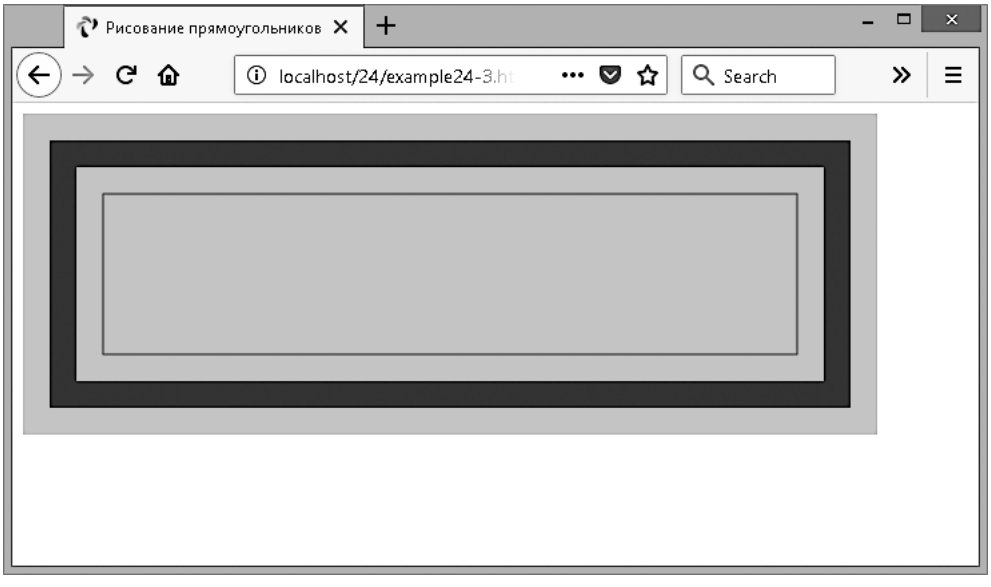

Рис. 24.2. Рисование концентрических прямоугольников

Чуть позже в данной главе будет показано, как можно внести последующие изменения в вывод, изменив тип и ширину обводки, но сначала обратимся к изменению заливки с применением градиентов (которые были представлены в разделе «Градиенты» главы 18).

Метод createLinearGradient

Есть два разных способа применения градиента для заливки, но проще всего это сделать с помощью метода `createLinearGradient`. Нужно указать начальные и конечные координаты *x* и *y* относительно холста (а не заливаемого объекта). Это позволяет добиться большей утонченности.

Например, можно указать, что градиент начинается с самого левого края и заканчивается в самом правом краю холста, но применяется только в области, определенной командой заливки, что и сделано в примере 24.4.

Пример 24.4. Применение градиентной заливки

```
gradient = context.createLinearGradient(0, 80, 640,80)
gradient.addColorStop(0, 'white')
gradient.addColorStop(1, 'black')
context.fillStyle = gradient
context.fillRect(80, 80, 480,80)
```

Для краткости и ясности в этом и многих последующих примерах показаны только значимые строки кода. Полные примеры с окружающим HTML, настройкой и другими разделами кода доступны для свободной загрузки с сайта http://lpmj.net.

В примере 24.4 создается градиентная заливка объекта по имени `gradient`, для чего используется метод `createLinearGradient` объекта `context`. Стартовая позиция (0, 80) находится на полпути вниз от левого края холста, а конечная позиция (640, 80) находится на полпути вниз от правого края холста.

Для создания своего собственного градиента нужно определить направление его распространения, а затем указать две точки, представляющие стартовую и конечную позиции. Независимо от того, какие значения будут предоставлены для этих точек, градиент будет выполнять плавный переход в указанном направлении, даже если точки находятся за пределами области заливки.

После этого предоставляются две цветовые опорные точки, а именно, самым первым цветом градиента указывается белый, а самым последним — черный. Затем градиент будет плавно переходить между этими цветами по всему холсту слева направо.

После того как объект `gradient` готов, он применяется к свойству `fillStyle` объекта `context`, чтобы им мог воспользоваться завершающий вызов метода `fillRect`. В этом вызове заливка применяется только к центральной прямоугольной области холста, поэтому, хотя градиент простирается с самого левого края к самому правому краю холста, отображаемая часть простирается лишь с 80 пикселов вовнутрь и вниз от верхнего левого угла на ширину 480 и глубину 80 пикселов. Результат (при добавлении кода к коду предыдущего примера) показан на рис. 24.3.

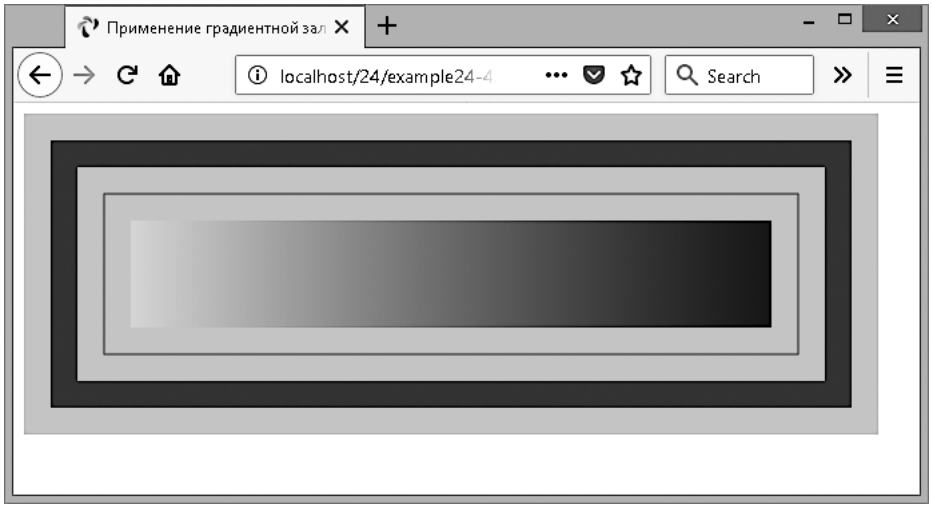

Рис. 24.3. У центрального прямоугольника имеется горизонтальная градиентная заливка

Указывая для градиента различные стартовые и конечные позиции, можно придать ему наклон для распространения в любом направлении, как показано в примере 24.5 и отображено на рис. 24.4.

Пример 24.5. Различные градиенты под разными углами и с разными цветами

```
gradient = context.createLinearGradient(0, 0, 160, 0)
gradient.addColorStop(0, 'white')
gradient.addColorStop(1, 'black')
context.fillStyle = gradient
context.fillRect(20, 20, 135, 200)

gradient = context.createLinearGradient(0, 0, 0, 240)
gradient.addColorStop(0, 'yellow')
gradient.addColorStop(1, 'red')
context.fillStyle = gradient
context.fillRect(175, 20, 135, 200)

gradient = context.createLinearGradient(320, 0, 480, 240)
gradient.addColorStop(0, 'green')
gradient.addColorStop(1, 'purple')
context.fillStyle = gradient
context.fillRect(330, 20, 135, 200)

gradient = context.createLinearGradient(480, 240, 640, 0)
gradient.addColorStop(0, 'orange')
gradient.addColorStop(1, 'magenta')
context.fillStyle = gradient
context.fillRect(485, 20, 135, 200)
```

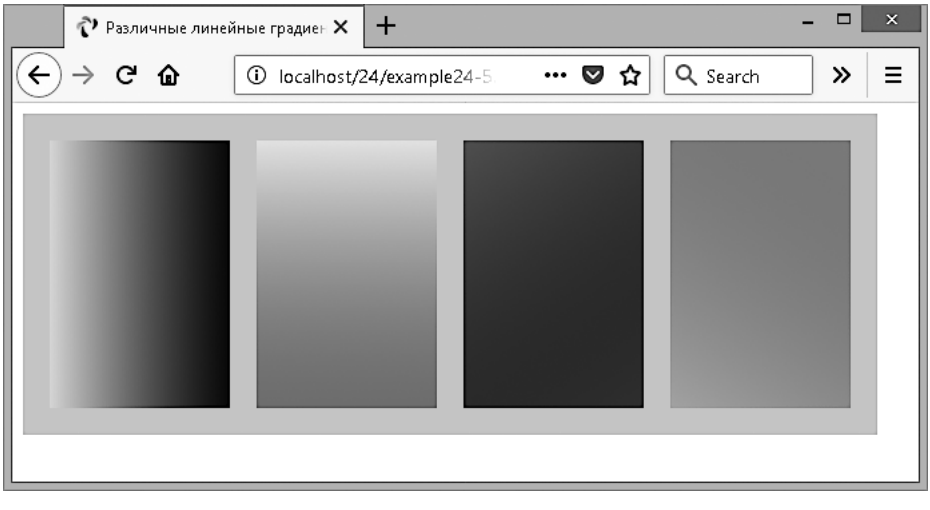

Рис. 24.4. Подборка различных линейных градиентов

В этом примере я решил поместить градиенты непосредственно в верхнюю часть заливаемых областей, чтобы более ясно показать максимальные отклонения по цвету с начала до конца.

Метод addColorStop в подробностях

В градиенте можно использовать сколько угодно цветовых опорных точек, а не только стартовую и конечную, которые до сих пор использовались в примерах. Это позволяет дать четкое описание почти что любому типу представляемого вами градиентного эффекта. Для этого нужно указать процент градиента, занимаемый каждым цветом. Для распределения по градиенту стартовой позиции в формате числа с плавающей точкой берется диапазон между `0` и `1`. Конечную позицию цвета вводить не нужно, поскольку она выводится из стартовой позиции следующей цветовой опорной точки или же градиент заканчивается, если позиция является последней указанной вами.

В предыдущих примерах были выбраны только два значения, стартовое и конечное, но для создания эффекта радуги можно настроить цветовые опорные точки, как показано в примере 24.6 (результат изображен на рис. 24.5).

Пример 24.6. Добавление нескольких цветовых опорных точек

```
gradient.addColorStop(0.00, 'red')
gradient.addColorStop(0.14, 'orange')
gradient.addColorStop(0.28, 'yellow')
gradient.addColorStop(0.42, 'green')
gradient.addColorStop(0.56, 'blue')
gradient.addColorStop(0.70, 'indigo')
gradient.addColorStop(0.84, 'violet')
```

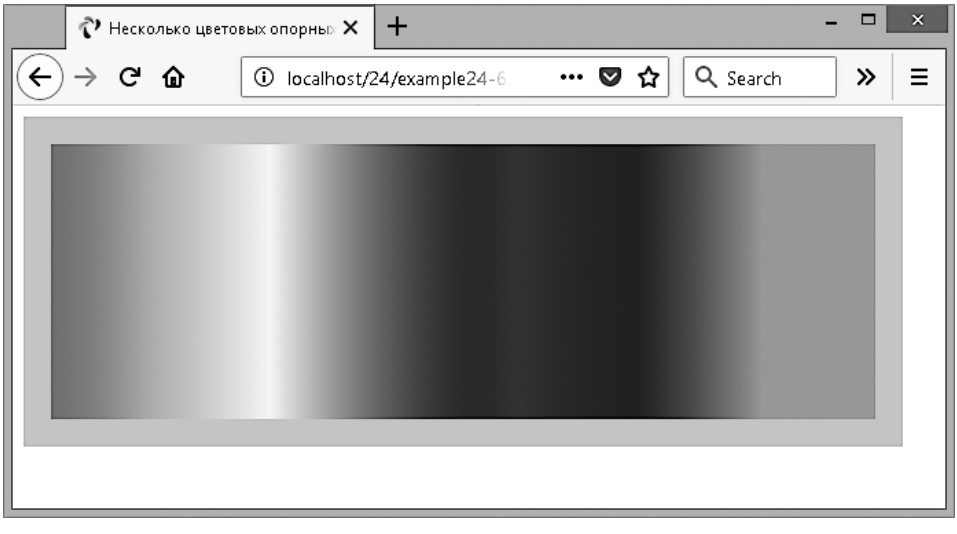

Рис. 24.5. Эффект радуги, полученный с указанием семи цветовых опорных точек

В примере 24.6 все цвета расположены примерно на одинаковом расстоянии друг от друга (каждому цвету выделено 14 % градиента, а последнему — 16 %),

но придерживаться этого не обязательно. Какие-то цвета можно прижать друг к другу, а каким-то дать больше простора. Сколько цветов использовать и где в градиенте они должны стартовать и финишировать, отдается полностью на ваше усмотрение.

Метод createRadialGradient

В HTML вы не ограничены созданием только линейных градиентов, на холсте можно создавать и радиальные градиенты, хоть это и немного сложнее сделать.

Для этого нужно передать расположение центра в виде пары координат x и y, а также указать радиус в пикселах. Они будут использованы в качестве стартовой позиции и внешней окружности соответственно. Затем передается еще один набор координат и радиус для указания окончания градиента.

Например, для создания градиента, который просто начинается в центре окружности, а затем распространяется наружу, можно выдать команду, показанную в примере 24.7 (результат изображен на рис. 24.6). Координаты стартовой и конечной позиций те же самые, но радиус задан нулевым для стартовой позиции и охватывает весь градиент для конечной позиции.

Пример 24.7. Создание радиального градиента

```
gradient = context.createRadialGradient (320, 120, 0, 320, 120, 320)
```

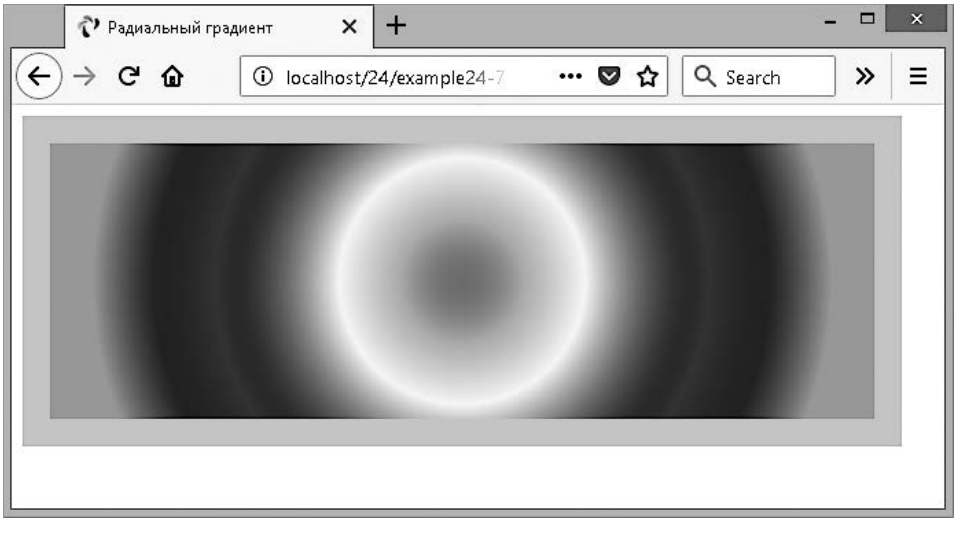

Рис. 24.6. Радиальный градиент, исходящий из центра

Или можно проявить фантазию и переместить стартовую и конечную позиции радиального градиента, как в примере 24.8 (результат изображен на рис. 24.7), где

градиент начинается с центра с координатами (0; 120) и радиусом 0 пикселов, а заканчивается в центре с координатами (480; 120) и радиусом 480 пикселов.

Пример 24.8. Растягивание радиального градиента

```
gradient = context.createRadialGradient(0, 120, 0, 480, 120, 480)
```

Манипулируя цифрами, задаваемыми этому методу, можно создать широкий диапазон необычных и удивительных эффектов, поэкспериментировать с которыми на основе предоставленных примеров вам предлагается самостоятельно.

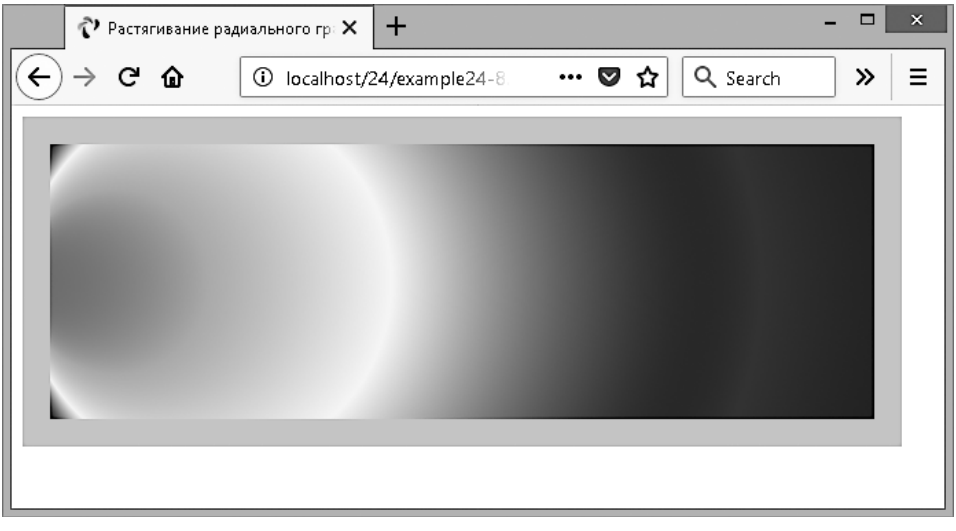

Рис. 24.7. Растянутый радиальный градиент

Использование узоров для заливки

Точно так же, как при заливке градиентом, можно в качестве узора заливки воспользоваться изображением. Это может быть изображение, находящееся где-нибудь в текущем документе, или даже изображение, созданное из холста посредством метода `toDataURL` (рассмотренного ранее в данной главе).

В примере 24.9 в новый объект изображения `image` загружается изображение 100 × 100 пикселов (символ инь-ян). Следующая инструкция прикрепляет к событию `onload` функцию, создающую повторяющийся узор для свойства `fillStyle` объекта `context`. Затем узор, как показано на рис. 24.8, используется для заливки области 600 × 200 пикселов.

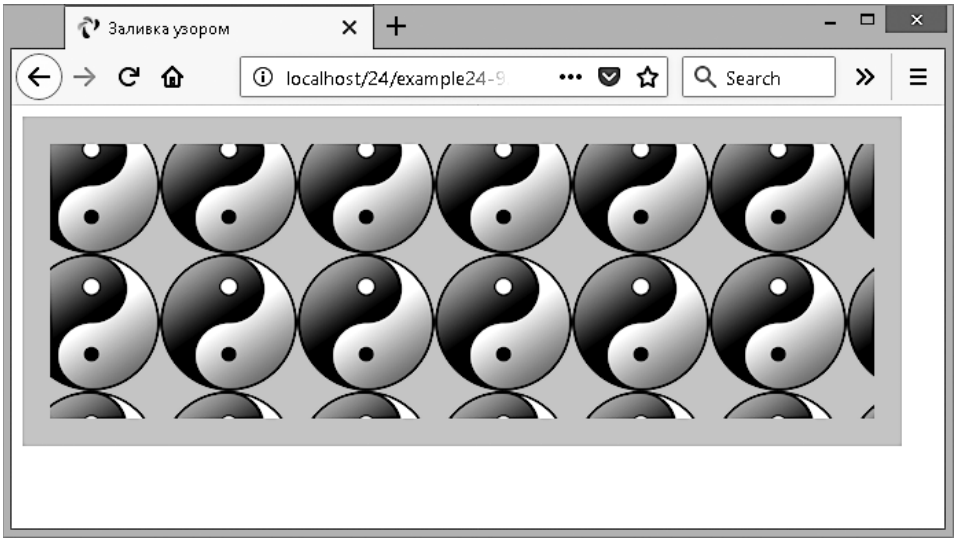

Рис. 24.8. Создание мозаики из изображения путем применения его в качестве узора заливки

Пример 24.9. Использование изображения для узора заливки

```
image = new image()
image.src = 'image.png'

image.onload = function()
{
  pattern           = context.createPattern(image, 'repeat')
  context.fillStyle = pattern
  context.fillRect(20, 20, 600, 200)
}
```

Мы создали узор, используя метод `createPattern`, который также поддерживает узоры без повторений или такие узоры, которые просто повторяются по оси *X* или *Y*. Это достигается передачей методу одного из следующих значений в качестве второго аргумента после используемого изображения:

- `repeat` — повторение изображения как по вертикали, так и по горизонтали;
- `repeat-x` — повторение изображения по горизонтали;
- `repeat-y` — повторение изображения по вертикали;
- `no-repeat` — отказ от повторения изображения.

Узор заливки применяется по отношению ко всей области холста, поэтому, когда команда заливки настроена на применение только к меньшей по размеру области внутри холста, изображения выводятся обрезанными сверху и слева.

Если бы в данном примере не использовалось событие onload, а вместо этого код просто выполнялся бы сразу, как только он попадался в сценарии, изображение ко времени вывода на экран веб-страницы могло быть еще не загруженным и могло не появиться на экране. Прикрепление к этому событию гарантирует доступность изображения для использования на холсте, поскольку событие наступает только в результате успешной загрузки изображения.

Запись текста на холсте

От набора графических функций вполне можно ожидать полной поддержки записи текста на холст с применением разнообразных методов задания шрифтов, выравниваний и заливок. Но зачем записывать текст на холст, когда сейчас уже есть превосходная поддержка веб-шрифтов в CSS?

Предположим, нужно отобразить схему или таблицу с графическими элементами. И вам наверняка также захочется часть из них снабдить надписями. К тому же, используя доступные команды, можно создать нечто большее, чем простой расцвеченный шрифт. Итак, начнем с предположения, что вами получено задание создать заголовок для сайта по плетению корзин под названием WickerpediA (вообще-то такой сайт уже есть, но все равно займемся этим делом).

Для начала нужно соответствующим образом выбрать подходящий шрифт и размер, возможно, как в примере 24.10, где стиль шрифта выбран полужирным, размером 140 пикселов и с гарнитурой Times. Кроме того, для свойства `textBaseline` установлено значение `top`, стало быть, методу `strokeText` можно передать координаты (0, 0) для верхней левой исходной точки текста, помещая его в верхнем левом углу холста. Как это выглядит, показано на рис. 24.9.

Пример 24.10. Запись текста на холст

```
context.font         = 'bold 140px Times'
context.textBaseline = 'top'
context.strokeText('WickerpediA', 0, 0)
```

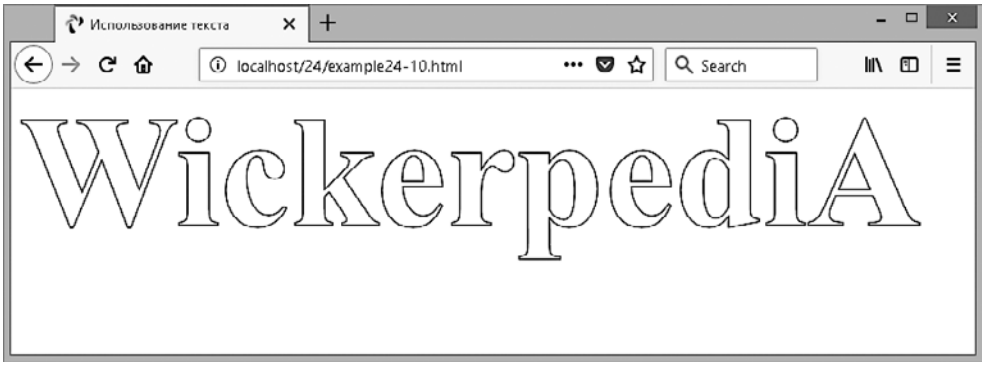

Рис. 24.9. Текст был записан на холст

Метод strokeText

Для записи текста на холсте нужно отправить текстовую строку и пару координат методу `strokeText`:

```
context.strokeText('WickerpediA', 0, 0)
```

Предоставленные координаты *x* и *y* будут использоваться в качестве относительной ссылки свойствами `textBaseLine` и `textAlign`.

Этот метод — использование рисования линий — является единственным способом рисования текста на холсте. Поэтому вдобавок ко всем следующим свойствам, влияющим на текст, на отображение текста будет также влиять свойство рисования линий `lineWidth` (которое далее будет рассмотрено более подробно).

Свойство textBaseLine

Свойство `textBaseLine` может быть задано с любым из следующих значений:

- `top` — выравнивание по верху текста;
- `middle` — выравнивание по середине текста;
- `alphabetic` — выравнивание по основной линии букв текста;
- `bottom` — выравнивание по низу текста.

Свойство font

По стилю шрифт может быть полужирным (`bold`), наклонным (`italic`) или обычным (`normal`, используется по умолчанию) либо комбинацией наклонного полужирного, а значения размера могут выражаться в `em`, `ex`, `px`, `%`, `in`, `cm`, `mm`, `pt` или `pc`, точно так же, как при использовании CSS. Шрифтом должен быть один из доступных в текущем браузере шрифтов, что обычно означает, что это должен быть один из шрифтов `Helvetica`, `Impact`, `Courier`, `Times` или `Arial`, или может быть выбран исходный шрифт пользовательской системы с засечками или без них. Если есть уверенность, что браузеру доступен другой нужный вам шрифт, можно также указать и его, но после него рекомендуется также добавить один из наиболее распространенных или стандартных вариантов, чтобы в том случае, если у пользователя предпочитаемый вам шрифт не установлен, мог быть выполнен более простой вариант стилизации.

Если требуется использовать такой шрифт, как Times New Roman, в названии которого имеются пробелы, соответствующую строку кода нужно заменить чем-нибудь подобным следующей строке. В ней внешние кавычки отличаются от тех, в которые заключено название шрифта:

```
context.font = 'bold 140px "Times New Roman"'
```

Свойство textAlign

Вдобавок к выбору способа выравнивания текста по вертикали можно указать горизонтальное выравнивание, задав свойству `textAlign` одно из следующих значений:

- `start` — выравнивание текста по левому краю, если направление текста в документе — слева направо; в противном случае выравнивание текста по правому краю. Эта настройка используется по умолчанию;
- `end` — выравнивание текста по правому краю, если направление текста в документе — слева направо; в противном случае выравнивание текста по левому краю;
- `left` — выравнивание текста по левому краю;
- `right` — выравнивание текста по правому краю;
- `center` — выравнивание текста по центру.

Это свойство используется следующим образом:

```
context.textAlign = 'center'
```

Применительно к текущему примеру нужно, чтобы текст был выровнен по левому краю, впритык к краю холста, поэтому свойство `textAlign` не используется и, следовательно, устанавливается заданное по умолчанию выравнивание по левому краю.

Метод fillText

Можно также выбрать свойство заливки текста на холсте, для чего нужно задать любой чистый цвет, линейный или радиальный градиент либо узорную заливку. Воспользуемся для вашего заголовка узорной заливкой на основе текстуры плетеной корзины, как в примере 24.11, результат выполнения которого показан на рис. 24.10.

Рис. 24.10. Теперь у текста есть узорная заливка

Пример 24.11. Заливка текста с помощью узора

```
image        = new image()
image.src = 'wicker.jpg'
image.onload = function()
{
  pattern           = context.createPattern(image, 'repeat')
  context.fillStyle = pattern
  context.fillText(  'WickerpediA', 0, 0)
  context.strokeText('WickerpediA', 0, 0)
}
```

На всякий случай в этом примере сохранен вызов метода `strokeText`, чтобы обеспечить черный контур текста; без него не было бы достаточно выраженных краев.

Здесь можно было также использовать широкое разнообразие других типов или узоров заливки, а простота работы с холстами существенно облегчает возможности экспериментирования. Более того, при желании, имея в наличии заголовок, можно также выбрать сохранение его копии путем вызова метода `toDataURL`, что уже было подробно рассмотрено в данной главе. Затем можно воспользоваться изображением в качестве логотипа, к примеру, для выкладывания на другие сайты.

Метод measureText

При работе с текстом на холсте иногда может понадобиться узнать, сколько пространства он будет занимать, чтобы выбрать для него самое подходящее место. Выполнить эту задачу можно с помощью метода `measureText` следующим образом (предполагая, что все различные свойства текста к данному моменту уже были определены):

```
metrics = context.measureText('WickerpediA')
width = metrics.width
```

Поскольку высота текста в пикселах эквивалентна размеру шрифта в пунктах при его определении, объект `metrics` показателя высоты не предоставляет.

Рисование линий

Холст предоставляет множество функций рисования линий для удовлетворения практически всех потребностей, включая выбор линий, их законцовок и соединений, а также путей и кривых всех типов. Но начнем со свойства, которое уже затрагивалось в предыдущем разделе.

Свойство lineWidth

Все методы рисования на холсте с применением линий пользуются рядом свойств этих линий, наиболее важным из которых является свойство `lineWidth`. Работать с этим свойством ничуть не сложнее, чем указывать ширину линии в пикселах, подобно следующему примеру, где задается ширина 3 пиксела:

```
context.lineWidth = 3
```

Свойства lineCap и lineJoin

Когда рисуемые линии заканчиваются и имеют ширину, превышающую 1 пиксел, можно выбрать тип появляющейся законцовки этой линии, используя свойство `lineCap`, у которого могут быть значения `butt` (используется по умолчанию), `round` или `square`. Например:

```
context.lineCap = 'round'
```

Кроме того, в местах соединения линий шире 1 пиксела важно точно указать, как они должны встретиться. Это делается с помощью свойства `lineJoin`, у которого могут быть значения `round`, `bevel` или `miter` (используется по умолчанию), например:

```
context.lineJoin = 'bevel'
```

В примере 24.12 (который из-за своей сложности здесь приведен полностью) в сочетании применяются все три значения каждого свойства, создавая информативный результат, показанный на рис. 24.11. Приведенные в данном примере методы `beginPath`, `closePath`, `moveTo` и `lineTo` будут рассмотрены далее.

Пример 24.12. Показ комбинаций законцовок и соединений линий

```
<!DOCTYPE html>
<html>
  <head>
    <title>Рисование линий</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='535' height='360'></canvas>

    <script>
      canvas               = O('mycanvas')
      context              = canvas.getContext('2d')
      S(canvas).background = 'lightblue'
      context.fillStyle    = 'red'
      context.font         = 'bold 13pt Courier'
      context.strokeStyle  = 'blue'
      context.textBaseline = 'top'
      context.textAlign    = 'center'
      context.lineWidth    = 20
      caps                 = ['  butt', ' round', 'square']
      joins                = [' round', ' bevel', ' miter']

      for (j = 0 ; j < 3 ; ++j)
      {
        for (k = 0 ; k < 3 ; ++k)
        {
          context.lineCap  = caps[j]
          context.lineJoin = joins[k]

          context.fillText(' cap:' + caps[j], 88 + j * 180, 45 + k * 120)
```

```
          context.fillText('join:' + joins[k], 88 + j * 180, 65 + k * 120)

          context.beginPath()
          context.moveTo( 20 + j * 180, 100 + k * 120)
          context.lineTo( 20 + j * 180, 20 + k * 120)
          context.lineTo(155 + j * 180, 20 + k * 120)
          context.lineTo(155 + j * 180, 100 + k * 120)
          context.stroke() context.closePath()
        }
      }
    </script>
  </body>
</html>
```

Этот код настраивает несколько свойств, а затем использует два цикла (один вложен в другой): один цикл применяется для законцовок, а другой — для соединений. Внутри центрального цикла сначала задаются текущие значения для свойств `lineCap` и `lineJoin`, которые затем показываются на холсте с помощью метода `fillText`.

Используя эти настройки, код рисует девять фигур с линиями шириной 20 пикселов, каждая из которых, как показано на рис. 24.11, имеет отличную от других комбинацию настроек законцовок и соединений линий.

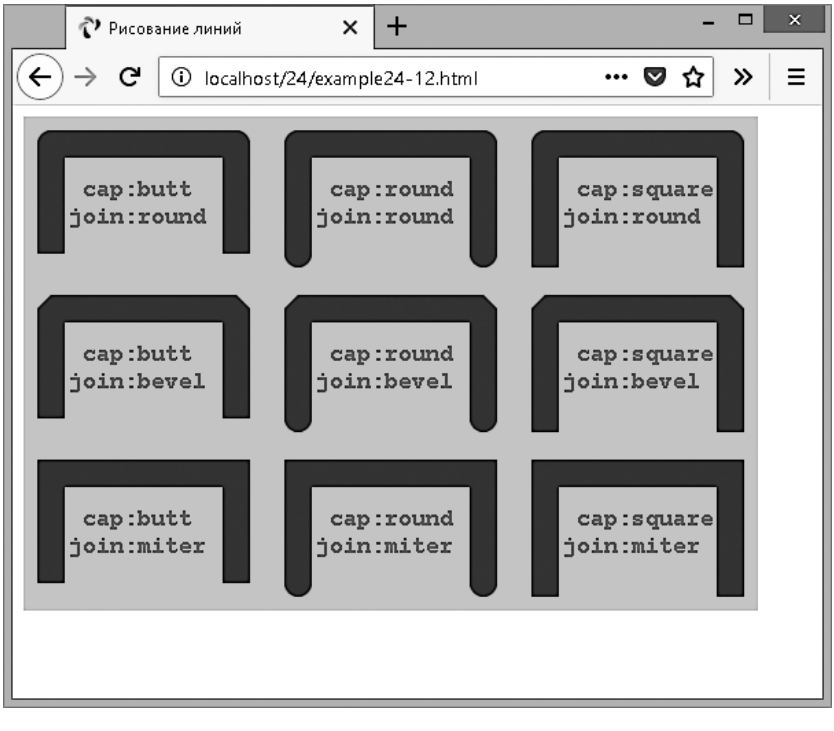

Рис. 24.11. Все комбинации законцовок и соединений линий

Как видите, законцовки с необработанным торцом (`butt`) короткие, квадратные законцовки (`square`) длиннее, а скругленные законцовки (`round`) по размеру находятся где-то между этими двумя. В то же время скругленные соединения линий (`round`) имеют изогнутую форму, скошенные (`bevel`) имеют срезанные углы, а угловые (`miter`) имеют острые углы. Соединения линий также применяются по отношению к углам, отличающимся от прямых.

Свойство miterLimit

Если обнаружится, что угловые соединения, подвергшиеся обрезке, слишком коротки, их можно продлить с помощью свойства `miterLimit`:

```
context.miterLimit = 15
```

По умолчанию используется значение `10`, поэтому лимит углового соединения можно также укоротить. Если для свойства `miterLimit` не установлено достаточно большое для углового соединения значение, то заостренные угловые соединения станут просто скошенными. Поэтому, если не удается сделать соединение угловым, нужно просто увеличивать значение, предоставляемое свойству `miterLimit`, до тех пор, пока не сформируется угол.

Использование путей

В предыдущем примере использовались два метода настройки путей для того, чтобы им следовали методы рисования линий. Метод `beginPath` устанавливает начало пути, а метод `closePath` — конец пути. Внутри каждого пути можно использовать различные методы для перемещения места рисования и для создания линий, кривых линий и других фигур. Изучим соответствующий фрагмент примера 24.12, упрощенный до создания одного экземпляра узора:

```
context.beginPath()
context.moveTo(20, 100)
context.lineTo(20,  20)
context.lineTo(155, 20)
context.lineTo(155,100)
context.stroke()
context.closePath()
```

В этом кодовом фрагменте путь начинается в первой линии, а затем место рисования перемещается в позицию на 20 пикселов в сторону и на 100 пикселов вниз от верхнего левого угла холста, для чего используется вызов метода `moveTo`.

Затем следуют три вызова метода `lineTo`, рисующие три линии, сначала вверх к позиции (20; 20), затем направо к (155; 20), а затем вниз к (155; 100). После установки этого пути для его прокладывания вызывается метод `stroke`, и, наконец, путь закрывается, поскольку он больше не нужен.

Важно закрыть путь сразу же, как только работа по его прокладыванию будет завершена, в противном случае при использовании нескольких путей можно получить весьма неожиданные результаты.

Методы moveTo и lineTo

Методы `moveTo` и `lineTo` получают в качестве своих аргументов простые координаты *x* и *y*, но `moveTo` поднимает воображаемое перо с текущего места и затем перемещает его на новое место, а `lineTo` рисует линию от текущей позиции воображаемого пера до указанной новой позиции. Или, точнее говоря, линия будет нарисована после вызова метода `stroke`, и никак иначе. Поэтому просто скажем, что метод `lineTo` создает *потенциальную* рисуемую линию, но она, к примеру, в равной степени может быть частью контура для области заливки.

Метод stroke

Метод `stroke` предназначен для фактического рисования всех линий, созданных до этого в пути на холсте. Если он вызывается в пределах незакрытого пути, тут же прорисовывается все вплоть до последней позиции воображаемого пера.

Но если путь закрыт, а затем выдан вызов метода `stroke`, в результате получится эффект соединения пути от текущего местоположения воображаемого пера до начала пути, что в данном примере превратит фигуры в прямоугольники (а это для нас нежелательно, поскольку нам нужно показать законцовки и соединения линий).

Этот эффект соединения закрытого пути нам еще пригодится (как будет показано ниже) для создания надлежащим образом замкнутых путей перед применением к ним любых нужных вам методов заливки. В противном случае графика, используемая вами для заливки, может выйти за пределы пути.

Метод rect

Если понадобится создать прямоугольники с четырьмя (а не с тремя, как в предыдущем примере) нарисованными сторонами (без закрытия пути), может быть выдан еще один вызов метода `lineTo` (выделенный полужирным шрифтом) для замыкания всего контура:

```
context.beginPath()
context.moveTo(20, 100)
context.lineTo(20, 20)
context.lineTo(155, 20)
context.lineTo(155, 100)
context.lineTo(20, 100)
context.closePath()
```

Но есть более простой способ рисования таких прямоугольников, предусматривающий использование метода `rect`:

```
rect(20, 20, 155, 100)
```

Всего лишь при одном вызове эта команда получает две пары координат x и y и рисует прямоугольник с верхним левым углом в позиции (20; 20) и нижним правым углом на позиции (155; 100).

Заливка областей

Используя пути, можно создавать сложные области, которые также могут быть залиты с применением чистого цвета, градиента или узора. В примере 24.13 для создания сложного, похожего на звезду узора используются основы тригонометрии. Не стану вдаваться в математические подробности, потому что для примера это неважно (хотя, если хотите поэкспериментировать с кодом, попробуйте для получения других эффектов изменить значения, присвоенные переменным `points`, `scale1` и `scale2`).

Пример 24.13. Заливка сложного пути

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заливка пути </title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='320'></canvas>
    <script>
      canvas               = O('mycanvas')
      context              = canvas.getContext('2d')
      S(canvas).background = 'lightblue'
      context.strokeStyle  = 'orange'
      context.fillStyle    = 'yellow'

      orig = 160 points = 21
      dist = Math.PI / points * 2 scale1 = 150 scale2 = 80

      context.beginPath()

      for (j = 0 ; j < points ; ++j)
      {
        x = Math.sin(j * dist)
        y = Math.cos(j * dist)
        context.lineTo(orig + x * scale1, orig + y * scale1)
        context.lineTo(orig + x * scale2, orig + y * scale2)
      }

      context.closePath()
      context.stroke()
      context.fill()
    </script>
  </body>
</html>
```

Реальный интерес представляют строки, выделенные полужирным шрифтом, в которых задается начало пути, находятся два вызова метода `lineTo`, определяющие очертание, задается закрытие пути, а затем вызываются методы `stroke` и `fill` для прорисовки очертания оранжевым цветом и заливки фигуры желтым цветом (рис. 24.12).

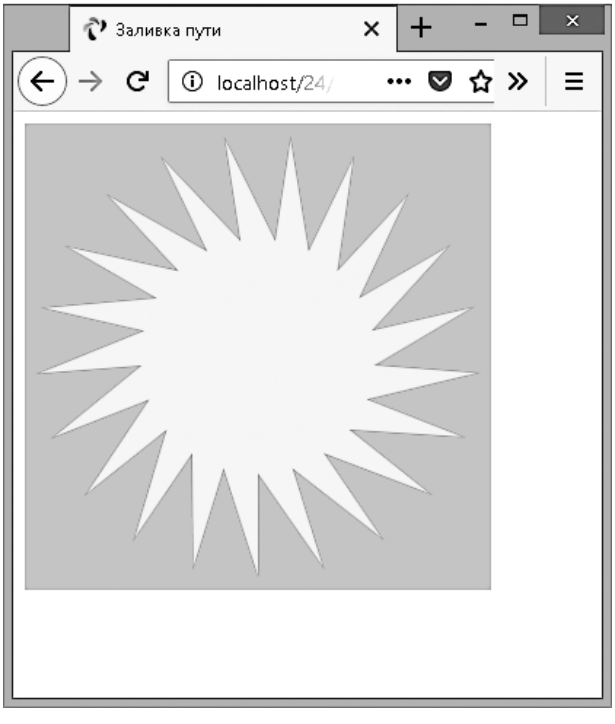

Рис. 24.12. Рисование и заливка сложного пути

Благодаря путям можно создать объект любой сложности либо с помощью формул, либо с помощью циклов (как в данном примере), либо просто с помощью длинной череды дополнительных вызовов методов moveTo и (или) lineTo или других вызовов.

Метод clip

Иногда при построении пути может потребоваться проигнорировать участки холста (возможно, если что-то рисуется «позади» другого объекта, нужно отобразить только видимую часть). Это делается с помощью метода `clip`, который создает границу, за пределами которой такие методы, как `stroke`, `fill` или другие методы, не будут иметь никакого эффекта.

Чтобы проиллюстрировать это, в примере 24.14 создается эффект, похожий на жалюзи. Сначала указатель воображаемого пера перемещается к левому краю, затем с помощью метода `lineTo` рисуется по направлению к правому краю, затем вниз на 30 пикселов, а затем опять назад к левому краю и т. д. Тем самым создается разновидность извилистого узора, состоящего из серии горизонтальных полос, нарисованных на холсте (рис. 24.13).

Пример 24.14. Создание вырезанной области

```
context.beginPath()

for (j = 0 ; j < 10 ; ++j)
{
  context.moveTo(20,  j * 48)
  context.lineTo(620, j * 48)
  context.lineTo(620, j * 48 + 30)
  context.lineTo(20,  j * 48 + 30)
}

context.stroke()
context.closePath()
```

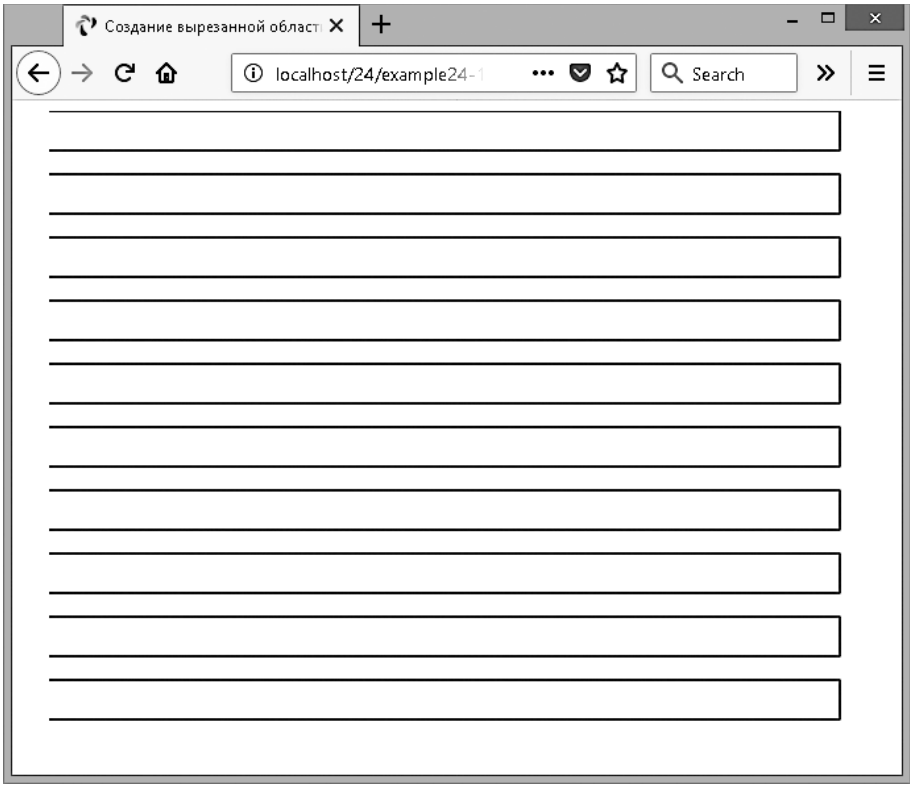

Рис. 24.13. Путь горизонтальных полос

Для превращения этого примера в вырезанную область холста нужно заменить вызов метода `stroke` (выделен в примере полужирным шрифтом) вызовом метода `clip`:

```
context.clip()
```

Теперь контур полос не будет виден, но вырезанная область, которая состоит из всех отдельно взятых полос, останется на месте. Чтобы проиллюстрировать это, в примере 24.15 осуществляется подстановка метода, затем предыдущий пример дополняется рисованием на холсте простой картинки зеленой травы под голубым небом, на котором находится сияющее солнце (отредактированное из примера 24.12). Все изменения выделены полужирным шрифтом, а результат показан на рис. 24.14.

Пример 24.15. Рисование внутри границ вырезанной области

```
context.fillStyle = 'white'
context.strokeRect(20, 20, 600, 440) // Черная граница
context.fillRect( 20, 20, 600, 440)  // Белый фон

context.beginPath()

for (j = 0 ; j < 10 ; ++j)
{
  context.moveTo(20,  j * 48)
  context.lineTo(620, j * 48)
  context.lineTo(620, j * 48 + 30)
  context.lineTo(20,  j * 48 + 30)
}
context.clip()
context.closePath()

context.fillStyle = 'blue'    // Синее небо
context.fillRect(20, 20, 600, 320)
context.fillStyle = 'green'   // Зеленая трава
context.fillRect(20, 320, 600, 140)
context.strokeStyle = 'orange'
context.fillStyle = 'yellow'

orig   = 170
points = 21
dist   = Math.PI / points * 2
scale1 = 130
scale2 = 80

context.beginPath()

for (j = 0 ; j < points ; ++j)
{
   x = Math.sin(j * dist)
   y = Math.cos(j * dist)
   context.lineTo(orig + x * scalel, orig + y * scale1)
   context.lineTo(orig + x * scale2, orig + y * scale2)
}

context.closePath()
context.stroke()       // Контур солнца
context.fill()         // Заливка солнца
```

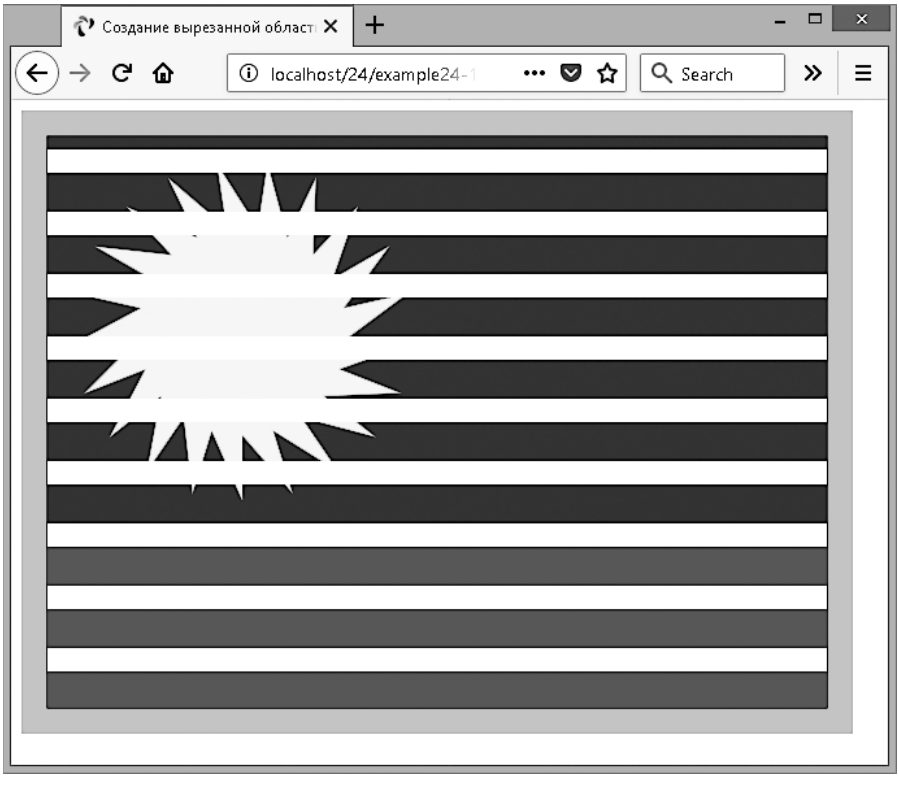

Рис. 24.14. Рисование осуществляется только внутри разрешенной вырезанной области

Мы здесь собираемся не выигрывать некое состязание, а показать, насколько действенным может быть вырезание при его эффективном использовании.

Метод isPointInPath

Иногда нужно знать, находится ли конкретная точка на выстроенном пути. Но, вероятнее всего, этот метод пригодится только в том случае, если вы очень хорошо разбираетесь в JavaScript и создаете весьма непростую программу. Тогда он будет вызываться, как правило, в составе условной инструкции `if`:

```
if (context.isPointInPath(23, 87))
{
  // Здесь выполняются какие-нибудь действия
}
```

Первым аргументом в вызове является координата позиции *x*, а вторым — координата позиции *y*. Если указанная позиция находится в любой точке пути, метод возвращает значение `true` и выполняется содержимое инструкции `if`. В противном случае возвращается значение `false` и содержимое инструкции `if` не выполняется.

Идеальным случаем использования метода `isPointInPath` можно считать игры с применением холста, в которых нужно проверять, попала ли ракета в цель, достиг ли мяч стены или биты или выполнение подобных пограничных условий.

Работа с кривыми линиями

В дополнение к прямым путям можно создавать практически бесконечное разнообразие криволинейных путей, выбирая различные методы, от позволяющих рисовать простые дуги и окружности до тех, что дают возможность рисовать сложные кривые второй степени и кривые Безье.

Фактически для создания многих линий, прямоугольников и кривых использовать пути не нужно, поскольку их можно рисовать непосредственно вызовом их методов. Но использование путей позволяет более тонко контролировать ситуацию, поэтому я почти всегда предпочитаю, как показано в следующих примерах, рисовать кривые по указанным путям.

Метод arc

Метод arc требует передачи ему координат центра дуги *x* и *y* и радиуса в пикселах. Наряду с этими значениями нужно передать пару смещений в радианах и необязательного направления:

```
context.arc(55, 85, 45, 0, Math.PI / 2, false)
```

Поскольку по умолчанию действует направление по часовой стрелке (значение `false`), его указание может быть опущено или изменено на `true` для рисования дуги против часовой стрелки.

В примере 24.16 создаются три набора из четырех дуг, первые две из которых рисуются по часовой стрелке, а третья и четвертая — против. Кроме того, для первого набора из четырех дуг их пути закрываются до вызова метода `stroke`, поэтому начальная и конечная точки соединяются, а два других набора дуг рисуются до того, как путь закрывается, поэтому соединения не происходит.

Пример 24.16. Рисование различных дуг

```
context.strokeStyle = 'blue' arcs =
[
  Math.PI,
  Math.PI * 2,
  Math.PI / 2,
  Math.PI / 180 * 59
]

for (j = 0 ; j < 4 ; ++j)
{
  context.beginPath()
  context.arc(80 + j * 160, 80, 70, 0, arcs[j])
  context.closePath()
```

```
  context.stroke()
}

context.strokeStyle = 'red' for (j = 0 ; j < 4 ; ++j)
{
  context.beginPath()
  context.arc(80 + j * 160, 240, 70, 0, arcs[j])
  context.stroke()
  context.closePath()
}

context.strokeStyle = 'green'

for (j = 0 ; j < 4 ; ++j) {
  context.arc(80 + j * 160, 400, 70, 0, arcs[j], true)
  context.stroke() context.closePath()
}
```

Для того чтобы код был короче, я нарисовал все дуги, используя циклы, поэтому длина каждой дуги хранилась в массиве `arcs`. Эти значения выражены в радианах, а поскольку радиан равен 180 / ϖ (где ϖ — это отношение длины окружности к ее диаметру, или приблизительно 3,1415927), они вычисляются следующим образом:

- `Math.PI` — эквивалентно 180°;
- `Math.PI * 2` — эквивалентно 360°;
- `Math.PI / 2` — эквивалентно 90°;
- `Math.PI / 180 * 59` — эквивалентно 59°.

На рис. 24.15 показаны три строки с дугами и проиллюстрированы оба варианта использования аргумента направлений, где значение `true` было установлено для последнего набора, и важность выбора места закрытия пути в зависимости от того, нужно ли рисовать линию, соединяющую стартовую и конечную позиции.

Если вместо радиан вы предпочитаете работать с градусами, можно создать новую функцию библиотеки Math:

```
Math.degreesToRadians = function(degrees)
{
  return degrees * Math.PI / 180
}
```

А затем заменить весь код по созданию массива, начинающийся в примере 24.16 со второй строки, следующим кодом:

```
arcs =
[
  Math.degreesToRadians(180),
  Math.degreesToRadians(360),
  Math.degreesToRadians(90),
  Math.degreesToRadians(59)
]
```

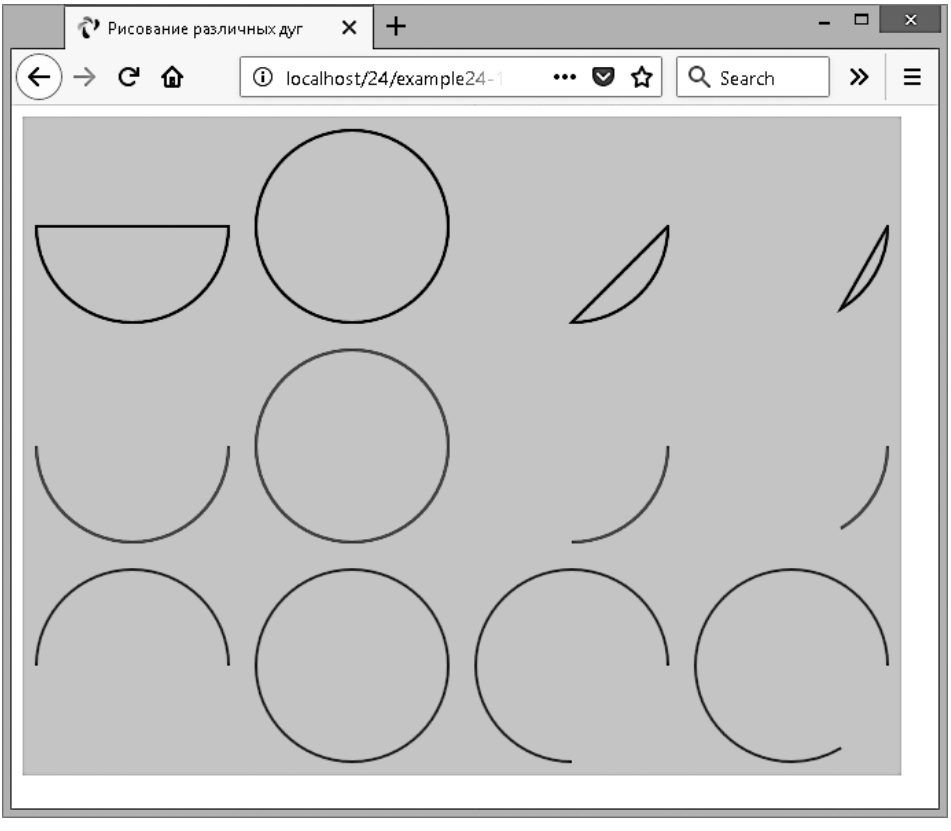

Рис. 24.15. Разнообразие типов дуг

Метод arcTo

Вместо создания сразу всей дуги можно выбрать проведение дуги из текущей позиции в пути до другой позиции, как в следующем вызове метода `arcTo` (который просто требует двух пар координат x и y и радиуса):

```
context.arcTo(100, 100, 200, 200, 100)
```

Передаваемые методу позиции представляют собой точки, где воображаемые касательные линии подводятся к контуру дуги в ее стартовой и конечной точках.

Чтобы проиллюстрировать, как это все работает, в примере 24.17 рисуются восемь различных дуг с радиусами от 0 до 280 пикселов. При каждом проходе цикла создается новый путь со стартовой точкой в позиции (20; 20). Затем рисуется дуга с использованием воображаемых касательных из этой позиции к позиции (240; 20) и из этой позиции к позиции (460; 20). В данном случае определяются пары касательных под углом 90° друг к другу в форме буквы V.

Пример 24.17. Рисование восьми дуг с разными радиусами

```
for (j = 0 ; j <= 280 ; j += 40)
{
  context.beginPath()
  context.moveTo(20, 20)
  context.arcTo(240, 240, 460, 20, j)
  context.lineTo(460, 20)
  context.stroke()
  context.closePath()
}
```

Метод `arcTo` рисует только до той точки, где дуга встречается со второй воображаемой касательной. Поэтому после каждого вызова метода `arcTo` метод `lineTo` создает оставшуюся часть линии с того места, в котором остановил свою работу метод `arcTo`, до позиции (460; 20). Затем результат рисуется на холсте путем вызова метода `stroke` и путь закрывается.

Как видно на рис. 24.16, когда вызывается метод `arcTo` со значением радиуса `0`, создается острое соединение. В данном случае это прямой угол (но если две воображаемые касательные находятся под другими углами по отношению друг к другу, соединение будет под этим углом). Затем по мере увеличения радиуса можно увидеть, что дуги становятся все больше и больше.

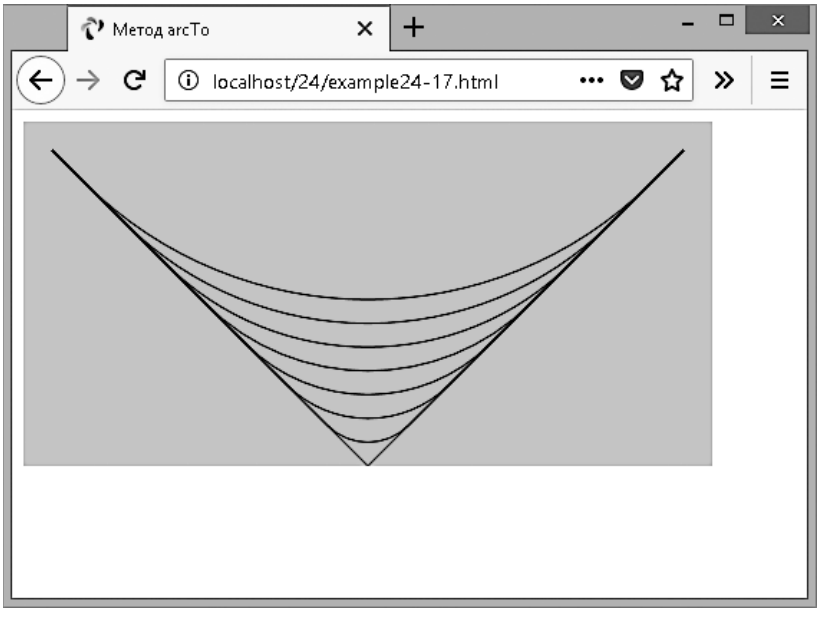

Рис. 24.16. Рисование дуг с разными радиусами

По сути, лучше всего `arcTo` использовать для создания кривой из одной части рисунка в другую, рисуя дугу на основе предыдущей и последующей позиций,

как будто они были касательными к создаваемой дуге. Если это воспринимается слишком сложно, не стоит переживать: скоро вы всему научитесь и поймете, что это действительно удобный и логичный способ рисования дуг.

Метод quadraticCurveTo

Как бы ни были дуги полезны, они являются всего лишь одним из типов кривых линий и могут стать препятствием для создания более сложных конструкций. Но нам нечего бояться: существуют и другие способы рисования кривых, например путем использования метода `quadraticCurveTo`. С помощью этого метода можно поместить воображаемую точку притяжения ближе к кривой (или дальше от нее), чтобы притянуть ее в этом направлении, точно так же, как меняется траектория объекта в космосе за счет его притяжения гравитацией планет и звезд, возле которых он проходит.

Но, в отличие от гравитации, чем дальше находится точка притяжения, тем больше она к себе притягивает!

В примере 24.18 имеется шесть вызовов этого метода, создающего путь для кучевого облака, которое затем заливается белым цветом. На рис. 24.17 показано, как углы пунктирной линии снаружи облака представляют точки притяжения, применяемые к каждой кривой.

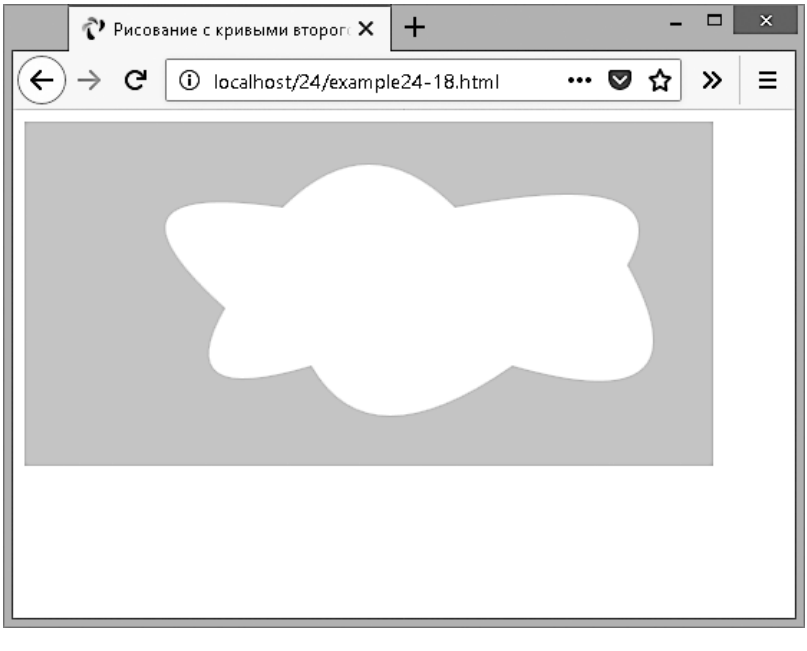

Рис. 24.17. Рисование с кривыми второго порядка

Пример 24.18. Рисование облака с кривыми второго порядка

```
context.beginPath()
context.moveTo(180, 60)
context.quadraticCurveTo(240,   0, 300,  60)
context.quadraticCurveTo(460,  30, 420, 100)
context.quadraticCurveTo(480, 210, 340, 170)
context.quadraticCurveTo(240, 240, 200, 170)
context.quadraticCurveTo(100, 200, 140, 130)
context.quadraticCurveTo( 40,  40, 180,  60)
context.fillStyle = 'white'
context.fill()
context.closePath()
```

Кстати, для получения на этом изображении пунктирной линии вокруг облака я воспользовался методом stroke в связке с методом setLineDash, которому передается список, представляющий длины пунктиров и пробелов. В данном случае я использовал setLineDash([2, 3]), но вы можете создавать пунктирные линии любой требуемой сложности, например setLine- Dash([1, 2, 1, 3, 5, 1, 2, 4]). Я не стал документировать это свойство, так как пока оно было реализовано только в IE, Opera и Chrome. Я очень надеюсь, что вскоре оно будет реализовано и в других браузерах, поскольку это было бы великолепным дополнением для создания, к примеру, контуров и границ карт.

Метод bezierCurveTo

Если гибкости кривых второго порядка вам не хватает, то как вы отнесетесь к наличию доступа к двум точкам притяжения для каждой кривой? Используя метод `bezierCurveTo`, можно делать то же самое, что и в примере 24.19, где кривая создается между позициями (24; 20) и (240; 220), но с невидимыми точками притяжения за пределами холста (в данном случае) в позициях (720; 480) и (–240; –240). Форма, получаемая этой кривой, показана на рис. 24.18.

Пример 24.19. Создание кривой Безье с двумя точками притяжения

```
context.beginPath()
context.moveTo(240, 20)
context.bezierCurveTo(720, 480, -240, -240, 240, 220)
context.stroke()
context.closePath()
```

Точки притяжения не обязательно должны быть по разные стороны от кривой, поскольку их можно помещать где угодно, и когда они расположены близко друг к другу, будет оказываться комбинированное притяжение (в отличие от точек притяжения, расположенных с разных сторон, как в предыдущем примере). Использование этих разновидностей методов рисования кривых дает возможность рисовать любые типы кривых, которые только могут понадобиться.

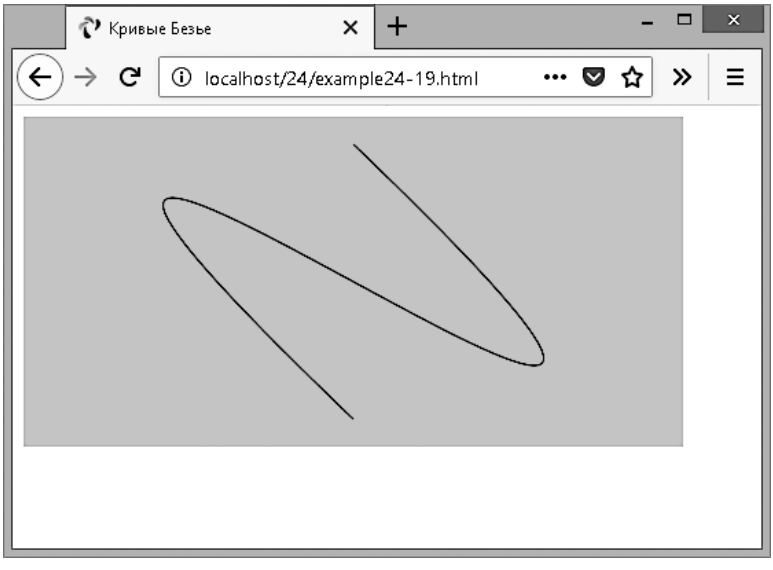

Рис. 24.18. Кривая Безье с двумя точками притяжения

Обработка изображений

Графические методы позволяют не только рисовать и вести запись на холсте, но и помещать на него изображения или извлекать их с холста. При этом вы не ограничены простыми командами копирования и вставки, поскольку изображения при их чтении или записи можно растягивать и искривлять, а также получить полный контроль над эффектами наложения и теней.

Метод drawImage

Используя метод `drawImage`, можно взять объект изображения, загруженный с сайта, выложить его на сервер или даже извлечь с холста и нарисовать его на холсте.

Метод поддерживает большое разнообразие аргументов, многие из которых являются необязательными, но в наипростейшем варианте `drawImage` можно вызвать, как показано далее, где ему передаются только изображение и пара координат *x* и *y*:

```
context.drawImage(myimage, 20, 20)
```

Эта команда рисует изображение, содержащееся в объекте `myimage` на холсте с контекстом `context`, с верхним левым углом изображения в позиции 20, 20.

Чтобы обеспечить загрузку изображения до его использования, лучше всего заключить код обработки рисунка в функцию, вызываемую только после загрузки изображения:

```
myimage = new Image() myimage.src = 'image.gif'

myimage.onload = function()
{
  context.drawImage(myimage, 20, 20)
}
```

Изменение размеров изображения

Если нужно изменить размеры изображения при помещении его на холст, то к вызову добавляется вторая пара аргументов, представляющая требуемую ширину и высоту (выделено полужирным шрифтом):

```
context.drawImage(myimage, 140, 20, 220, 220)
context.drawImage(myimage, 380, 20,  80, 220)
```

Здесь изображение помещается в два места: в первое, точку с координатами (140; 20), где оно увеличивается (со 100-пиксельного до 220-пиксельного), и во второе, которое относится к позиции (380; 20), где изображение сжато по горизонтали и расширено по вертикали, чтобы получить ширину и высоту 80 × 220 пикселов.

Выбор области изображения

Вы не обязаны использовать все изображение, можно также выбрать в нем область при использовании `drawImage`. Это может, к примеру, пригодиться, если нужно поместить все графические изображения, в отношении которых имеются планы, в один файл, а затем просто извлечь нужную часть изображения. Этот прием разработчики часто используют для ускорения загрузки страницы и снижения нагрузки на сервер.

Но выполнение такой задачи не обходится без небольших ухищрений, потому что вместо добавления дополнительных аргументов в конец списка для этого метода при извлечении области изображения данные аргументы нужно поместить первыми.

Например, чтобы поместить изображение в позицию (20; 140), нужно выдать следующую команду:

```
context.drawImage(myimage, 20, 140)
```

А для передачи методу высоты и ширины 100 × 100 пикселов нужно изменить вызов (изменения выделены полужирным шрифтом):

```
context.drawImage(myimage, 20, 140, 100, 100)
```

Но, к примеру, для захвата (или вырезки) подраздела размером 40 × 40 пикселов с верхним левым углом изображения (30; 30) нужно вызвать метод подобным образом (где новые аргументы выделены полужирным шрифтом):

```
context.drawImage(myimage, 30, 30, 40, 40, 20, 140)
```

А для изменения размеров захваченной части до квадрата со стороной 100 пикселов нужно воспользоваться следующим кодом:

```
context.drawImage(myimage, 30, 30, 40, 40, 20, 140, 100, 100)
```

Я считаю, что такой способ работы метода слишком запутан, и не могу найти этому никакого логического объяснения. Но, поскольку он так работает, боюсь, что с этим ничего не поделаешь и остается только заставить себя запомнить, какой аргумент куда помещается и при каких условиях.

В примере 24.20 используются различные вызовы метода `drawImage` для получения результата, показанного на рис. 24.19. Для того чтобы было понятнее, я разредил аргументы пробелами, чтобы значения каждого столбца представляли одинаковую информацию.

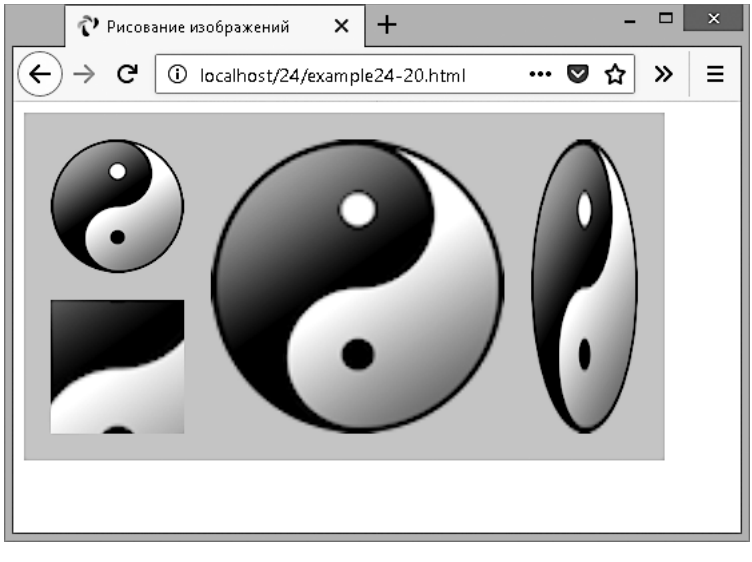

Рис. 24.19. Рисование изображений на холсте с изменениями размеров и вырезкой

Пример 24.20. Различные способы рисования изображения на холсте

```
myimage = new Image()
myimage.src = 'image.png'

myimage.onload = function()
```

```
{
   context.drawImage(myimage,                  20,  20          )
   context.drawImage(myimage,                 140,  20, 220, 220)
   context.drawImage(myimage,                 380,  20,  80, 220)
   context.drawImage(myimage, 30, 30, 40, 40,  20, 140, 100, 100)
}
```

Копирование с холста

Холст также можно использовать в качестве исходного изображения для рисования на том же (или на другом) холсте. Нужно просто предоставить вместо объекта изображения имя объекта холста и воспользоваться всеми остальными аргументами точно так же, как они применялись бы с изображением.

Добавление теней

Когда на холсте рисуется изображение (либо его часть) или какой-нибудь другой объект, можно также поместить тень под этим рисунком с помощью одного или нескольких следующих свойств:

- `shadowOffsetX` — горизонтальное смещение в пикселах, на величину которого тень должна быть сдвинута вправо (или влево при отрицательном значении);
- `shadowOffsetY` — вертикальное смещение в пикселах, на величину которого тень должна быть сдвинута вниз (или вверх при отрицательном значении);
- `shadowBlur` — количество пикселов, над которым будет размыто очертание тени;
- `shadowColor` — основной цвет, используемый для тени. Если применяется размытие, этот цвет будет в размываемой области смешиваться с фоном.

Эти свойства могут применяться к тексту и линиям, а также к цельным изображениям, как в примере 24.21, где тени добавляются к тексту, изображению и объекту, созданному с использованием пути. На рис. 24.20 можно увидеть тени, разумно обтекающие видимые части изображений, а не только их прямоугольные границы.

Пример 24.21. Применение теней при рисовании на холсте

```
myimage     = new image()
myimage.src = 'apple.png'

orig   = 95
points = 21
dist   = Math.PI / points * 2
scale1 = 75
scale2 = 50

myimage.onload = function()
{
  context.beginPath()

  for (j = 0 ; j < points ; ++j)
```

```
  {
    x = Math.sin(j * dist)
    y = Math.cos(j * dist)
    context.lineTo(orig + x * scale1, orig + y * scale1)
    context.lineTo(orig + x * scale2, orig + y * scale2)
  }

  context.closePath()

  context.shadowOffsetX = 5
  context.shadowOffsetY = 5
  context.shadowBlur    = 6
  context.shadowColor   = '#444'
  context.fillStyle     = 'red'
  context.stroke()
  context.fill()

  context.shadowOffsetX = 2
  context.shadowOffsetY = 2
  context.shadowBlur    = 3
  context.shadowColor   = 'yellow'
  context.font          = 'bold 36pt Times'
  context.textBaseline  = 'top'
  context.fillStyle     = 'green'
  context.fillText('Sale now on!', 200, 5)

  context.shadowOffsetX = 3
  context.shadowOffsetY = 3
  context.shadowBlur    = 5
  context.shadowColor   = 'black'
  context.drawImage(myimage, 245, 45)
}
```

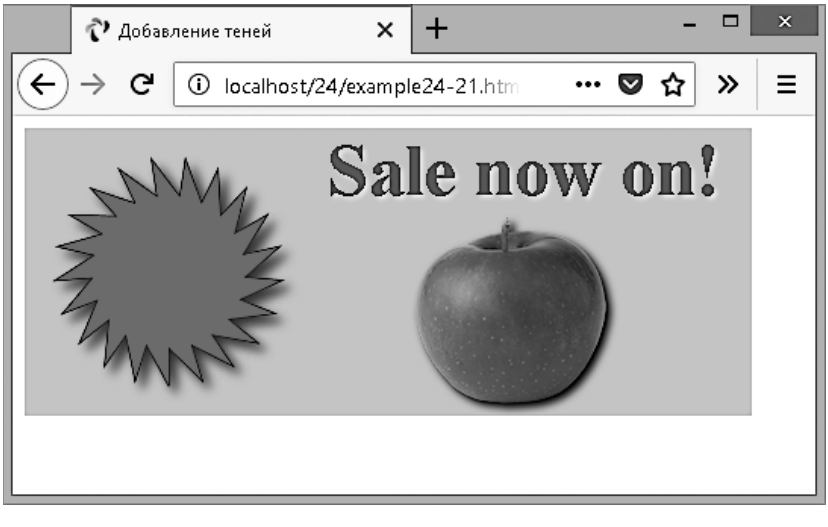

Рис. 24.20. Тени под различными типами рисованных объектов

Редактирование на уровне пикселов

Холсты в HTML5 не только предоставляют множество эффективных методов рисования, но и позволяют закатав рукава покопаться в изображениях непосредственно на уровне пикселов с помощью трех весьма действенных методов.

Метод getImageData

Метод `getImageData` позволяет захватить часть холста (или весь холст целиком), предоставляя возможность как угодно изменять извлеченные данные, а затем сохранять их или помещать где-нибудь на холсте (либо на другом холсте).

Чтобы проиллюстрировать работу метода, в примере 24.22 сначала загружается уже готовое изображение, которое рисуется на холсте. Затем данные с холста считываются в объект по имени `idata`, где все цвета усредняются для перевода каждого пиксела в оттенки серого цвета, а затем, как показано на рис. 24.21, немного корректируются для сдвига каждого цвета в сторону светло-коричневых оттенков. В следующем разделе рассматривается массив пикселов `data` и что происходит при прибавлении или вычитании значения 50 из элементов массива.

Рис. 24.21. Преобразование изображения в светло-коричневые оттенки (при просмотре в оттенках серого будут видны лишь незначительные различия)

Пример 24.22. Манипулирование данными изображения

```
myimage = new Image()
myimage.src = 'photo.jpg'

myimage.onload = function()
{
  context.drawImage(myimage, 0, 0)
```

```
  idata = context.getImageData(0, 0, myimage.width, myimage.height)

  for (y = 0 ; y < myimage.height ; ++y)
  {
    pos = y * myimage.width * 4

    for (x = 0 ; x < myimage.width ; ++x)
    {
      average =
      (
        idata.data[pos]     +
        idata.data[pos + 1] +
        idata.data[pos + 2]
      ) / 3

      idata.data[pos]     = average + 50
      idata.data[pos + 1] = average
      idata.data[pos + 2] = average – 50
      pos += 4;
    }
  }
  context.putImageData(idata, 320, 0)
}
```

Массив data

Эта манипуляция изображением работает благодаря массиву `data`, который является свойством объекта `idata`, возвращаемым при вызове метода `getImageData`. Этот метод возвращает массив, содержащий все пиксельные данные для выбранной области изображения в их составных частях из красного, зеленого, синего цвета и альфа-прозрачности. Таким образом, для сохранения каждого цветного пиксела используются четыре элемента данных.

Все данные сохраняются в массиве `data` последовательно, так что за значением для красного цвета идет значение для зеленого, а затем для синего цвета, после чего идет значение для альфа-прозрачности, затем следующий элемент массива со значением для красного цвета для следующего пиксела и т. д. Таким образом, для пиксела в позиции (0; 0) получаются следующие данные:

```
idata.data[0] // Уровень красного
idata.data[1] // Уровень зеленого
idata.data[2] // Уровень синего
idata.data[3] // Уровень альфа-прозрачности
```

Затем следуют данные для пиксела в позиции (1; 0):

```
idata.data[4] // Уровень красного
idata.data[5] // Уровень зеленого
idata.data[6] // Уровень синего
idata.data[7] // Уровень альфа-прозрачности
```

В данном изображении все продолжается таким же образом до самого правого пиксела изображения в строке 0 — это 320-й пиксел в позиции (319; 0). В этом месте

значение 319 умножается на 4 (количество элементов данных в каждом пикселе), чтобы прийти к следующему массиву элементов, содержащему данные пикселов:

```
idata.data[1276] // Уровень красного
idata.data[1277] // Уровень зеленого
idata.data[1278] // Уровень синего
idata.data[1279] // Уровень альфа-прозрачности
```

Это заставляет указатель данных проходить весь путь назад к первому столбцу изображения, но теперь к строке 1 в позиции (0; 1), которая (поскольку каждая строка в этом изображении имеет ширину 320 пикселов) находится по смещению $(0 \times 4) + (1 \times 320 \times 4)$, или 1280:

```
idata.data[1280] // Уровень красного
idata.data[1281] // Уровень зеленого
idata.data[1282] // Уровень синего
idata.data[1283] // Уровень альфа-прозрачности
```

Следовательно, если данные изображения хранятся в idata, ширина изображения — в w, а позиция пиксела, к которому идет обращение, — в x и y, ключевой используемой формулой при непосредственном доступе к данным изображения будет такая:

```
red   = idata.data[x * 4 + y * w * 4    ]
green = idata.data[x * 4 + y * w * 4 + 1]
blue  = idata.data[x * 4 + y * w * 4 + 2]
alpha = idata.data[x * 4 + y * w * 4 + 3]
```

Зная об этом, мы создали эффект светло-коричневых оттенков на рис. 24.21, взяв красный, зеленый и синий компоненты каждого пиксела и усреднив их, как в этом коде (где pos является изменяющимся указателем на место в массиве для текущего пиксела):

```
average =
(
  idata.data[pos]     +
  idata.data[pos + 1] +
  idata.data[pos + 2]
) / 3
```

Когда теперь в average содержится усредненное цветовое значение (получаемое путем сложения всех пиксельных значений и делением на 3), оно записывается обратно во все цвета пиксела, но красный компонент повышается на значение, равное 50, а синий понижается на такое же значение:

```
idata.data[pos]     = average + 50
idata.data[pos + 1] = average
idata.data[pos + 2] = average - 50
```

В результате в каждом пикселе увеличивается уровень красного цвета и уменьшается уровень синего, придавая изображению светло-коричневые тона (если этого не сделать, то при записи для этих цветов только усредненного значения получится монохромное изображение).

Если нужно осуществить более сложные манипуляции с изображением, можно обратиться к Halfpap (http://tinyurl.com/convolut1) или HTML5 Rocks (http://tinyurl.com/convolut2), в которых подробно рассмотрены вопросы свертки в холстах HTML5.

Метод putImageData

После внесения требуемых изменений в массив изображения `data` для записи этого изображения на холст, как показано в предыдущем примере, нужно лишь вызвать метод `putImageData`, передав ему объект `idata` и координаты верхнего левого угла, в котором оно должно появиться. Показанный ранее вызов помещает измененную копию изображения справа от оригинала:

```
context.putImageData(idata, 320, 0)
```

Если нужно изменить только часть холста, а весь холст захватывать не нужно, извлеките только ту часть, в которой содержится интересующая вас область. Обратную запись данных изображения не обязательно производить в то же место, откуда они были взяты, — их можно записать в любую часть холста.

Метод createImageData

Вы не обязаны создавать объект непосредственно с холста, новый объект можно также создать с пустыми данными, вызвав для этого метод `createImageData`. В следующем примере создается объект шириной 320 и высотой 240 пикселов:

```
idata = createImageData(320, 240)
```

Кроме того, новый объект можно создать из существующего объекта:

```
newimagedataobject = createImageData(imagedata)
```

Далее можно поступать с этими объектами по вашему усмотрению: добавлять к ним данные пикселов или изменять их содержимое каким-то другим способом, вставлять их в холст или создавать из них другие объекты и т. д.

Более сложные графические эффекты

У более сложных свойств, доступных в холстах HTML5, имеются возможности назначения разнообразных эффектов наложений и прозрачности, а также применения эффективных преобразований, таких как масштабирование, расширение и поворот.

Свойство globalCompositeOperation

Есть 12 различных методов, доступных для тонкой настройки способа помещения объекта на холст, принимая во внимание существующие и будущие объекты. Они называются настройками *наложения* и применяются следующим образом:

```
context.globalCompositeOperationProperty = 'source-over'
```

Существуют следующие типы наложений.

- `source-over` — используется по умолчанию. Новое изображение копируется на старое.
- `source-in` — показываются только те части нового изображения, которые будут появляться в границах старого изображения, а старое изображение удаляется. Любая альфа-прозрачность в новом изображении заставляет удалять оказавшееся под ней старое изображение.
- `source-out` — показываются любые части нового изображения, не появляющиеся в границах старого изображения, а старое изображение удаляется. Любая альфа-прозрачность в новом изображении заставляет удалять оказавшееся под ней старое изображение.
- `source-atop` — новое изображение показывается там, где оно накладывается на старое изображение. Старое изображение показывается там, где оно непрозрачно, а новое изображение прозрачно. Остальные области приобретают прозрачность.
- `destination-over` — новое изображение рисуется под старым.
- `destination-in` — старое изображение показывается только в местах наложения нового изображения на старое, но не в любых областях прозрачности нового изображения. Новое изображение не показывается.
- `destination-out` — показываются только те части старого изображения, на которые не накладываются непрозрачные части нового изображения. Новое изображение не показывается.
- `destination-atop` — новое изображение показывается там, где не показывается старое. Там, где происходит наложение старого и нового изображения, показывается старое изображение. Любая прозрачность в новом изображении не дает показываться в этой области старому изображению.
- `lighter` — сумма нового и старого изображений применяется таким образом, что там, где они не накладываются друг на друга, они отображаются как обычно, а там, где накладываются, показывается сумма обоих изображений, но в осветленном виде.
- `darker` — сумма нового и старого изображений применяется таким образом, что там, где они не накладываются друг на друга, они отображаются как обычно, а там, где накладываются друг на друга, показывается сумма обоих изображений, но в затемненном виде.

- `copy` — новое изображение копируется поверх старого. Любые прозрачные области нового изображения, накладываемые на старое изображение, не дают ему в этих областях отображаться.
- `xor` — там, где новое и старое изображения не накладываются друг на друга, они отображаются как обычно. А там, где они накладываются, их цветовые значения подвергаются операции исключающего ИЛИ.

В примере 24.23 показан эффект всех этих типов наложений путем создания 12 разных холстов, на каждом из которых находятся два объекта (окружность с заливкой и изображение инь-ян), смещенные относительно друг друга, но имеющие наложенные друг на друга области.

Пример 24.23. Использование всех 12 типов эффектов наложения

```
image     = new Image()
image.src = 'image.png'

image.onload = function()
{
  types =
  [
    'source-over',     'source-in',        'source-out',
    'source-atop',     'destination-over', 'destination-in',
    'destination-out', 'destination-atop', 'lighter',
    'darker',          'copy',             'xor'
  ]

  for (j = 0 ; j < 12 ; ++j)
  {
    canvas                = O('c' + (j + 1))
    context               = canvas.getContext('2d')
    S(canvas).background = 'lightblue'
    context.fillStyle     = 'red'

    context.arc(50, 50, 50, 0, Math.PI * 2, false)
    context.fill()
    context.globalCompositeOperation = types[j]
    context.drawImage(image, 20, 20, 100, 100)
  }
}
```

Как и в случае с другими примерами данной главы, этот пример (который можно загрузить с сопутствующего книге сайта) для улучшения отображения включает в себя код HTML и (или) CSS, который здесь не показывается, поскольку для операций, проводимых в программе, он не играет важной роли.

Для обхода каждого типа наложения, сохраненного в массиве `types`, в программе используется цикл `for`. При каждом проходе цикла создается новый контекст для следующего из 12 элементов холста, уже созданного в предыдущем (здесь не показанном) коде HTML, с идентификатором от `c1` до `c12`.

На каждый холст сначала в верхний левый угол помещается красный круг диаметром 100 пикселов, а поверх него со смещением вправо и вниз на 20 пикселов помещается изображение инь-ян. Результат каждого типа наложения в действии показан на рис. 24.22. Как видите, можно достичь весьма широкого разнообразия эффектов.

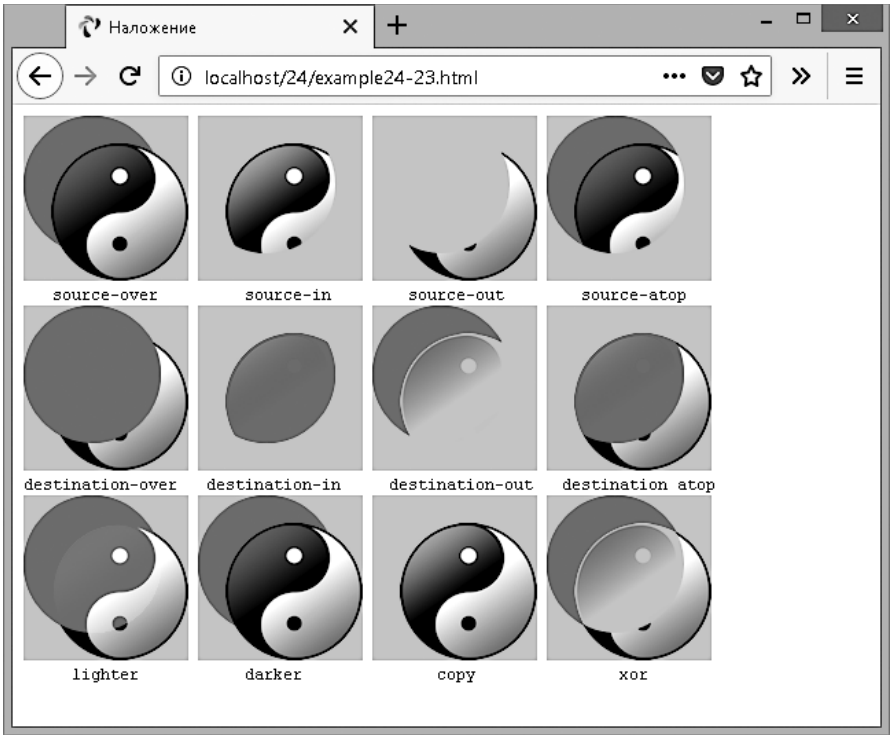

Рис. 24.22. Двенадцать эффектов наложения в действии

Свойство globalAlpha

При рисовании на холсте можно указать применяемую степень прозрачности, воспользовавшись для этого свойством `globalAlpha`, которое поддерживает значения от `0` (полная прозрачность) до `1` (полная непрозрачность). Следующая команда устанавливает тексту `The following command` значение альфа-прозрачности, равное `0.9`, то есть операции рисования будут проводиться при 90 % непрозрачности (или при 10 % прозрачности):

```
context.globalAlpha = 0.9
```

Это свойство может использоваться с другими свойствами, включая варианты наложений.

Преобразования

Холсты поддерживают четыре функции для применения преобразований к элементам при их рисовании на холсте HTML5: `scale`, `rotate`, `translate` и `transform`. Они могут использоваться поодиночке или вместе для создания еще более интересных эффектов.

Метод scale

Будущие операции рисования можно масштабировать, предварительно вызвав метод `scale` и предоставив ему коэффициенты горизонтального и вертикального масштабирования, которые могут быть со знаком «минус», нулем или положительным значением.

В примере 24.24 на холсте рисуется изображение инь-ян в своем исходном размере 100 × 100 пикселов. Затем применяется троекратное увеличение масштаба по горизонтали и двукратное по вертикали, а потом опять вызывается метод `drawImage` для помещения растянутого изображения рядом с исходным. И наконец, масштабирование применяется повторно со значениями `0.33` и `0.5`, чтобы все вернуть к нормальным размерам, а изображение рисуется еще раз, теперь уже под исходным изображением.

Результат показан на рис. 24.23.

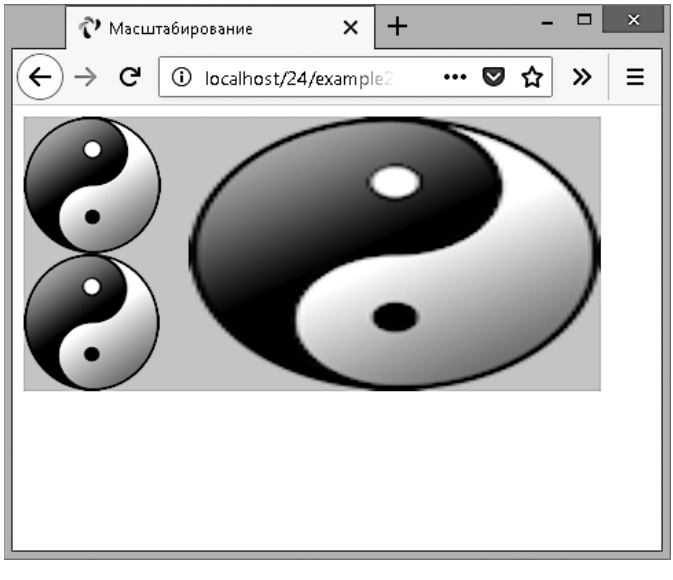

Рис. 24.23. Масштабирование изображения сначала вверх, а затем вниз

Пример 24.24. Масштабирование вверх и вниз

```
context.drawImage(myimage, 0, 0)
context.scale(3, 2)
context.drawImage(myimage, 40, 0)
context.scale(.33, .5) context.drawImage(myimage, 0, 100)
```

Если присмотреться, можно заметить, что копия под исходным изображением из-за масштабирования вверх-вниз немного размыта.

Путем использования для одного или нескольких параметров масштабирования отрицательных значений можно получить перевернутое изображение элемента в горизонтальном или вертикальном направлении (или в обоих направлениях) вместе с масштабированием (или вместо него). Например, следующий код переворачивает содержимое для получения зеркально отображенного изображения:

```
context.scale(-1, 1)
```

Методы save и restore

Если нужно применить несколько операций масштабирования к разным нарисованным элементам, результат может быть не только размытым, но и весьма затратным по времени, из-за того что вычисление троекратного увеличения масштаба требует применения значения `0.33` для обратного масштабирования (а увеличение масштаба в два раза требует для обратного процесса указать значение `0.5`).

Поэтому можно вызвать метод `saveto` для сохранения текущего контекста перед выдачей вызова `scale`, а чуть позже вернуть прежний нормальный масштаб, сделав вызов метода `restore`. Просмотрите следующий код, которым можно заменить код примера 24.24:

```
context.drawImage(myimage, 0, 0)
context.save()
context.scale(3, 2)
context.drawImage(myimage, 40, 0)
context.restore()
context.drawImage(myimage, 0, 100)
```

Методы `save` и `restore` работают весьма эффективно, поскольку не применяют к элементу масштабирование. Фактически они применяются параллельно со всеми следующими свойствами и поэтому могут использоваться в любой момент для сохранения текущих свойств с их последующим восстановлением: `fillStyle`, `font`, `globalAlpha`, `globalCompositeOperation`, `lineCap`, `lineJoin`, `lineWidth`, `miterLimit`, `shadowBlur`, `shadowColor`, `shadowOffsetX`, `shadowOffsetY`, `strokeStyle`, `textAlign` и `textBaseline`. Свойства всех четырех методов преобразования также управляются посредством `save` и `restore`: `scale`, `rotate`, `translate` и `transform`.

Метод rotate

Используя метод `rotate`, можно выбрать угол, под которым поместить объект (или результат работы любого из методов рисования) на холст. Этот угол указывается в радианах, каждый из которых можно выразить как 180 / ϖ, или около 57°.

Поворот осуществляется относительно исходной точки холста, которая по умолчанию находится в верхнем левом углу (но вскоре будет показано, что ее местоположение можно изменить). В примере 24.25 четыре раза показано изображение инь-ян с поворотом каждого последующего изображения на `Math.PI / 25` радиан.

Пример 24.25. Поворот изображения

```
for (j = 0 ; j < 4 ; ++j)
{
  context.drawImage(myimage, 20 + j * 120 , 20)
  context.rotate(Math.PI / 25)
}
```

Как видно на рис. 24.24, результат может не полностью совпадать с вашими ожиданиями, потому что изображение поворачивалось не вокруг своей оси. Вместо этого поворот осуществлялся вокруг исходной точки холста в позиции (0; 0). Кроме того, каждый новый поворот сочетался с предыдущим. Но, чтобы скорректировать ситуацию, можно применить метод `translate` в сочетании с методами `save` и `restore`.

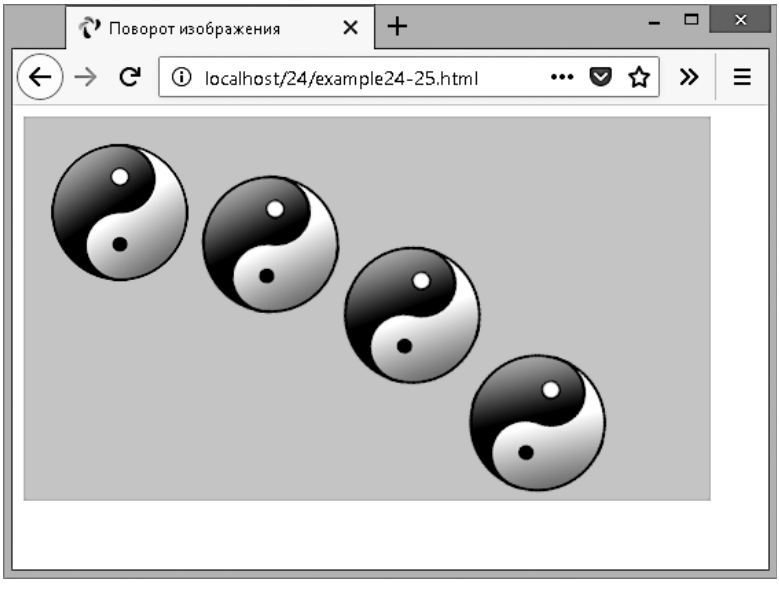

Рис. 24.24. Изображение после четырех разных поворотов

Радианы являются удобной единицей измерения, поскольку в полной окружности $\pi \times 2$ радиан. Следовательно, π радиан составляют половину окружности, $\pi / 2$ радиан — ее четверть, а $\pi / 2 \times 3$ (или $\pi \times 1{,}5$) радиан — это три четверти окружности и т. д. Чтобы не запоминать значение числа π, можно всегда ссылаться на его значение в виде Math.PI.

Метод translate

Для изменения исходной точки поворота можно вызвать метод `translate` и сместить эту точку в какое-нибудь другое место, которое должно быть где-нибудь внутри (или снаружи) холста или, чаще всего, где-нибудь в месте назначения объекта (обычно в его центре).

В примере 24.26 это перемещение выполняется перед каждым вызовом `rotate`, в результате чего теперь получается эффект, который, вероятнее всего, и ожидался в предыдущем примере. Кроме того, перед каждой операцией и после нее вызываются методы `save` и `restore`, обеспечивая независимое применение каждого поворота, не накладывающегося на предыдущий поворот.

Пример 24.26. Вращение объектов на месте

```
w = myimage.width
h = myimage.height

for (j = 0 ; j < 4 ; ++j)
{
  context.save()
  context.translate(20 + w / 2 + j * (w + 20), 20 + h / 2)
  context.rotate(Math.PI / 5 * j)
  context.drawImage(myimage, -(w / 2), -(h / 2))
  context.restore()
}
```

В этом примере перед каждым поворотом контекст сохраняется и исходная точка перемещается в самый центр того места, где будет нарисовано каждое изображение. Затем вызывается поворот, и благодаря предоставлению отрицательных значений изображение рисуется выше и левее новой исходной точки так, чтобы его центр совпадал с исходной точкой. Результат показан на рис. 24.25.

Напомню: когда нужно повернуть или преобразовать (эта операция будет рассмотрена ниже) объект на месте, требуется выполнить следующие действия.

1. Сохранить контекст.
2. Переместить исходную точку холста в центр того места, где будет находиться объект.
3. Выдать инструкцию поворота или преобразования.

4. Нарисовать объект с помощью любого поддерживаемого метода рисования, используя отрицательные значения позиции назначения в половину ширины объекта влево и в половину его высоты вверх.
5. Восстановить исходный контекст, чтобы вернуть исходную точку на место.

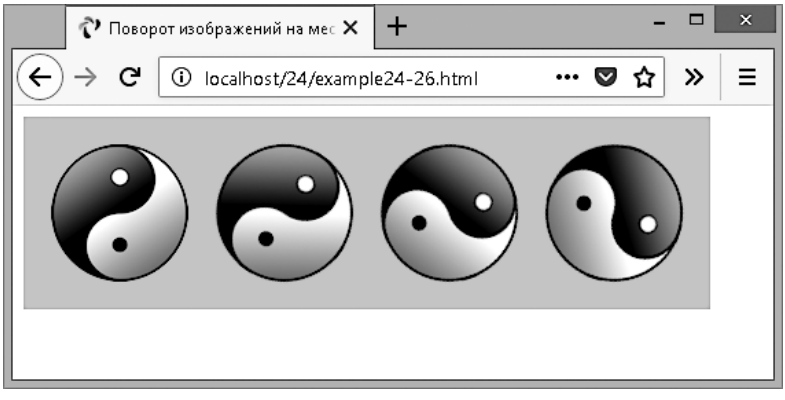

Рис. 24.25. Поворот изображений на месте

Метод transform

Когда будут исчерпаны все другие функции холста и вы все равно не сможете манипулировать объектами необходимым образом, настанет время обратиться к методу `transform`. С его помощью к объектам, рисуемым на холсте, можно применять матрицу преобразований, что откроет перед вами множество возможностей и эффективных функций, способных сочетать масштабирование и поворот в одной инструкции.

Используемая этим методом матрица преобразований имеет формат 3 × 3 и состоит из девяти значений, но только шесть из них предоставляются методу `transform` извне. Итак, вместо того чтобы объяснять, как эта матрица умножения работает, мне нужно всего лишь объяснить эффекты ее шести аргументов, которые составляют следующий упорядоченный список (порядок следования может быть несколько иным, чем тот, который подсказывает интуиция).

1. Горизонтальное масштабирование.
2. Горизонтальный наклон.
3. Вертикальный наклон.
4. Вертикальное масштабирование.
5. Горизонтальное перемещение.
6. Вертикальное перемещение.

Эти значения можно применить множеством различных способов, например имитируя метод `scale` из примера 24.24 и заменив такой вызов:

```
context.scale(3, 2)
```

вот этим:

```
context.transform(3, 0, 0, 2, 0, 0)
```

Подобным образом можно заменить вызов из примера 24.26:

```
context.translate(20 + w / 2 + j * (w + 20), 20 + h / 2)
```

следующим вызовом:

```
context.transform(1, 0, 0, 1, 20 + w / 2 + j * (w + 20), 20 + h / 2)
```

Обратите внимание, как аргументам горизонтального и вертикального масштабирования даются значения 1, чтобы обеспечить результат 1:1, а наклону дается значение 0, чтобы результат был без наклона.

Можно даже объединить предыдущие две строки кода, чтобы получить одновременно перемещение и масштабирование:

```
context.transform(3, 0, 0, 2, 20 + w / 2 + j * (w + 20), 20 + h / 2)
```

Если обратиться к аргументам `skew`, можно ожидать, что они приведут к наклону элемента в указанном направлении (к примеру, создадут из квадрата ромб).

В качестве другого приема наклона в примере 24.27 на холсте рисуется изображение инь-ян, затем следует его наклонная копия, созданная с помощью метода `transform`. Значение наклона может иметь любую отрицательную, нулевую или положительную величину, но я выбрал горизонтальное значение, равное `1`, которое привело к наклону нижней части изображения на одну его ширину вправо и пропорционально потащило за собой все остальное (рис. 24.26).

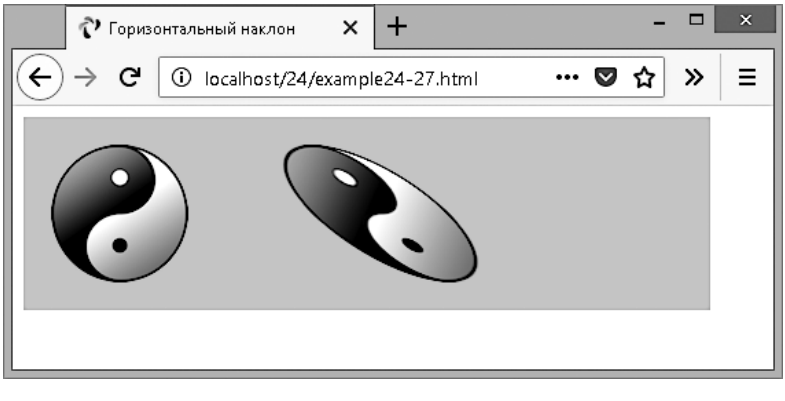

Рис. 24.26. Горизонтальный наклон объекта вправо

Пример 24.27. Создание исходного и наклонного изображений

```
context.drawImage(myimage, 20, 20)
context.transform(1, 0, 1, 1, 0, 0)
context.drawImage(myimage, 140, 20)
```

Можно даже повернуть объект с помощью transform, предоставив для наклона одно отрицательное и одно противоположное положительное значение. Но учтите: совершая эти действия, вы измените размер элемента и столкнетесь с необходимостью одновременной подстройки аргументов масштабирования. Кроме того, вам нужно будет не забыть о перемещении исходной точки. Поэтому, пока вы не наберетесь достаточного опыта в использовании метода transform, я рекомендую остановиться для выполнения этой задачи на методе rotate.

Метод setTransform

Вместо того чтобы использовать методы `save` и `restore`, можно настроить абсолютное преобразование, у которого имеется эффект перезапуска матрицы преобразования с последующим предоставлением значений. Метод `setTransform` работает так же, как метод `transform` (см. следующий пример, где применяется горизонтальный положительный наклон со значением `1`):

```
context.setTransform(1, 0, 1, 1, 0, 0)
```

Получить дополнительные сведения о матрицах преобразования можно в статье на эту тему в «Википедии»: http://tinyurl.com/transform-matrix.

Холст HTML5 предоставляет собой огромный новый актив для веб-разработчиков, чтобы можно было продолжить создавать более объемные, качественные, профессиональные и интересные сайты. В следующей главе будут рассмотрены два других важных усовершенствования HTML: встроенные в браузер и не требующие дополнительных модулей аудио и видео.

Вопросы

1. Как в HTML создать элемент холста?
2. Как предоставить JavaScript доступ к элементу холста?
3. Как указать старт и финиш при создании пути холста?
4. Какой метод можно использовать для извлечения данных с холста в изображение?
5. Как создать градиентную заливку из более чем двух цветов?
6. Как настроить ширину линий при рисовании?

7. Какой метод нужно использовать для указания части холста, чтобы предстоящее рисование происходило только внутри этой области?
8. Как нарисовать сложную кривую с двумя воображаемыми точками притяжения?
9. Сколько элементов данных для каждого пиксела возвращается методом `getlmageData`?
10. Какие два параметра метода `transform` применяются для операций масштабирования?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

25 Аудио и видео в HTML5

Одним из самых мощных стимулов возрастания роли Интернета стал постоянно увеличивающийся спрос со стороны пользователей на повышение доли мультимедийного содержимого в форме аудио- и видеоконтента. Поначалу высокая пропускная способность обходилась настолько дорого, что организовывать такие вещи, как живой поток, не представлялось возможным, а загрузка звуковой дорожки, не говоря уже о видео, могла продолжаться несколько минут или даже часов.

Высокая стоимость трафика и ограниченная доступность высокоскоростных модемов стимулировали разработку более быстродействующих и эффективных алгоритмов сжатия, таких как MP3-аудио и MPEG-видео, но даже при этом единственный способ загрузки файлов за разумное время был связан с резким ухудшением их качества.

Один из моих ранних интернет-проектов в далеком 1997 году представлял собой первую в Великобритании онлайн-радиостанцию, лицензированную музыкальными полномочными органами. Фактически это, скорее всего, был подкаст (еще до того, как этот термин был придуман), поскольку там создавалось ежедневное получасовое шоу, которое затем сжималось до восьмиразрядного моносигнала с полосой пропускания 11 кГц с помощью алгоритма, первоначально разработанного для телефонии. При этом звучание было телефонного качества или еще хуже. И все же мы быстро собрали тысячи слушателей, загружавших шоу и затем прослушивающих его по мере обхода обсуждаемых сайтов с использованием всплывающего окна браузера, содержащего дополнительный модуль.

К счастью для нас и для всех, кто публиковал мультимедиа, вскоре появилась возможность предложить более высокое качество аудио и видео, но по-прежнему только путем обращения к пользователю с просьбой загрузить и установить дополнительный модуль проигрывателя. Наиболее популярным из этих проигрывателей после победы над соперниками, например над RealAudio, стал Flash, но он получил

неважную репутацию по причине множественных сбоев браузера и постоянных требований обновления при выпуске новых версий.

Поэтому все согласились с тем, что в перспективе нужны веб-стандарты для поддержки мультимедиа непосредственно в браузере. Разумеется, такие разработчики браузеров, как Microsoft и Google, имели различное видение того, как должны выглядеть эти стандарты, но когда пыль улеглась, они пришли к согласию по поводу поднабора типов файлов, которые должны изначально воспроизводиться на всех браузерах, и этот поднабор был введен в спецификацию HTML5.

И наконец, теперь появилась возможность (если аудио и видео кодируются в нескольких различных форматах) выкладывать мультимедиа на веб-сервер, указывать пару HTML-тегов на веб-странице и проигрывать медиа в любом из основных браузеров для настольных компьютеров, смартфонов или планшетных устройств, не заставляя при этом пользователя загружать дополнительный модуль или вносить какие-либо другие изменения.

Еще не вышли из обращения старые браузеры, и для их поддержки сохраняется важная роль технологии Flash. В этой главе будет показано, как добавить код для использования Flash для подстраховки аудио и видео HTML5, чтобы охватить как можно больше комбинаций оборудования и программного обеспечения.

О кодеках

Кодек представляет собой *ко*дировщик-*дек*одировщик. Им описывается функциональная возможность, предоставляемая программным обеспечением, занимающимся кодированием и декодированием такого медиа, как аудио и видео. В HTML5 имеется несколько различных наборов доступных кодеков, состав которых зависит от используемого браузера.

Применение аудио и видео связано с одним осложнением, которое редко применяется к графике и другому традиционному веб-контенту. Имеется в виду лицензирование, осуществляемое в отношении форматов и кодеков. Многие из этих форматов и кодеков предоставляются за определенную плату, поскольку они были разработаны какой-либо компанией или консорциумом компаний, которые выбрали вариант получения лицензионных отчислений. Некоторые бесплатные браузеры с открытым исходным кодом не поддерживают самые популярные форматы и кодеки, потому что их использование невозможно оплатить, или потому что разработчики в принципе не согласны с лицензионными отчислениями. Поскольку законы об авторском праве варьируются от страны к стране и соблюдение лицензионных прав трудно обеспечить, кодеки обычно можно найти в Интернете без каких-либо затрат, но их использование в месте вашего проживания может быть технически незаконным.

HTML5-тегом `<audio>` поддерживаются следующие кодеки (в том числе и когда аудио прикреплено к HTML5-видео).

- *AAC*. Этот аудиокодек, чье название означает Advanced Audio Encoding, в числе других применяется в Apple-магазине iTunes и является частной запатентованной технологией, поддерживаемой Apple, Google и Microsoft. Как правило, для файлов используется расширение `.aac`. MIME-типом является `audio/aac`.
- *MP3*. Этот аудиокодек, чье название означает MPEG Audio Layer 3, доступен уже многие годы. Хотя это понятие зачастую (и неправильно) используется для ссылки на любой тип цифрового аудио, оно относится к частной запатентованной технологии, поддерживаемой Apple, Google, Mozilla Firefox и Microsoft. Для файлов используется расширение `.mp3`. MIME-типом является `audio/mpeg`.
- *PCM*. Этот аудиокодек, чье название означает Pulse Coded Modulation, сохраняет полноценные данные, закодированные аналого-цифровым преобразователем, и относится к формату, используемому для хранения данных на аудиокомпакт-дисках. Поскольку сжатие в нем не используется, он называется кодеком без потерь и его файлы, как правило, во много раз больше, чем файлы AAC или MP3. Поддерживается Apple, Mozilla Firefox, Microsoft и Opera. Обычно файлы этого типа имеют расширение `.wav`. MIME-типом является `audio/wav`, но может встретиться и тип `audio/wave`.
- *Vorbis*. Иногда называется Ogg Vorbis, поскольку для файлов, как правило, используется расширение `.ogg`. Этот аудиокодек не обременен патентами и свободен от выплаты роялти. Поддерживается Google, Mozilla Firefox и Opera. MIME-типом является `audio/ogg` и иногда `audio/oga`.

В следующем списке дается сводка основных операционных систем и браузеров, а также типов аудио, поддерживаемых их самыми последними версиями:

- **Apple iOS:** AAC, MP3, PCM;
- **Apple Safari:** AAC, MP3, PCM;
- **Google Android 2.3+:** AAC, MP3, Vorbis;
- **Google Chrome:** AAC, MP3, Vorbis;
- **Microsoft Internet Explorer:** AAC, MP3, PCM;
- **Microsoft Edge:** AAC, MP3, PCM;
- **Mozilla Firefox:** MP3, PCM, Vorbis;
- **Opera:** PCM, Vorbis.

Результат этих различных уровней поддержки кодеков приводит к тому, что для гарантированного проигрывания каждого аудиофайла на всех платформах всегда нужно предоставлять как минимум две его версии. Это могут быть PCM и AAC, PCM и MP3, PCM и Vorbis, AAC и Vorbis, или MP3 и Vorbis.

Элемент audio

Чтобы угодить всем платформам, нужно записать или преобразовать содержимое с использованием нескольких кодеков, а затем перечислить их в тегах `<audio>` и `</audio>`, как в примере 25.1. Вложенные теги `<source>` содержат различные медиа, которые вы желаете предложить браузеру. Благодаря предоставлению атрибута `controls` результат выглядит так, как показано на рис. 25.1.

Пример 25.1. Встраивание трех различных типов аудиофайлов

```
<audio controls>
  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>
```

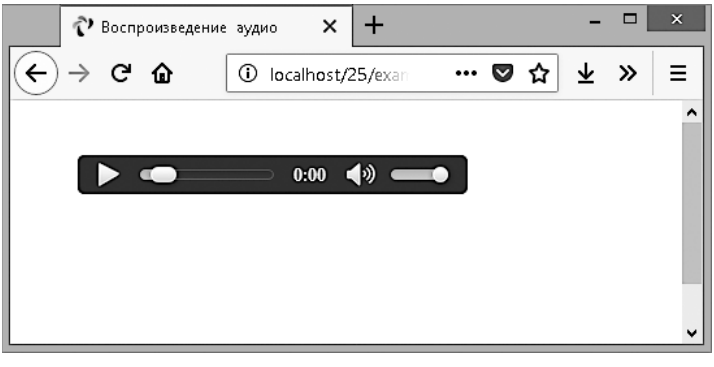

Рис. 25.1. Проигрывание аудиофайла

В данный пример я включил три различных типа аудио, потому что это вполне приемлемо и может пригодиться, если нужно обеспечить, чтобы каждый браузер смог найти предпочтительный для себя формат, а не только один из форматов, который он в состоянии обработать. Но можно быть уверенным, что пример все равно будет проигрываться на всех платформах, если из него выбросить один из файлов: либо MP3, либо AAC (но не оба сразу).

Тег `<audio>` и его партнер тег `<source>` поддерживают несколько атрибутов, в числе которых:

- `autoplay` — заставляет аудио запускаться на проигрывание сразу по готовности;
- `controls` — заставляет вывести панель управления;
- `loop` — устанавливает аудио на бесконечное проигрывание;
- `preload` — заставляет аудио приступить к загрузке даже до того, как пользователь выберет команду Play (Воспроизвести);
- `src` — указывает исходное местоположение аудиофайла;
- `type` — указывает кодек, используемый для создания аудио.

Если тег <audio> не снабдить атрибутом controls, а также не воспользоваться атрибутом autoplay, звук проигрываться не будет, а также не будет кнопки Play (Воспроизвести), которую можно было бы нажать для начала воспроизведения. Это не оставит вам никаких других вариантов, кроме как предложить эти функциональные возможности в JavaScript, как показано в примере 25.2 (с необходимым дополнительным кодом, выделенным полужирным шрифтом). У вас появится возможность проигрывать аудио и ставить его на паузу, как показано на рис. 25.2.

Пример 25.2. Проигрывание аудио с помощью JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Воспроизведение аудио с помощью JavaScript</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <audio id='myaudio'>
      <source src='audio.m4a' type='audio/aac'>
      <source src='audio.mp3' type='audio/mp3'>
      <source src='audio.ogg' type='audio/ogg'>
    </audio>

    <button onclick='playaudio()'>Play Audio</button>
    <button onclick='pauseaudio()'>Pause Audio</button>

    <script>
      function playaudio()
      {
        O('myaudio').play()
      }
      function pauseaudio()
      {
        O('myaudio').pause()
      }
    </script>
  </body>
</html>
```

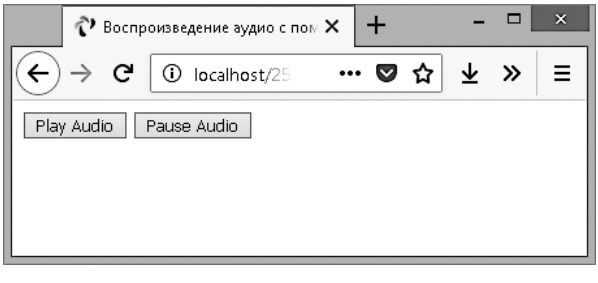

Рис. 25.2. HTML5-аудио можно управлять с помощью JavaScript

Проигрыватель будет работать при нажатии кнопок за счет вызова метода play или pause элемента myaudio.

Поддержка браузеров, не работающих с HTML5

В весьма редких случаях (например, при разработке веб-страниц для внутридомового интранета, использующего устаревшие браузеры) вам может понадобиться для обработки аудио вернуться к использованию Flash-технологии. В примере 25.3 показывается, как это можно сделать с использованием дополнительного модуля Flash, сохраненного в виде файла `audioplayer.swf` (который доступен, наряду со всеми примерами, в архивном файле на сопровождающем книгу сайте по адресу http://lpmj.net). Добавляемый код выделен полужирным шрифтом.

Пример 25.3. Предоставление Flash в качестве резервного варианта для браузеров, не поддерживающих HTML5

```
<audio controls>
  <object type="application/x-shockwave-flash"
    data="audioplayer.swf" width="300" height="30">
    <param name="FlashVars"
      value=''mp3=audio.mp3&showstop=1&showvolume=r'>
  </object>

  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>
```

Здесь мы воспользовались тем обстоятельством, что в браузерах, не поддерживающих HTML5, влияние оказывает все, что находится внутри тега `<audio>` (кроме игнорируемых элементов `<source>`). Следовательно, помещая туда элемент `<object>`, вызывающий Flash-проигрыватель, мы гарантируем, что у браузеров, не поддерживающих HTML5, по крайней мере будет шанс воспроизвести аудио, если у них есть установленный Flash (рис. 25.3).

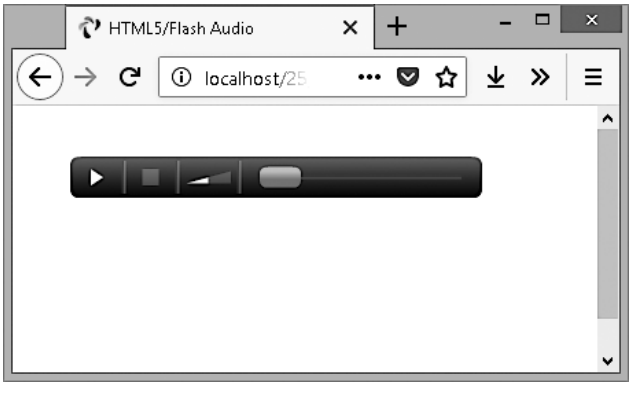

Рис. 25.3. Был загружен аудиопроигрыватель Flash

Конкретному аудиопроигрывателю `audioplayer.swf`, использованному в данном примере, передаются следующие аргументы и значения атрибута FlashVar элемента `<param>`:

- `mp3` — URL-адрес аудиофайла MP3;
- `showstop` — если передано значение `1`, выводится кнопка остановки, в противном случае она не показывается;
- `showvolume` — если передано значение `1`, выводится панель громкости, в противном случае она не показывается.

Как и в случае со многими другими элементами, вы можете легко изменить размеры объекта, например на 300 × 30 пикселов, предоставив эти значения его атрибутам `width` и `height`.

Элемент video

Проигрывание видео в HTML5 очень похоже на проигрывание аудио. Нужно просто воспользоваться тегом `<video>` и предоставить элементы `<source>` для предлагаемого вами медиа. В примере 25.4 показано, как это делается с тремя различными типами видеокодеков (рис. 25.4).

Пример 25.4. Проигрывание HTML5-видео

```
<video width='560' height='320' controls>
  <source src='movie.mp4'  type='video/mp4'>
  <source src='movie.webm' type='video/webm'>
  <source src='movie.ogv'  type='video/ogg'>
</video>
```

Рис. 25.4. Проигрывание HTML5-видео

Видеокодеки

Как и в случае с аудио, есть несколько доступных видеокодеков с различной поддержкой среди нескольких браузеров. Эти кодеки поступают в разных контейнерах.

- *MP4* — лицензированный стандартный формат мультимедийного контейнера, определяемый как часть MPEG-4, поддерживается браузерами Apple, Microsoft и в меньшей степени Google, у которой есть свой собственный формат контейнера WebM. MIME-типом является `video/mp4`.
- *Ogg* — бесплатный открытый формат контейнера, поддерживаемый Xiph.Org Foundation. Создатели формата Ogg утверждают, что он не ограничен патентами на ПО и предназначен для обеспечения эффективного потокового мультимедиа и управления высококачественным цифровым мультимедийным контентом. MIME-типом является `video/ogg` и иногда `video/ogv`.
- *WebM* — аудио- и видеоформат, предназначенный для предоставления свободного от выплаты роялти открытого формата сжатия видео для использования с видео в HTML5. Разработка проекта спонсируется компанией Google. Существует две версии: VP8 и более новая VP9. MIME-типом является `video/webm`.

Затем они могут содержать один из следующих видеокодеков.

- *H.264* — запатентованный видеокодек с правами собственности, проигрывание с применением которого для конечного пользователя бесплатно, но для всех частей процессов кодирования и передачи может потребовать выплаты роялти. На момент написания данного издания этот кодек поддерживали браузеры Apple, Google, Mozilla Firefox и браузеры компании Microsoft, но не поддерживал Opera (последний из основных браузеров).
- *Theora* — видеокодек, свободный от патентных обязательств и от выплаты роялти на всех уровнях кодирования передачи и проигрывания. Этот кодек поддерживается браузерами Google, Mozilla Firefox и Opera.
- *VP8* — этот видеокодек похож на Theora, но является собственностью компании Google, опубликовавшей его в качестве открытого кода и освободившей от выплат роялти. Поддерживается браузерами Google, Mozilla Firefox и Opera.
- *VP9* — предоставляет те же преимущества, что и VP8, но более эффективен и использует половину скорости потока данных.

В следующем списке дается сводка основных операционных систем и браузеров, а также типов видео, поддерживаемых их самыми последними версиями:

- **Apple iOS:** MP4/H.264;
- **Apple Safari:** MP4/H.264;
- **Google Android:** MP4, Ogg, WebM/H.264, Theora, VP8;

- **Google Chrome:** MP4, Ogg, WebM/H.264, Theora, VP8, VP9;
- **Microsoft Internet Explorer:** MP4/H.264;
- **Microsoft Edge:** MP4/H.264, WebM/H.264;
- **Mozilla Firefox:** MP4, Ogg, WebM/H.264, Theora, VP8, VP9;
- **Opera:** OGG, WebM/Theora, VP8.

Из этого списка становится ясно, что наиболее поддерживаемым является формат MP4/H.264, исключение составляет браузер Opera. Поэтому для достижения охвата в 96 % пользователей вам нужно предоставлять свое видео, использующее только этот один тип файла. Но для обеспечения максимальной возможности просмотра нужно все-таки кодировать еще и в формате Ogg/Theora или Ogg/VP8.

Следовательно, в файле `movie.webm` из примера 25.4 нет острой необходимости, но с его помощью показано, как можно добавлять различные подходящие вам типы файлов, чтобы дать браузерам возможность проигрывания предпочитаемого ими формата.

Элемент `<video>` и сопровождающий его тег `<source>` поддерживают следующие атрибуты:

- `autoplay` — заставляет видео запускаться на проигрывание сразу по готовности;
- `controls` — заставляет вывести панель управления;
- `height` — указывает высоту, с которой нужно показывать видео;
- `loop` — устанавливает видео на бесконечное проигрывание;
- `muted` — выключает звуковое сопровождение;
- `poster` — позволяет выбрать изображение, показываемое при проигрывании видео;
- `preload` — заставляет видео приступить к загрузке даже до того, как пользователь выберет команду Play (Воспроизвести);
- `src` — указывает исходное местоположение видеофайла;
- `type` — указывает кодек, используемый для создания видео;
- `width` — указывает ширину, с которой нужно показывать видео.

Если нужно управлять проигрыванием видео из JavaScript, это можно сделать с помощью кода, подобного показанному в примере 25.5 (с необходимым дополнительным кодом, выделенным полужирным шрифтом). Результат работы кода продемонстрирован на рис. 25.5.

Пример 25.5. Управление проигрыванием видео из JavaScript

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>Воспроизведение видео с помощью JavaScript</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <video id='myvideo' width='560' height='320'>
    <source src='movie.mp4' type='video/mp4'>
    <source src='movie.webm' type='video/webm'>
    <source src='movie.ogv' type='video/ogg'> </video><br>

    <button onclick='playvideo()'>Play Video</button>
    <button onclick='pausevideo()'>Pause Video</button>

    <script>
      function playvideo()
      {
        O('myvideo').play()
      }
      function pausevideo()
      {
        O('myvideo').pause()
      }
    </script>
  </body>
</html>
```

Рис. 25.5. Для управления видео был использован JavaScript

Этот код похож на тот, который использовался для управления проигрыванием аудио из JavaScript. Для проигрывания видео и установки его на паузу нужно просто вызвать метод `play` и (или) метод `pause` объекта `myvideo`.

Поддержка устаревших браузеров

В примере 25.6 показано, как вести разработку для браузеров, не совместимых с HTML5 (соответствующий код выделен полужирным шрифтом), с использованием файла `flowplayer.swf` (архивный файл с которым, включая примеры, доступен на сопровождающем книгу сайте по адресу http://lpmj.net). На рис. 25.6 видно, как это выглядит в браузере, не поддерживающем видео HTML5.

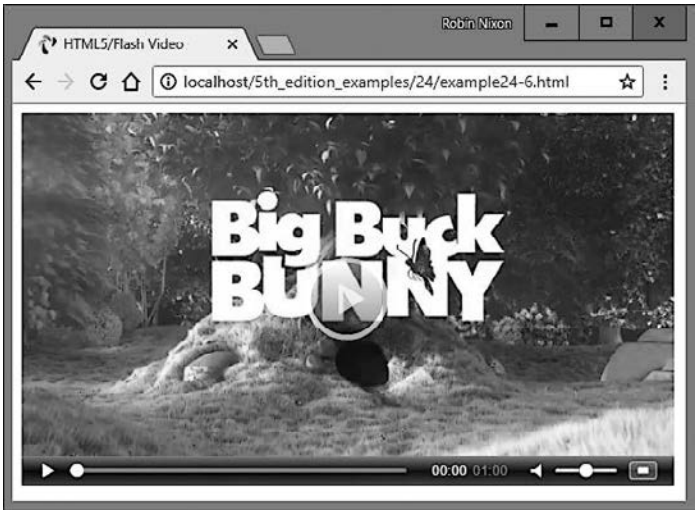

Рис. 25.6. Flash предоставляет удобную альтернативу для браузеров, не поддерживающих HTML5

Пример 25.6. Предоставление Flash в качестве резервного видеопроигрывателя

```
<video width='560' height='320' controls>
  <object width='560' height='320'
    type='application/x-shockwave-flash'
    data='flowplayer.swf'>
    <param name='movie' value='flowplayer.swf'>
    <param name='flashvars'
    value='config={"clip": {
      "url": "movie.mp4",
      "autoPlay":false, "autoBuffering":true}}'>
  </object>

  <source src='movie.mp4'  type='video/mp4'>
  <source src='movie.webm' type='video/webm'>
  <source src='movie.ogv'  type='video/ogg'>
</video>
```

Этот видеопроигрыватель Flash проявляет особую заботу о безопасности. Он не будет проигрывать видео с локальной файловой системы и станет работать только

с веб-сервера, поэтому я предоставил файл на сопровождающем книгу сайте (http://lpmj.net/video.mp4), чтобы его можно было проигрывать в данном примере.

Атрибуту FlashVar элемента `<param>` передаются следующие аргументы:

- `url` — URL-адрес файла `.mp4`, предназначенного для проигрывания (должен быть на веб-сервере);
- `autoPlay` — если передано значение `true`, проигрывание начинается автоматически; в противном случае ожидается щелчок на кнопке проигрывания;
- `autoBuffering` — если передано значение `true`, то, чтобы впоследствии свести к минимуму буферизацию при медленных подключениях, перед началом проигрывания видео будет предварительно загружено соответственно доступной пропускной способности.

Дополнительные сведения о программе Flash flowplayer (и HTML5-версии) вы найдете по адресу http://flowplayer.org.

Используя информацию, представленную в данной главе, вы сможете встроить любое нужное аудио и видео почти для всех браузеров и платформ, не переживая о том, могут или не могут пользователи их проиграть.

Из следующей главы вы узнаете, как использовать некоторые другие свойства HTML5, включая геолокацию и локальное хранилище.

Вопросы

1. Какие два тега HTML-элементов используются для вставки аудио и видео в документ HTML5?
2. Сколько аудиокодеков нужно использовать, чтобы гарантировать максимальную возможность проигрывания на всех платформах?
3. Какие методы можно вызывать для проигрывания медиа в HTML5 и для постановки его на паузу?
4. Как можно поддержать проигрывание медиа в браузерах, не работающих с HTML5?
5. Какие два видеокодека нужно использовать, чтобы гарантировать максимальную возможность проигрывания на всех платформах?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

26 Другие свойства HTML5

В этой последней главе из тех, что посвящены HTML5, будут рассмотрены вопросы использования геолокации, локального хранилища и продемонстрированы приемы перетаскивания, ставшие доступными в браузере.

Строго говоря, большинство из этих свойств (как и многое другое в HTML5) реально не являются расширениями HTML, поскольку доступ к ним осуществляется с помощью JavaScript, а не с помощью разметки HTML. Это просто технологии, которые были включены разработчиками браузеров и которым было дано общее удобное название HTML5.

Но это означает, что для их правильного применения вам нужно иметь полное представление о руководстве по JavaScript, приведенном в данной книге. И все же, освоив эти технологии, вы станете удивляться тому, как можно было раньше обходиться без таких эффективных новых функций.

Геолокация и служба GPS

Служба GPS (Global Positioning System) состоит из нескольких спутников, находящихся на земной орбите, с точно известными позициями. Когда устройство, применяющее GPS, развернуто в направлении этих спутников, различия во времени прихода сигналов от них позволяют устройству точно определить свое местоположение. Поскольку скорость света (а следовательно, и радиоволн) — известная постоянная величина, время, затрачиваемое сигналом на то, чтобы добраться от спутника до устройства GPS, показывает удаление этого спутника.

С учетом разных показателей времени поступления сигналов от разных спутников, для которых имеются точные сведения об их орбитальном местоположении в любой заданный момент времени, простое триангуляционное вычисление дает устройству сведения о его позиции относительно спутников с точностью до нескольких метров или даже выше.

Во многих мобильных устройствах, таких как телефоны или планшетные компьютеры, имеются микросхемы GPS, и они могут предоставить информацию о местоположении устройства. Но в одних устройствах таких микросхем нет, в других они могут быть выключены, а третьи могут использоваться внутри помещений, будучи закрытыми от спутников GPS, и поэтому не смогут получать какие-либо сигналы. В таких случаях для попытки определения местоположения устройства можно применять дополнительные технологии.

Следует также предупредить насчет соблюдения конфиденциальности при применении геолокации, особенно если используемое приложение в рамках своей работы передает координаты вашего устройства на сервер. У любого приложения с функциями геолокации должна быть явно выраженная политика соблюдения конфиденциальности. Кстати, с технической точки зрения геолокация фактически не относится к стандарту HTML5. Это автономная функция, определенная консорциумом W3C/WHATWG, но, по мнению большинства, она считается частью HTML5.

Другие методы определения местоположения

Если у вашего устройства нет микросхемы GPS, но есть аппаратура мобильного телефона, можно попробовать провести триангуляцию путем хронометрирования сигналов, полученных от различных вышек связи (чьи позиции известны с большой точностью), с которыми устройство может обменяться данными.

Если есть несколько вышек, местоположение может быть получено почти с такой же точностью, как и от GPS. Но если вышка только одна, то для определения приблизительного радиуса вокруг башни можно использовать силу сигнала, а созданная окружность представит район, в котором вы, скорее всего, находитесь. Место может быть определено в двух-трех километрах от вашего фактического местоположения, доходя по точности до нескольких десятков метров.

При отсутствии и этой возможности могут быть определены точки доступа Wi-Fi, чьи позиции известны в диапазоне работы вашего устройства. Поскольку у всех точек доступа имеются уникальные идентификационные адреса, называемые MAC-адресами (Media Access Control), можно получить достаточно хорошие приблизительные данные о местоположении, возможно, в пределах одной-двух улиц. Этот тип информации собирают машины просмотра улиц Google Street View Vehicles (от применения некоторых из которых со временем пришлось отказаться из-за потенциального нарушения прав на соблюдение конфиденциальности данных).

Если и этой возможности не окажется, можно будет запросить IP-адрес вашего устройства и воспользоваться им как грубым индикатором вашего местополо-

жения. Но чаще всего предоставляется место основного коммутатора, принадлежащего вашему интернет-провайдеру, которое может быть удалено от вас на десятки и даже сотни километров. На самый крайний случай ваш IP-адрес может (в большинстве случаев) уточнить страну, а иногда и регион, в котором вы находитесь.

IP-адреса часто применяются медиакомпаниями для территориальных ограничений по проигрыванию их контента. Но можно очень просто настроить прокси-серверы, использующие перенаправленный IP-адрес (на территории, которая блокирует доступ извне) для извлечения и передачи контента через блокаду непосредственно «иностранному» браузеру. Прокси-серверы также часто применяют для скрытия реального IP-адреса пользователя или для обхода цензурных ограничений. Они могут совместно применяться многими пользователями, к примеру, через точки подключения Wi-Fi. Поэтому, если вы определили чье-то местонахождение по IP-адресу, нельзя быть абсолютно уверенными в идентификации правильного региона или даже страны и эту информацию нужно воспринимать только как предположение.

Геолокация и HTML5

В главе 22 я коротко описал вам имеющуюся в HTML5 геолокацию. Теперь настало время взглянуть на нее поглубже, начиная с примера, который был дан ранее и показан снова в примере 26.1.

Пример 26.1. Вывод карты с учетом местоположения пользователя

```
<!DOCTYPE html>
<html>
  <head>
    <title>Пример применения геолокации</title>
  </head>
  <body>
    <script>
      if (typeof navigator.geolocation == 'undefined')
        alert("Геолокация не поддерживается.")
      else
        navigator.geolocation.getCurrentPosition(granted, denied)

      function granted(position)
      {
        var lat = position.coords.latitude
        var lon = position.coords.longitude

        alert("Разрешение дано. Ваше местонахождение:\n\n"
          + lat + ", " + lon +
          "\n\nClick 'OK' to load Google Maps with your location")

        window.location.replace("https://www.google.com/maps/@"
```

```
          + lat + "," + lon + ",8z")
      }

      function denied(error)
      {
        var message

        switch(error.code)
        {
          case 1: message = 'Доступ запрещен'; break;
          case 2: message = 'Позиция недоступна'; break;
          case 3: message = 'Время ожидания операции истекло'; break;
          case 4: message = 'Неизвестная ошибка'; break;
        }

        alert("Ошибка геолокации: " + message)
      }
    </script>
  </body>
</htm1>
```

Пройдемся по этому коду и посмотрим, как он работает, начиная с раздела `<head>`, в котором выводится заголовок. Раздел документа `<body>` составлен полностью из кода JavaScript, который тут же запускается с исследования свойства `navigator.geolocation`. Если возвращается значение `undefined`, то геолокация браузером не поддерживается и появляется окно предупреждения об ошибке.

В противном случае вызывается метод `getCurrentPosition`, которому передаются имена двух функций: `granted` и `denied` (не забывайте, что, передавая только лишь имена функций, мы передаем реальный код функции, а не результат ее вызова, который передавался бы в том случае, если бы к именам функций примыкали скобки):

```
navigator.geolocation.getCurrentPosition(granted, denied)
```

Эти функции появляются в сценарии чуть позже и предназначены для обработки двух возможных вариантов разрешения на предоставление данных о местоположении: разрешено (`granted`) или отказано (`denied`).

Первой следует функция `granted`, и вход в нее осуществляется, только если к данным может быть получен доступ. В таком случае переменным `lat` и `lon` присваиваются значения, возвращаемые выполняемыми в браузере процедурами геолокации.

Затем появляется окно уведомления со сведениями о текущем местоположении пользователя. Когда пользователь нажимает кнопку `OK`, окно уведомления закрывается и текущая веб-страница заменяется страницей одной из карт Google Maps. Этой странице передаются широта и долгота, возвращенные из вызова геолокации, и для нее используется увеличение поля видимости (`zoom`) с установленным значением `8`. Можно задать другой уровень увеличения, изменив в конце вызова `window.location.replace` значение `8z` на другое числовое значение, заканчивающееся буквой `z`.

Вывод на экран карты достигается вызовом метода `window.location.replace`. Результат показан на рис. 26.1.

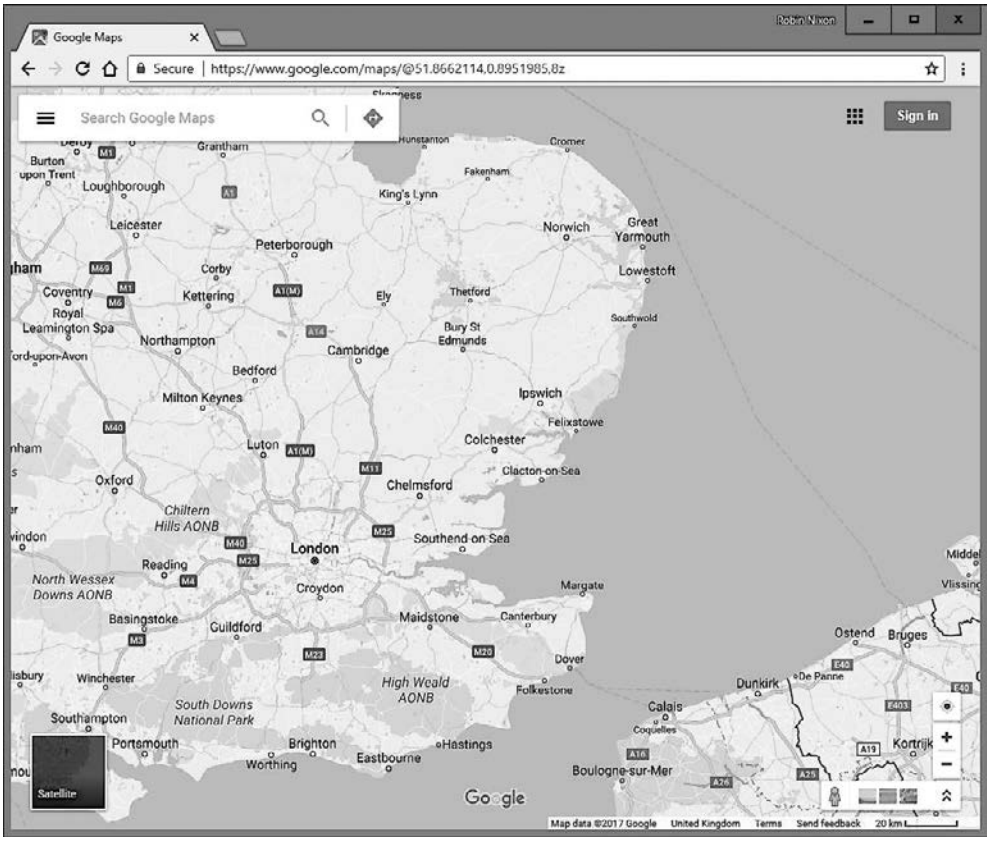

Рис. 26.1. Интерактивная карта, выведенная с учетом местоположения пользователя

Если в разрешении отказано (или возникла другая проблема), функцией `denied` будет выведено сообщение об ошибке, для чего ею будет показано свое собственное окно уведомления, информирующее пользователя о возникшей ошибке:

```
switch(error.code)
{
  case 1: message = 'Доступ запрещен'; break;
  case 2: message = 'Позиция недоступна'; break;
  case 3: message = 'Время ожидания операции истекло'; break;
  case 4: message = 'Неизвестная ошибка'; break;
}

alert("Ошибка геолокации: " + message)
```

Когда браузер запрашивает у хоста данные геолокации, он спрашивает разрешение у пользователя, которое тот может дать или не дать. Отказ приводит к состоянию «Доступ запрещен», когда же пользователь дал разрешение, но хост-система не в состоянии определить его местонахождение, возникает состояние «Позиция недоступна», а когда есть разрешение пользователя и хост пытается получить его позицию, но время выполнения запроса истекает, складывается ситуация «Время ожидания операции истекло».

Существует еще одно условие возникновения ошибки, при котором некоторые сочетания платформ и браузеров позволяют пользователю отклонить диалог запроса разрешения без предоставления этого разрешения или отказа в его предоставлении. В результате чего приложение на время ожидания обратного вызова «зависает».

В предыдущих редакциях этой книги для встраивания карты непосредственно в веб-страницу использовалось обращение к Google Maps API, но теперь этот сервис требует от вас самих предоставления уникального ключа API, и для пользования сервисом в определенном объеме требуется оплата. Поэтому теперь в примере просто создается ссылка на Google Maps. Если же требуется вставить Google Maps в ваши веб-страницы или веб-приложения, все необходимые сведения можно найти на сайте по адресу https://developers.google.com/maps/. Разумеется, существует также множество других инструментальных средств вывода карт на страницу, например Bing Maps (https://www.bing.com/maps) и OpenStreetMap (https://www.openstreetmap.org), у которых имеются доступные вам API.

Локальное хранилище

Cookie являются неотъемлемой частью современного Интернета, потому что позволяют сайтам сохранять на каждой пользовательской машине небольшие фрагменты информации, которые могут применяться для отслеживания действий пользователя. Теперь это не воспринимается столь же зловеще, как звучит, поскольку в большинстве случаев проводимое отслеживание помогает пользователям, сохраняя имена и пароли, избавляя от необходимости регистрироваться на сайтах и в таких социальных сетях, как Twitter, Facebook и т. д.

Cookie позволяют также сохранять на локальной машине ваши предпочтения при обращении к сайту (вместо того чтобы хранить эти предпочтения на сервере сайта) или могут использоваться для отслеживания наполнения товарной корзины при формировании заказа на сайте электронной торговли.

Конечно, они также могут использоваться более агрессивно — для отслеживания того, какие сайты посещаются чаще всего, составляя представление о ваших интересах, что позволит эффективнее направлять рекламу. Вот почему Европейский союз ввел положение о том, что «требуется предварительное информированное согласие на хранение информации, находящейся на конечном оборудовании

пользователя или на доступ к такой информации» (http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm).

Теперь подумайте с точки зрения веб-разработчика, насколько полезно может быть хранение данных на пользовательских устройствах, особенно если у вас небольшой ресурс компьютерных серверов и дискового пространства. Например, можно создавать браузерные веб-приложения и службы для редактирования текстовых документов, электронных таблиц и графических изображений, сохраняя все данные удаленно на пользовательских компьютерах, выдерживая закупочный бюджет своего сервера как можно ниже.

С пользовательской точки зрения подумайте, насколько быстрее может загружаться документ локально по сравнению с загрузкой из Интернета, особенно при медленном подключении. Кроме того, вам будет намного спокойнее, если вы будете знать, что на сайте не хранятся копии ваших документов. Разумеется, полную безопасность сайта или веб-приложения гарантировать невозможно, и вы никогда не будете работать над сугубо конфиденциальными документами, используя программы (или оборудование), которые могут входить в Сеть.

В отношении документов, носящих сугубо личный характер, таких как семейные фотографии, возможно, будет комфортнее пользоваться веб-приложением, которое хранит данные локально, а не на внешнем сервере.

Использование локального хранилища

Самая большая проблема при использовании cookie в качестве локального хранилища заключается в том, что в каждом из них можно хранить максимум 4 Кбайт данных. Cookie также должны курсировать в обоих направлениях при каждой перезагрузке страницы. И, если только ваш сервер не использует шифрование на уровне передачи данных — Transport Layer Security (TLS), являющееся более безопасным потомком SSL-шифрования (Secure Sockets Layer), при каждой передаче cookie путешествуют в открытом виде.

Но с появлением HTML5 у вас появляется доступ к намного более объемному локальному хранилищу (обычно, в зависимости от браузера, между 5 и 10 Мбайт на каждый домен), которое сохраняет информацию между загрузками страницы и посещениями сайта (даже после выключения и включения компьютера). Кроме того, данные локального хранилища не отправляются на сервер при каждой загрузке страницы.

Эти данные хранятся в парах «ключ — значение». Ключ является именем, присваиваемым для ссылки на данные, а значение может содержать любой тип данных, но сохраняется в виде строки. Все данные для текущего домена уникальны, и из соображений безопасности любое локальное хранилище, созданное сайтами из других доменов, обособляется от текущего локального хранилища, которое становится недоступным любому домену, отличающемуся от того, что сохраняет данные.

Объект localStorage

Доступ к локальному хранилищу можно получить с помощью объекта `localStorage`. Чтобы проверить доступность этого объекта, запрашивается его тип, позволяющий понять, был ли он определен:

```
if (typeof localStorage == 'undefined')
{
  // Локальное хранилище недоступно, нужно сообщить об этом пользователю
  // и завершить работу. Или же вместо этого предложить сохранить данные
  // на веб-сервере.
}
```

Можно ли обойтись без доступа к локальному хранилищу, будет зависеть от предполагаемых целей его использования, поэтому код, помещаемый в инструкцию `if`, зависит от вас.

Убедившись в доступности локального хранилища, можно приступить к его использованию с помощью методов `setItem` и `getItem` объекта `localStorage`:

```
localStorage.setItem('loc', 'USA')
localStorage.setItem('lan', 'English')
```

Чтобы впоследствии просмотреть эти данные, необходимо передать ключ методу getItem:

```
loc = localStorage.getItem('loc')
lan = localStorage.getItem('lan')
```

В отличие от сохранения и чтения cookie, эти методы можно вызвать в любое время, без необходимости предварительной отправки веб-сервером каких-нибудь заголовков. Сохраненные значения будут оставаться в локальном хранилище до тех пор, пока не будут уничтожены следующим образом:

```
localStorage.removeItem('loc')
localStorage.removeItem('lan')
```

Или же можно полностью уничтожить локальное хранилище для текущего домена, вызвав метод `clear`:

```
localStorage.clear()
```

В примере 26.2 предыдущие примеры объединены в один документ, показывающий текущие значения двух ключей в появляющемся окне предупреждения, которое изначально будет иметь значение `null`. Затем ключи и значения сохраняются в локальном хранилище, извлекаются из него и заново показываются, на этот раз имея присвоенные значения. И наконец, ключи удаляются, а затем предпринимается попытка повторного извлечения значений, но возвращаемые значения снова равны `null`.

Второе из этих предупреждений продемонстрировано на рис. 26.2.

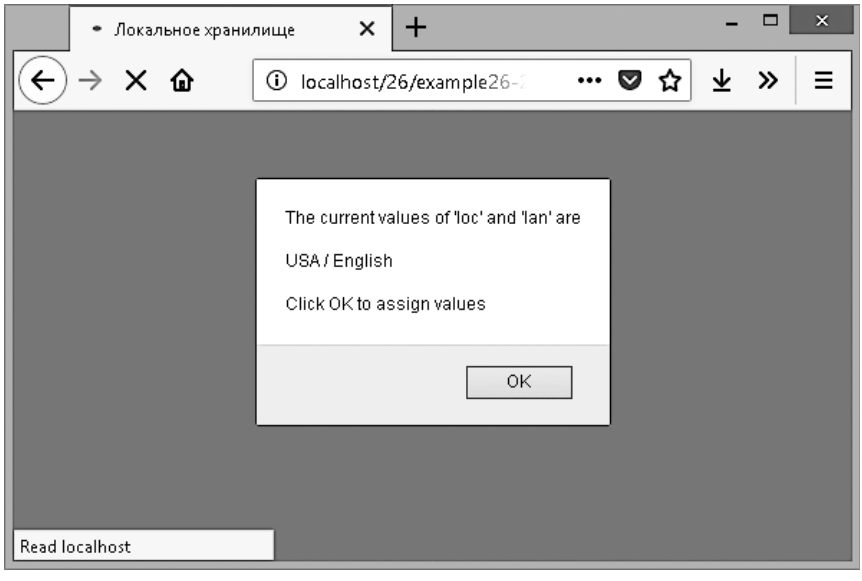

Рис. 26.2. Чтение из локального хранилища двух ключей и их значений

Пример 26.2. Получение, установка и удаление данных локального хранилища

```
<!DOCTYPE html>
<html>
  <head>
    <title>Локальное хранилище</title>
  </head>
  <body>
    <script>
      if (typeof localStorage == 'undefined')
      {
        alert("Локальное хранилище недоступно")
      }
      else
      {
       loc = localStorage.getItem('loc')
       lan = localStorage.getItem('lan')
       alert("Текущие значения 'loc' и 'lan':\n\n" +
         loc + " / " + lan + "\n\nНажмите кнопку OK,
         чтобы присвоить значения")

       localStorage.setItem('loc', 'USA')
       localStorage.setItem('lan', 'English')
       loc = localStorage.getItem('loc')
       lan = localStorage.getItem('lan')
       alert("Текущие значения 'loc' и 'lan':\n\n" +
```

```
        loc + " / " + lan + "\n\nНажмите кнопку OK,
        чтобы очистить значения")

      localStorage.removeItem('loc')
      localStorage.removeItem('lan')
      loc = localStorage.getItem('loc')
      lan = localStorage.getItem('lan')
      alert("Текущие значения 'loc' и 'lan':\n\n" +
        loc + " / " + lan)
    }
  </script>
 </body>
</html>
```

В локальное хранилище можно включать данные практически любого типа и сколько угодно пар «ключ — значение» вплоть до достижения доступного лимита хранения для вашего домена.

Рабочие веб-процессы

Рабочие веб-процессы запускают на выполнение фоновые задания и хорошо подходят для длительных вычислений, которым не следует мешать пользователям заниматься другими делами. Чтобы воспользоваться рабочим веб-процессом, можно создать разделы кода JavaScript, запускаемые в фоновом режиме. В этом коде не должны устанавливаться и отслеживаться прерывания, как это приходится делать заданиям в некоторых асинхронных системах. Вместо этого при наличии чего-то, о чем нужно сообщить, ваш фоновый процесс связывается с основным кодом JavaScript путем выдачи события.

Это означает, что решать, как наиболее эффективно распределить отрезки времени, будет интерпретатор JavaScript, а вашему коду останется только позаботиться о связи с фоновой задачей при наличии передаваемой информации.

В примере 26.3 показывается, как рабочие веб-процессы можно настроить на вычисление повторяющейся задачи в фоновом режиме: в данном случае на вычисление простых чисел.

Пример 26.3. Настройка рабочего веб-процесса и обмен данными с ним

```
<!DOCTYPE html>
<html>
  <head>
    <title>Рабочие веб-процессы</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    Текущее самое большое простое число:
```

```
    <span id='result'>0</span>

    <script>
      if (!!window.Worker)
      {
        var worker = new Worker('worker.js')

        worker.onmessage = function (event)
        {
          O('result').innerText = event.data;
        }
      }
      else
      {
        alert("Рабочие веб-процессы не поддерживаются")
      }
    </script>
  </body>
</html>
```

В этом примере сначала создается элемент `<span>` с идентификатором `result`, в который будут помещены выходные данные от рабочего веб-процесса. Затем в разделе `<script>` с помощью пары операторов НЕ (!!) тестируется `window.Worker`. В результате, если метод `Worker` существует, возвращается логическое значение `true`, в противном случае возвращается `false`. Если это значение не равно `true`, то сообщение, выводимое в разделе `else`, предупреждает нас о том, что рабочие веб-процессы недоступны.

В противном случае вызовом метода `Worker` создается новый объект `worker`, которому передается имя файла `worker.js`. Затем к безымянной функции, которая помещает любые переданные ей сценарием `worker.js` сообщения в свойство `innerText` ранее созданного элемента `<span>`, прикрепляется событие `onmessage` нового объекта `worker`.

Код самого рабочего веб-процесса сохраняется в файле `worker.js`, который приведен в примере 26.4.

Пример 26.4. Рабочий веб-процесс worker.js

```
var n = 1

search: while (true)
{
  n += 1

  for (var i = 2; i <= Math.sqrt(n); i += 1)
  {
    if (n % i == 0) continue search
  }

  postMessage(n)
}
```

В этом файле переменной n присваивается значение 1. Затем запускается бесконечный цикл с увеличением значения n на единицу и проверкой значения «в лоб» на принадлежность к простым числам. При этом тестируются все значения от 1 до корня квадратного из n для проверки, делится ли n на них без остатка. Если сомножитель будет найден, то команда continue тут же останавливает лобовую атаку, поскольку число не является простым, и начинает обработку снова в отношении следующего более высокого значения n.

Но если все возможные сомножители будут протестированы и не будет найден результат с нулевым остатком, то n должно быть простым числом, следовательно, его значение передается функции postMessage, отправляющей сообщение событию onmessage объекта, который установил этот рабочий веб-процесс.

Результат выглядит следующим образом:

```
Текущее самое большое простое число: 30477191
```

Чтобы остановить выполнение рабочего веб-процесса, нужно вызвать метод terminate объекта worker:

```
worker.terminate()
```

Если нужно остановить выполнение процесса в данном конкретном примере, в адресной строке браузера можно ввести такой код:

```
javascript:worker.terminateO
```

Следует также заметить, что из-за способа решения в Chrome вопросов безопасности использовать рабочие веб-процессы в отношении файловой системы невозможно, их можно запускать только с веб-сервера (или запускать файлы из localhost на таких серверах разработки, как AMPPS, подробно рассмотренный в главе 2).

У рабочих веб-процессов имеется ряд ограничений, связанных с соблюдением мер безопасности, о которых следует знать.

- Рабочие веб-процессы запускаются в своем собственном независимом контексте JavaScript и не имеют непосредственного доступа к чему-либо в любом другом контексте выполнения кода, включая основной поток JavaScript или другие рабочие веб-процессы.
- Обмен данными между контекстами рабочих веб-процессов осуществляется посредством рассылки веб-сообщений (postMessage).
- Поскольку у рабочих веб-процессов доступа к контексту основного кода JavaScript не имеется, они не могут вносить изменения в DOM. Единственными DOM-методами, доступными рабочим процессам, являются atob, btoa, clearInterval, clearTimeout, dump, setInterval и setTimeout.
- Рабочие веб-процессы ограничены политикой единого домена, поэтому без прохождения методов безопасного обмена данными между сайтами, загрузить рабочий веб-процесс из другого домена, кроме исходного, невозможно.

Перетаскивание

Как показано в примере 26.5, поддержку перетаскивания на веб-странице можно легко организовать, добавив обработчики для событий `ondragstart`, `ondragover` и `ondrop`.

Пример 26.5. Объекты перетаскивания

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Перетаскивание</title>
    <script src='OSC.js'></script>
    <style>
      #dest {
        background:lightblue;
        border    :1px solid #444;
        width     :320px;
        height    :100px;
        padding   :10px;
      }
    </style>
  </head>
  <body>
    <div id='dest' ondrop='drop(event)' ondragover='allow(event)'></div><br>
    Перетащите нижнее изображение в верхний элемент <br><br>

    <img id='source1' src='image1.png' draggable='true' ondragstart='drag(event)'>
    <img id='source2' src='image2.png' draggable='true' ondragstart='drag(event)'>
    <img id='source3' src='image3.png' draggable='true' ondragstart='drag(event)'>
    <script>
      function allow(event)
      {
        event.preventDefault()
      }

      function drag(event)
      {
        event.dataTransfer.setData('image/png', event.target.id)
      }

      function drop(event)
      {
        event.preventDefault()
        var data=event.dataTransfer.getData('image/png')
        event.target.appendChild(O(data))
      }
    </script>
  </body>
</html>
```

После указания тегов `html`, `title` и загрузки файла `OSC.js` в этом документе задается стиль `<div>`-элементу с идентификатором `dest`. Для него устанавливаются цвет фона, граница, размеры и отступы.

Затем в разделе `<body>` создается `<div>`-элемент и объявляются события `ondrop` и `ondragover` с прикрепленными к ним функциями-обработчиками `drop` и `allow`.

После этого следует текст, а за ним выводятся три изображения, у которых для их свойств `draggable` установлено значение `true`, а к событию `ondragstart` каждого из них прикрепляется функция `drag`.

В разделе `<script>` функция `allow`, выполняющая роль обработчика события, просто предотвращает для перетаскивания выполнение действия по умолчанию (запрещая его). Функция `drag`, выполняющая такую же роль, вызывает метод `setData` объекта `dataTransfer`, передавая ему MIME-тип `image/png` и `target.id` события. Объект `dataTransfer` содержит перетаскиваемые данные в ходе самой операции перетаскивания.

И наконец, функция `drop`, выполняющая роль обработчика события, также перехватывает действие по умолчанию, разрешая завершить перетаскивание, а затем извлекает содержимое перетаскиваемого объекта из объекта `dataTransfer`, передавая при этом MIME-тип объекта. Затем освобождаемые после перетаскивания данные присоединяются к цели (к `<div>`-элементу с идентификатором `dest`), используя его метод `appendChild`.

Если проверить работу этого примера, можно будет перетаскивать изображения в `<div>`-элемент, где они и будут оставаться (рис. 26.3). Изображения не могут быть перетащены никуда, кроме элементов, к которым прикреплены обработчики событий `drop` и `allow`.

Рис. 26.3. Два изображения здесь уже были перетащены

Можно прикрепить и другие события, включающие `ondragenter` (для запуска в тех случаях, когда при операции перетаскивания происходит вход указателя мыши в элемент), `ondragleavefor` (для запуска в тех случаях, когда указатель мыши покидает элемент) и `ondragendfor` (для запуска в тех случаях, когда операция перетаскивания завершается). Эти события можно использовать, к примеру, для изменения внешнего вида указателя мыши в ходе таких операций.

Обмен сообщениями между документами

Вы уже видели обмен сообщениями немного ранее, в разделе о рабочих веб-процессах. Но в подробности там я не вдавался, поскольку веб-процессы не относились к основной рассматриваемой теме и сообщения в любом случае передавались одному и тому же документу. Однако по вполне понятным соображениям безопасности обмен сообщениями между документами должен применяться осмотрительно, то есть, если вы планируете его использование, необходимо понимать, как он работает.

До появления HTML5 разработчики браузеров запрещали междокументное выполнение сценариев, но наряду с блокировкой потенциальных вредоносных сайтов запрещалась и связь между вполне законными страницами, что вынудило осуществлять взаимодействия любого рода посредством асинхронного обмена данными и сторонних веб-серверов, создавая неоправданные сложности и неудобства при создании и поддержке.

Но теперь веб-обмен сообщениями позволяет сценариям взаимодействовать, преодолевая эти границы при соблюдении разумных ограничительных мер безопасности, направленных на предотвращение вредоносных попыток взлома. Это достигается путем использования метода `postMessage`, позволяющего отправлять простые текстовые сообщения от одного домена другому, но неизменно в одном браузере.

Для этого требуется, чтобы JavaScript сначала приобрел объект `window` получаемого документа, допуская отправку сообщений различным окнам, тегам `frame` или `iframe`, непосредственно связанным с документом отправителя. Событие полученного сообщения имеет следующие атрибуты:

- `data` — входящее сообщение;
- `origin` — происхождение отправителя документа, включая схему, имя хоста и порт;
- `source` — исходное окно отправителя документа.

Как показано в примере 26.6, код для отправки сообщений представляет собой одну инструкцию, в которую передается отправляемое сообщение, и домен, к которому оно относится.

Пример 26.6. Отправка веб-сообщений в iframe

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Веб-сообщения (a)</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <iframe id='frame' src='example26-6.html' width='360'
      height='75'></iframe>

    <script>
      count = 1

      setInterval(function()
      {
        O('frame').contentWindow.postMessage('Message ' + count++, '*')
      }, 1000)
    </script>
  </body>
</html>
```

Здесь, как обычно, используется файл `OSC.js`, чтобы можно было взять из него функцию `O`. Затем создается элемент `<iframe>` с идентификатором `frame`, который загружает код примера 26.7 (`example26-7.html`). После этого в разделе `<script>` переменной `count` присваивается начальное значение `1` и устанавливается интервал в одну секунду для повторяющейся отправки строки `'Message '` (с использованием метода `postMessage`) наряду с текущим значением счетчика, значение которого после этого увеличивается на единицу. Вызов `postMessage` прикреплен к свойству `contentWindow` объекта `iframe`, а не к самому объекту `iframe`. Это важно, потому что обмен веб-сообщениями требует, чтобы сообщения отправлялись в окно, а не в объект в окне.

Пример 26.7. Получение сообщений от другого документа

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Веб-сообщения (b)</title>
    <style>
      #output {
        font-family:"Courier New";
        white-space:pre;
      }
    </style>
    <script src='OSC.js'></script>
  </head>
  <body>
    <div id='output'>Полученные сообщения будут отображаться здесь</div>

    <script>
      window.onmessage = function(event)
      {
        O('output').innerHTML =
```

```
          '<b>Origin:</b> ' + event.origin + '<br>' +
          '<b>Source:</b> ' + event.source + '<br>' +
          '<b>Data:</b>   ' + event.data
      }
    </script>
  </body>
</html>
```

В этом примере производятся незначительные стилевые настройки, делающие вывод более выразительным, затем создается `<div>`-элемент с идентификатором `output`, в который будет помещено содержимое полученного сообщения. В разделе `<script>` находится одна безымянная функция, прикрепленная к событию `onmessage` объекта `window`. В этой функции, как показано на рис. 26.4, выводятся на экран значения свойств `event.origin`, `event.source` и `event.data`.

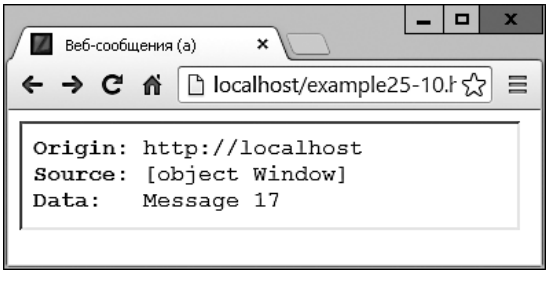

Рис. 26.4. На данный момент iframe получил 29 сообщений

Обмен веб-сообщениями работает только между доменами, поэтому его нельзя протестировать, загружая файлы из файловой системы, и нужно воспользоваться веб-сервером. На рис. 26.4 показано, что источником является http://localhost, поскольку эти примеры запущены на локальном сервере, предназначенном для разработки. Источником является объект `window`, а текущим сообщением — `Message 29`.

На данный момент пример 26.6 нельзя признать полностью безопасным, так как в качестве значения домена, переданного `postMessage`, используется групповой символ *:

```
O('frame').contentWindow.postMessage('Message ' + count++, '*')
```

Чтобы направить сообщения только тем документам, источником которых является конкретный домен, этот параметр можно изменить. В данном случае значение http://localhost обеспечит отправку сообщений только тем документам, которые были загружены с локального сервера:

```
O('frame').contentWindow.postMessage('Message ' + count++, 'http://localhost')
```

Более того, в подобных условиях прослушивающая программа выводит абсолютно все получаемые сообщения. Такое положение дел также не отличается высоким уровнем безопасности, поскольку вредоносные документы, присутствующие в браузере, могут осуществить попытку отправки сообщений, к которым, если

не предпринять мер защиты, может быть получен доступ со стороны неосмотрительно написанного кода, прослушивающего сообщения. Но вы можете ограничить круг сообщений, на которые реагирует прослушивающий код, воспользовавшись инструкцией `if`:

```
window.onmessage = function(event)
{
  if (event.origin) == 'http://localhost')
  {
    O('output').innerHTML =
      '<b>Origin:</b> ' + event.origin + '<br>' +
      '<b>Source:</b> ' + event.source + '<br>' +
      '<b>Data:</b>   ' + event.data
  }
}
```

Если для сайта, с которым ведется работа, всегда используется надлежащий домен, ваш обмен веб-сообщениями будет безопаснее. Тем не менее следует помнить, что, поскольку сообщения отправляются в открытом виде, в отношении некоторых браузеров или их дополнительных модулей могут быть сомнения насчет защищенности этой разновидности обмена данными. Тогда одним из способов повышения безопасности будет использование системы запутывания или шифрования, а также рассмотрение вопроса введения своих собственных двусторонних протоколов связи для проверки подлинности каждого сообщения.

Как правило, значения `origin` и `source` пользователю не показываются, они используются лишь для проверок, связанных с обеспечением безопасности. Но в этих примерах они видны, чтобы помочь вам провести эксперименты с обменом веб-сообщениями и разобраться в том, что происходит. С помощью этого метода документы в появляющихся окнах и других вкладках могут обмениваться данными, не прибегая к использованию `iframes`.

Микроданных больше нет

В предыдущих изданиях этой книги рассматривались приемы использования интересной новой функции HTML5 под названием «микроданные», представляющей собой поднабор HTML, разработанный с целью предоставления документу метаданных, для придания различным элементам данных дополнительного значения с точки зрения программных средств и таких поисковых роботов, как Google.

Но время не стоит на месте, и вместо того, чтобы эту функцию принимало все больше и больше браузеров, обе компании, и Google, и Apple отказались от ее поддержки, и она была исключена из основной HTML5-спецификации в W3C.

Поэтому я больше не рекомендую помечать какой-либо код микроданными, поскольку он станет выглядеть вышедшим из употребления или похожим на таковой, и даже если поисковые роботы Google и Bing все еще используют какую-либо часть этой информации, то в скором времени, они, вероятнее всего, прекратят к ней прибегать.

Другие теги HTML5

В HTML5 появилось несколько других новых тегов, внедренных во все основные браузеры, это: `<article>`, `<aside>`, `<details>`, `<figcaption>`, `<figure>`, `<footer>`, `<header>`, `<hgroup>`, `<mark>`, `<menuitem>`, `<meter>`, `<nav>`, `<output>`, `<progress>`, `<rp>`, `<rt>`, `<ruby>`, `<section>`, `<summary>`, `<time>` и `<wbr>`. Дополнительную информацию об этих и других тегах HTML5 можно получить по адресу http://bit.ly/2I6CkrP (ищите элементы, помеченные значком NEW).

На этом введение в HTML5 завершается. Теперь в вашем распоряжении имеется множество новых эффективных средств, придающих сайтам еще больше динамичности и привлекательности. В заключительной главе будет показано, как можно собрать воедино все описанные в книге различные технологии для создания небольшого сайта социальной сети.

Вопросы

1. Какой метод нужно вызвать для запроса геолокационных данных у браузера?
2. Как определить, поддерживает ли браузер локальное хранилище?
3. Какой метод можно вызвать, чтобы удалить все данные локального хранилища для текущего домена?
4. Какой способ считается наилучшим для связи рабочих веб-процессов с основной программой?
5. Каким образом можно воспрепятствовать запуску рабочего веб-процесса?
6. Как предотвратить действие по умолчанию, не позволяющее перетаскивание для событий, и в результате обеспечить поддержку операций перетаскивания?
7. Как сделать обмен сообщениями между документами более безопасным?

Ответы на эти вопросы можно найти в приложении А, в соответствующем разделе.

27 Объединение технологий

В завершение книги я хочу привести вам реальный пример использования рассмотренных технологий, в котором вы можете досконально разобраться. В действительности это несколько примеров, объединенных в простом проекте социальной сети, имеющей все атрибуты, которые ожидаются на подобном сайте или даже в веб-приложении.

В разных файлах проекта представлены примеры создания таблиц MySQL и доступа к базе данных, таблиц стилей CSS, включения других файлов, управления сессией, доступа к DOM, асинхронных вызовов, обработки событий и ошибок, загрузки файлов на сервер, работы с изображениями, холстами HTML5 и решения многих других задач.

Каждый файл, приводимый в качестве примера, является завершенной и самодостаточной программой, способной работать совместно с остальными файлами с целью построения полностью работоспособного сайта социальной сети. К тому же, подключив таблицу стилей, вы сможете полностью изменить внешний вид проекта. Благодаря небольшому объему и несложной конструкции конечный продукт будет особенно полезен на мобильных платформах, таких как смартфоны или планшеты, но он также хорошо будет работать и на полноразмерных настольных компьютерах.

Вы можете убедиться в том, что, задействуя всю мощь jQuery и jQuery Mobile, код работает довольно быстро, отличается простотой использования, хорошей адаптированностью ко всем средам и хорошо смотрится.

И все же я постарался максимально ужать код, чтобы он легче усваивался. Исходя из этого, имеются весьма широкие возможности по его улучшению, например в целях повышения безопасности путем хранения хешей вместо незашифрованных паролей и более плавных переходов между входом в систему и выходом из нее, но

давайте оставим все это в качестве пресловутых упражнений для читателя, тем более что в конце этой главы ему не задается никаких вопросов!

Можно взять любой представляющийся полезным фрагмент кода и дополнить его в соответствии с поставленными задачами. Возможно, у вас даже появится желание создать на основе этих файлов собственную социальную сеть.

Проектирование приложения социальной сети

Перед написанием кода я определяю для себя важные составляющие подобного приложения, среди которых:

- процесс регистрации;
- форма для входа на сайт;
- средство для завершения работы с сайтом;
- управление сессией;
- пользовательские профили с загруженными миниатюрными изображениями;
- каталог участников сети;
- добавление участников в список друзей;
- открытый и закрытый обмен сообщениями между участниками;
- способ стилевого оформления проекта.

Я решил назвать проект «Сообщество Робина» — Robin's Nest. Если же вы воспользуетесь этим кодом, нужно будет изменить имя и логотип в файлах `index.php` и `header.php`.

Информация на сайте

Все примеры, приводимые в данной главе, можно найти на прилагаемом к книге сайте http://lpmj.net. Можно также загрузить примеры с этого сайта на свой компьютер, щелкнув на ссылке 5th Edition Examples (Примеры пятого издания) в верхней части страницы, и будет загружен архивный файл, который можно будет распаковать в удобное место на вашем компьютере.

С учетом того что данная глава представляет особый интерес, внутри ZIP-файла есть папка robinsnest, в которой все следующие примеры сохранены с использованием тех имен, что требуются этому учебному приложению. Поэтому вы можете просто скопировать все эти примеры в свою папку разработки веб-приложения, чтобы увидеть их в действии.

Файл functions.php

Перейдем непосредственно к проекту и начнем с примера 27.1, `functions.php`, который включает в себя основные функции. Но в этом файле содержатся не только функции. Я добавил в него сведения, необходимые для входа в базу данных, чтобы не использовать для этой цели лишний файл. В первых четырех строках кода определяются хост, имя базы данных, а также имя пользователя и пароль.

Изначально в качестве имени пользователя базы данных MySQL используется `robinsnest`, а база данных, используемая программой, также называется `robinsnest`. Имя пользователя базы данных MySQL, поскольку оно уже существует, не имеет значения, равно как и имя самой базы данных. В главе 8 предоставляются подробные инструкции по созданию нового пользователя и (или) базы данных. Но, по сути, при наличии соответствующих полномочий новую базу данных по имени `robinsnest` можно создать из командной строки MySQL:

```
CREATE DATABASE robinsnest;
```

Затем, также при наличии соответствующих полномочий, можно создать пользователя по имени `robinsnest`, имеющего доступ к этой базе данных:

```
GRANT ALL ON robinsnest.* TO 'robinsnest'@'localhost'
  IDENTIFIED BY 'rnpassword';
```

Несомненно, вы воспользуетесь более стойким паролем для этого пользователя, чем `rnpassword`, но, чтобы ничего не усложнять, в приводимых здесь примерах используется именно этот пароль, нужно будет только не забыть его изменить в том случае, если вы воспользуетесь какими-то фрагментами данного кода на производственном сайте (или же, как уже упоминалось, можно будет воспользоваться ранее существовавшими именами пользователя и базы данных).

Если задать правильные значения в следующих двух строках файла, будет открыто подключение к MySQL и выбрана база данных.

Функции. В проекте используются пять основных функций:

- `createTable` — проверяет факт существования таблицы и создает отсутствующую таблицу;
- `queryMysql` — выдает запрос к MySQL, а при сбое выводит сообщение об ошибке;
- `destroySession` — уничтожает PHP-сессию и очищает машину от ее данных для завершения сеанса работы пользователей;
- `sanitizeString` — удаляет потенциально вредный код или теги из информации, введенной пользователем;
- `showProfile` — отображает миниатюрные изображения пользователей и их записи About me (Обо мне), если таковые имеются.

Работа всех этих функций должна быть вам понятна. За исключением, может быть, функции `showProfile`, которая осуществляет поиск изображения по имени `user.jpg` (где `user` — это пользовательское имя текущего пользователя) и после успешного поиска выводит его на экран. Она также отображает любой текст About me (Обо мне), который пользователь мог сохранить.

Все нуждающиеся в этом функции снабжены кодом обработки ошибок, который позволяет перехватывать любую опечатку или другие допущенные ошибки ввода и сгенерировать сообщение об ошибке. Но если какая-нибудь из этих функций используется на рабочем сервере, то вам, скорее всего, захочется предоставить для этой цели собственные обработчики ошибок, чтобы сделать код более дружелюбным по отношению к пользователю.

Наберите код примера 27.1 и сохраните его в файле `functions.php` (или загрузите файл с прилагаемого к книге сайта), после чего вы будете готовы перейти к изучению следующего раздела.

Пример 27.1. functions.php

```
<?php
  $dbhost = 'localhost';  // Эта строка вряд ли нуждается в  изменении
  $dbname = 'robinsnest'; // А значения этих переменных
  $dbuser = 'robinsnest'; // поменяйте на те, что соответствуют
  $dbpass = 'rnpassword'; // вашим настройкам

  $connection = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($connection->connect_error) die("Fatal Error");

  function createTable($name, $query)
  {
    queryMysql("CREATE TABLE IF NOT EXISTS $name($query)");
    echo "Таблица '$name' создана или уже существовала<br>";
  }

  function queryMysql($query)
  {
    global $connection;
    $result = $connection->query($query);
    if (!$result) die("Fatal Error");
    return $result;
  }

  function destroySession()
  {
    $_SESSION=array():
    if (session_id() != "" || isset($_COOKIE[session_name()]))
      setcookie(session_name(), '', time()-2592000, '/'):

    session_destroy();
  }

  function sanitizeString($var)
  {
    global $connection;
```

```
    $var = strip_tags($var);
    $var = htmlentities($var);
    if (get_magic_quotes_gpc())
      $var = stripslashes($var);
    return $connection->real_escape_string($var);
  }

  function showProfile($user)
  {
    if (file_exists("$user.jpg"))
    echo "<img src='$user.jpg' align='left'>";

    $result = queryMysql("SELECT * FROM profiles WHERE user='$user'");

    if ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_ASSOC);
      echo stripslashes($row['text']) . "<br style='clear:left;'><br>";
    }
    else echo "<p>Здесь пока не на что смотреть </p><br>";
  }
?>
```

Если вы читали предыдущее издание этой книги, в котором в этих примерах указывалось старое mysql-расширение, то должны были заметить, что для ссылки на базу данных MySQL с использованием mysqli в функциях queryMysql и sanitizeString следует применять ключевое слово global, чтобы позволить им использовать значение, находящееся в $connection.

Файл header.php

Чтобы сохранить единство стиля, каждая страница проекта должна иметь доступ к одному и тому же набору функций. Поэтому я поместил соответствующие настройки в файл `header.php`, показанный в примере 27.2. Этот файл включается практически во все остальные файлы, и он, в свою очередь, включает в себя файл `functions.php`. Это означает, что в каждом файле нужна только одна инструкция `require_once`.

Код файла `header.php` начинается с вызова функции `session_start`. Из материалов главы 12 следует, что эта функция настраивает сессию, которая будет запоминать конкретные значения, необходимые для использования в различных PHP-файлах. Иными словами, в сессии отображается визит пользователя на сайт, и у нее может истечь срок действия, если пользователь игнорирует сайт определенный период времени.

С началом сессии программа выводит код HTML, необходимый для настройки каждой веб-страницы, включая загрузку таблиц стилей и различные нужные библиотеки JavaScript. Затем в код включается файл функций (`functions.php`),

и переменной $userstr присваивается исходное строковое значение Welcome Guest («Добро пожаловать, гость»).

Затем программа проверяет, присвоено ли значение переменной сессии user. Если оно присвоено, значит пользователь уже вошел на сайт и значение переменной $loggedin установлено как TRUE, а имя пользователя извлечено из переменной сессии user в PHP-переменную $user с соответствующим обновлением значения переменной $userstr. Если пользователь еще не вошел на сайт, переменной $loggedin присваивается значение FALSE.

После этого выводится код HTML с приветствием пользователю (или гостю, если вход еще не был осуществлен), и -элементы, требуемые jQuery Mobile для заголовка страницы и разделов содержимого. Затем благодаря использованию значения переменной $loggedin и блока if отображается один из двух наборов меню. Набор для не вошедшего на сайт пользователя предлагает выбор только из главной страницы — Home, регистрации — Sign up и входа на сайт — Log in, а версия меню для вошедших на сайт предлагает полный доступ к функциям проекта. Кнопки получают стилевое оформление с использованием формы записи библиотеки jQuery Mobile, например data-role='button', для отображения элемента в виде кнопки, data-inline='true' для отображения элементов в ряд (действует подобно элементу) и data-transition="slide", чтобы заставить новые страницы наплывать на окно просмотра при щелчке, согласно описанию, которое было дано в главе 22.

Дополнительное стилевое оформление, применяемое к этому файлу, находится в файле styles.css (см. пример 27.13, рассматриваемый в конце главы).

Пример 27.2. header.php

```
<?php //
  session_start();

echo <<<_INIT
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8'>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' href='jquery.mobile-1.4.5.min.css'>
    <link rel='stylesheet' href='styles.css'>
    <script src='javascript.js'></script>
    <script src='jquery-2.2.4.min.js'></script>
    <script src='jquery.mobile-1.4.5.min.js'></script>

_INIT;

  require_once 'functions.php';

  $userstr = 'Welcome Guest';

  if (isset($_SESSION['user']))
  {
```

```
    $user = $_SESSION['user'];
    $loggedin = TRUE;
    $userstr = "Logged in as: $user";
  }
  else $loggedin = FALSE;

echo <<<_MAIN
    <title>Robin's Nest: $userstr</title>
  </head>
  <body>
    <div data-role='page'>
      <div data-role='header'>
        <div id='logo'
          class='center'>R<img id='robin' src='robin.gif'>bin's Nest</div>
        <div class='username'>$userstr</div>
      </div>
      <div data-role='content'>

_MAIN;

  if ($loggedin)
  {
echo <<<_LOGGEDIN
    <div class='center'>
      <a data-role='button' data-inline='true' data-icon='home'
        data-transition="slide" href='members.php?view=$user'>Home</a>
      <a data-role='button' data-inline='true'
        data-transition="slide" href='members.php'>Members</a>
      <a data-role='button' data-inline='true'
        data-transition="slide" href='friends.php'>Friends</a>
      <a data-role='button' data-inline='true'
        data-transition="slide" href='messages.php'>Messages</a>
      <a data-role='button' data-inline='true'
        data-transition="slide" href='profile.php'>Edit Profile</a>
      <a data-role='button' data-inline='true'
        data-transition="slide" href='logout.php'>Log out</a>
    </div>

_LOGGEDIN;
  }
  else
  {
echo <<<_GUEST
    <div class='center'>
      <a data-role='button' data-inline='true' data-icon='home'
        data-transition='slide' href='index.php'>Home</a>
      <a data-role='button' data-inline='true' data-icon='plus'
        data-transition="slide" href='signup.php'>Sign Up</a>
      <a data-role='button' data-inline='true' data-icon='check'
        data-transition="slide" href='login.php'>Log In</a>
    </div>
    <p class='info'>(You must be logged in to use this app)</p>

_GUEST;
  }
?>
```

Файл setup.php

После создания двух включаемых файлов настала очередь настройки используемых ими MySQL-таблиц. Это делается с помощью кода из примера 27.3, `setup.php`, который следует набрать и загрузить в свой браузер до вызова любых других файлов. В противном случае будут получены многочисленные сообщения об ошибках MySQL.

Создаваемые таблицы весьма лаконичны и имеют следующие имена и столбцы:

- `members` — имя пользователя `user` (проиндексированный столбец), пароль `pass`;
- `messages` — идентификатор `id` (проиндексированный столбец), автор `auth` (проиндексированный столбец), адресат `recip`, тип сообщения `pm`, сообщение `message`;
- `friends` — имя пользователя `user` (проиндексированный столбец), имя пользователей-друзей `friend`;
- `profiles` — имя пользователя `user` (проиндексированный столбец), About me (Обо мне) `text`.

Поскольку функция `createTable` сначала проверяет факт существования таблицы, эта программа может быть безопасно вызвана несколько раз без выдачи сообщений об ошибках.

Вполне возможно, что при принятии решения о расширении проекта вам понадобится добавить к этим таблицам множество дополнительных столбцов. После принятия такого решения для пересоздания таблицы может потребоваться MySQL-команда `DROP TABLE`.

Пример 27.3. setup.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Настройка базы данных</title>
  </head>
  <body>

    <h3>Setting up...</h3> // Настройка...

  <?php
    require_once 'functions.php';

    createTable('members',
                'user VARCHAR(16),
                pass VARCHAR(16),
                INDEX(user(6))');

    createTable('messages',
                'id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
                auth VARCHAR(16),
```

```
                recip VARCHAR(16),
                pm CHAR(1),
                time INT UNSIGNED,
                message VARCHAR(4096),
                INDEX(auth(6)),
                INDEX(recip(6))');

    createTable('friends',
                'user VARCHAR(16), friend VARCHAR(16),
                INDEX(user(6)),
                INDEX(friend(6))');

    createTable('profiles',
                'user VARCHAR(16), text VARCHAR(4096),
                INDEX(user(6))');
?>

    <br>...done. // ... завершена.
  </body>
</html>
```

Чтобы код примера 27.3 заработал, сначала нужно убедиться в том, что база данных, указанная в переменной $dbname в примере 27.1, создана, а также в том, что пользователям с именем, заданным в $dbuser с паролем в $dbpass, к ней предоставлен доступ.

Файл index.php

Это очень простой, но тем не менее необходимый файл, без которого у проекта не будет главной страницы. Он всего лишь отображает приветствие. В настоящем приложении это может быть страница, сообщающая о достоинствах вашего сайта, подталкивающая посетителя к регистрации.

Кстати, если все MySQL-таблицы вы уже настроили и включаемые файлы создали, то теперь вы можете загрузить файл примера 27.4, `index.php`, в свой браузер, чтобы получить первое представление о новом приложении. На экране должно появиться изображение, показанное на рис. 27.1.

Пример 27.4. index.php

```
<?php
  session_start();
  require_once 'header.php';

  echo "<div class='center'>Welcome to Robin's Nest,";
                          // Добро пожаловать в сообщество Робина
  if ($loggedin) echo " $user, you are logged in";
                          // Вы вошли в приложение
```

```
  else           echo ' please sign up or log in';
                     // Пожалуйста, зарегистрируйтесь или войдите

  echo <<<_END
      </div><br>
    </div>
    <div data-role="footer">
      <h4>Web App from <i><a href='http://lpmj.net/5thedition'
      target='_blank'>Learning PHP MySQL & JavaScript Ed. 5</a></i></h4>
    </div>
  </body>
</html>
_END;
?>
```

Рис. 27.1. Главная страница сайта

Файл signup.php

Теперь нам нужен модуль, позволяющий пользователям присоединиться к новой сети. Это файл `signup.php`, который показан в примере 27.5. Получилась более длинная программа, но все ее части вам уже встречались.

Начнем изучение с блока HTML, расположенного в конце программы. Это простая форма, позволяющая ввести имя пользователя и пароль. Но обратите внимание на использование пустого `<span>`-контейнера с атрибутом `id`, имеющим значение `info`. В этот контейнер будут помещены результаты асинхронного вызова, который имеется в программе и с помощью которого проверяется возможность применения того имени пользователя, которое вы хотите. Полное описание принципов работы этого вызова дано в главе 17.

Проверка возможности применения желаемого имени пользователя

Вернемся к началу программы, где имеется блок кода JavaScript, начинающийся с функции `checkUser`. Эта функция вызывается событием JavaScript `onBlur`, возникающим при перемещении фокуса за пределы принадлежащего форме поля `username`. Сначала эта функция вставляет в упоминавшийся ранее `<span>`-контейнер (у него `id` имеет значение `info`) пустую строку, которая очищает этот контейнер, если он уже имел какое-нибудь содержимое.

Затем делается запрос к программе `checker.php`, которая сообщает о возможности применения имени пользователя `user`. После этого возвращенный асинхронным вызовом результат — приветственное сообщение — помещается в `<span>`-контейнер с идентификатором `info`.

За блоком JavaScript следует PHP-код, известный по разделу главы 16, в котором рассматривалась проверка данных формы. Этот блок кода также задействует функцию `sanitizeString`, чтобы удалить потенциально вредные символы перед поиском имени пользователя в базе данных, и, если такое имя еще никем не задействовано, он вставляет новое имя пользователя `$user` и пароль `$pass` в таблицу базы данных.

Регистрация

После успешной регистрации пользователю предлагается войти на сайт. Более гибкой реакцией на появление нового пользователя мог бы стать автоматический вход на сайт, но я не захотел излишне усложнять код и оставил модули регистрации и входа не связанными друг с другом. Я уверен, что при желании вы сможете и сами без особого труда реализовать подобную функцию.

В коде примера 27.5 используется CSS-класс `fieldname`, предназначенный для приведения в порядок полей формы и их точного выравнивания друг под другом в столбцах. При загрузке в браузер (вместе с приведенным далее файлом `checkuser.php`) эта программа отобразит информацию, показанную на рис. 27.2, из которой видно, что асинхронный вызов помог определить доступность имени Robin для использования. Если нужно, чтобы в поле пароля показывались только звездочки, измените тип этого поля с `text` на `password`.

Рис. 27.2. Страница регистрации

Пример 27.5. signup.php

```
<?php
  require_once 'header.php';

echo <<<_END
  <script>
    function checkUser(user)
    {
      if (user.value == '')
      {
        $('#used').html(' ')
        return
      }

      $.post
      (
        'checkuser.php',
        { user : user.value },
        function(data)
        {
          $('#used').html(data)
        }
      )
    }
  </script>
_END;

  $error = $user = $pass = "";
  if (isset($_SESSION['user'])) destroySession();

  if (isset($_POST['user']))
  {
    $user = sanitizeString($_POST['user']);
    $pass = sanitizeString($_POST['pass']);

    if ($user == "" || $pass == "")
      $error = 'Not all fields were entered<br><br>';
    else
    {
      $result = queryMysql("SELECT * FROM members WHERE user='$user'");

      if ($result->num_rows)
        $error = 'That username already exists<br><br>';
      else
      {
        queryMysql("INSERT INTO members VALUES('$user', '$pass')");
        die('<h4>Account created</h4>Please log in.</div></body></html>');
      }
    }
  }

echo <<<_END
      <form method='post' action='signup.php'>$error
      <div data-role='fieldcontain'>
```

```
        <label></label>
        Please enter your details to sign up
      </div>
      <div data-role='fieldcontain'>
        <label>Username</label>
        <input type='text' maxlength='16' name='user' value='$user'
          onBlur='checkUser(this)'>
        <label></label><div id='used'> </div>
      </div>
      <div data-role='fieldcontain'>
        <label>Password</label>
        <input type='text' maxlength='16' name='pass' value='$pass'>
      </div>
      <div data-role='fieldcontain'>
        <label></label>
        <input data-transition='slide' type='submit' value='Sign Up'>
      </div>
    </div>
  </body>
</html>
_END;
?>
```

При использовании рабочего сервера я не рекомендую сохранять пользовательские пароли так, как это сделано с целью экономии места и упрощения кода в данном проекте, то есть в открытом виде. К паролям нужно подмешивать произвольные строки и хранить их в виде хеш-строк, получаемых с помощью односторонних функций. Более подробно этот процесс рассмотрен в главе 12.

Файл checkuser.php

Файл `checkuser.php`, показанный в примере 27.6, предназначен для работы с файлом `signup.php`. В нем содержится программа, осуществляющая поиск имени пользователя в базе данных и возвращающая строку, которая свидетельствует о том, что такое имя уже было кем-то использовано. Поскольку работа этой программы зависит от функций `sanitizeString` и `queryMysql`, в нее в самом начале включается файл `functions.php`.

Если переменная, являющаяся элементом массива `$_POST`, который имеет ключ `'user'`, хранит какое-нибудь значение, то функция ищет его в базе данных. В зависимости от того, используется такое значение в качестве имени пользователя или нет, функция выводит либо строку `Sorry, already taken` («К сожалению, имя занято»), либо строку `Username available` («Это имя доступно»). Для решения данной задачи достаточно проверить значение, возвращенное функцией `mysql_num_rows`. Если будет возвращен нуль, значит такое имя не найдено, а если единица, то запись с таким именем уже существует.

Для установки перед строкой либо крестика, либо флажка используются HTML-элементы `&#x2718;` и `&#x2714;`, а строка будет отображаться либо красным цветом для класса `taken`, либо зеленым цветом для класса `available`, что определено в файле `styles.css`, показанном далее в этой главе.

Пример 27.6. checkuser.php

```
<?php
  require_once 'functions.php';

  if (isset($_POST['user']))
  {
    $user = sanitizeString($_POST['user']);
    $result = queryMysql("SELECT * FROM members WHERE user='$user'");

    if ($result->num_rows)
      echo "<span class='taken'> &#x2718; " .
           "The username '$user' is taken</span>";
           // Имя занято

    else
      echo "<span class='available'> &#x2714; ".
           "The username '$user' is available</span>";
           // Имя доступно
  }
?>
```

Файл login.php

После того как пользователи получили возможность регистрироваться на сайте, в файле `login.php`, показанном в примере 27.7, предоставляется код, который необходим для входа на сайт. Страница, выводимая этим кодом, как и страница регистрации, похожа на обычную HTML-форму, имеет простую проверку на отсутствие ошибок, а также использует функцию `sanitizeString` перед отправкой запроса к базе данных MySQL.

Стоит обратить особое внимание на присваивание в случае успешной проверки переменным сессии (элементам массива с ключами `user` и `pass`) значений имени пользователя и пароля. На период активности текущей сессии эти переменные будут доступны всем программам проекта, позволяя им автоматически предоставлять доступ вошедшим на сайт пользователям.

Возможно, у вас возникнет вопрос, почему в случае успешного входа на сайт используется функция `die`. Это сделано из соображений экономии, поскольку данная команда объединяет в себе сразу две команды: `echo` и `exit`. Для стилевого оформления этого файла (как и большинства других) применяется класс `main`, позволяющий задать отступ содержимого от левого края.

Если вызвать эту программу в браузере, появится изображение (рис. 27.3). Обратите внимание на то, как для маскировки пароля звездочками тегу `<input>` был присвоен тип `password`, чтобы никто из заглядывающих пользователю через плечо не смог его увидеть.

Пример 27.7. login.php

```
<?php
  require_once 'header.php';
  $error = $user = $pass = "";

  if (isset($_POST['user']))
  {
    $user = sanitizeString($_POST['user']);
    $pass = sanitizeString($_POST['pass']);

    if ($user == "" || $pass == "")
      $error = 'Not all fields were entered';
    else
    {
      $result = queryMySQL("SELECT user,pass FROM members
        WHERE user='$user' AND pass='$pass'");

      if ($result->num_rows == 0)
      {
        $error = "Invalid login attempt";
      }
      else
      {
        $_SESSION['user'] = $user;
        $_SESSION['pass'] = $pass;
        die("You are now logged in. Please <a data-transition='slide'
             href='members.php?view=$user'>click here</a> to continue.</div>
             </body></html>");
      }
    }
  }

echo <<<_END
      <form method='post' action='login.php'>
        <div data-role='fieldcontain'>
          <label></label>
          <span class='error'>$error</span>
        </div>
        <div data-role='fieldcontain'>
          <label></label>
          Please enter your details to log in
        </div>
        <div data-role='fieldcontain'>
          <label>Username</label>
          <input type='text' maxlength='16' name='user' value='$user'>
        </div>
        <div data-role='fieldcontain'>
          <label>Password</label>
```

```
          <input type='password' maxlength='16' name='pass' value='$pass'>
        </div>
        <div data-role='fieldcontain'>
          <label></label>
          <input data-transition='slide' type='submit' value='Login'>
        </div>
      </form>
    </div>
  </body>
</html>
_END;
?>
```

Рис. 27.3. Страница входа на сайт

Файл profile.php

После регистрации и входа на сайт у новых пользователей может появиться желание создать профиль. Это можно будет сделать с помощью файла `profile.php`, код которого показан в примере 27.8. Я думаю, в этом коде можно найти несколько интересных для вас фрагментов, к которым относятся функции загрузки изображений на сайт, изменения их размера и четкости.

Для начала рассмотрим HTML-код, который находится в конце этого файла. Он похож на формы, которые вы только что видели, но на этот раз в форме используется параметр `enctype='multipart/form-data'`. Он позволяет одновременно отправлять более одного типа данных, разрешая отправку на сайт изображений вместе с текстом. В форме также есть элемент `<input>` типа `file`, с помощью которого создается кнопка просмотра. Нажав ее, пользователь может выбрать файл для загрузки на сайт.

При отправке данных формы выполняется код, который находится в начале программы. Перед тем как разрешить выполнение программы, этот код проверяет, вошел ли пользователь на сайт. И только после этого отображается заголовок страницы.

В главе 22 уже говорилось, что по причине использования в библиотеке jQuery Mobile функции асинхронного обмена данными выкладывать файл на сайт из кода HTML при использовании этой функции не представляется возможным, пока она не будет отключена путем добавления в элемент <form> атрибута data-ajax='false'. Тем самым выкладка файлов из кода HTML будет проходить нормально, но при этом будет утрачена возможность выполнения анимации смены страницы.

Добавление текста в поле About Me (Обо мне)

Работа продолжается проверкой `$_POST`-переменной `text` для определения факта отправки текста программе. Если текст отправлен, из него удаляются потенциально вредные фрагменты, а все длинные последовательности пробелов (а также символы возврата каретки и перевода строки) заменяются одним пробелом. В эту функцию включена двойная проверка безопасности, гарантирующая, что пользователь с таким именем действительно существует в базе данных и что перед вставкой этого текста в базу данных, где он превратится в сведения о пользователе в поле About Me (Обо мне), не сможет пройти никакая атака со стороны взломщика.

Если текст не отправлен, делается запрос к базе данных на обнаружение уже существующего текста, чтобы заранее заполнить текстовое окно, тем самым позволяя пользователю отредактировать текст.

Добавление изображения профиля

Теперь перейдем к разделу, проверяющему системную переменную `$_FILES` на наличие загруженного на сайт изображения. Если изображение было загружено, создается строковая переменная `$saveto`. Ей присваивается значение, состоящее из имени пользователя и расширения JPG. Например, для пользователя с именем `Jill` переменной `$saveto` будет присвоено значение `Jill.jpg`. Это имя файла, в котором будет сохранено загруженное изображение, предназначенное для вывода в профиле пользователя.

Сразу за этим проверяется тип загруженного изображения, которое принимается только в том случае, если имеет JPEG-, PNG- или GIF-формат. Если тип загруженного изображения не относится к разрешенным, флажок `$typeok` принимает значение `FALSE`, что препятствует выполнению последнего блока кода загрузки изображения. Но если изображение принимается, загруженное изображение присваивается переменной `$src`. Для этого используется одна из функций `imagecreatefrom`, соответствующая типу загруженного изображения. Теперь изображение находится в том формате, который может быть обработан средствами PHP.

Обработка изображения

Сначала в переменных `$w` и `$h` сохраняются размеры изображения, для чего используется следующая инструкция, представляющая собой быстрый способ присваивания значений из массива отдельным переменным:

```
list($w, $h) = getimagesize($saveto);
```

Затем с использованием значения переменной `$max` (оно равно `100`) вычисляются новые размеры, которые приведут к созданию нового изображения с таким же соотношением сторон, но с размерами, не превышающими 100 пикселов. В результате этого переменным `$tw` и `$th` присваиваются новые значения. При желании получить миниатюры меньшего или большего размера нужно просто соответствующим образом изменить значение переменной `$max`.

После этого вызывается функция `imagecreatetruecolor`, которая создает новую пустую картинку шириной `$tw` и высотой `$th` и сохраняет ее в переменной `$tmp`. Затем вызывается функция `imagecopyresampled`, которая изменяет размер изображения, сохраненного в переменной `$src`, на тот, который хранится в новой переменной `$tmp`. Иногда изменение размера изображения может привести к небольшой потере резкости получаемой копии, поэтому в следующем фрагменте кода используется функция `imageconvolution`, слегка повышающая резкость изображения.

И наконец, изображение сохраняется как JPEG-файл в том месте, которое определено значением переменной `$saveto`, после чего оба изображения — исходное и пустое, имеющее измененные размеры, — удаляются из памяти функцией `imagedestroy`, возвращая системе занятую под них память.

Отображение текущего профиля

И последнюю, но не менее важную задачу выполняет функция `showProfile` из файла `functions.php`, которая позволяет пользователю посмотреть, как выглядит текущий профиль перед его редактированием. Эта функция вызывается до отображения формы HTML. Если профиля еще нет, ничего отображаться не будет.

При выводе изображения профиля с помощью CSS создаются граница, тень и поле справа, чтобы отделить текст профиля от изображения. Результат загрузки в браузер файла, код которого содержится в примере 27.8, показан на рис. 27.4. На этом рисунке можно увидеть, что текстовое поле было заранее заполнено текстом About me (Обо мне).

Пример 27.8. profile.php

```
<?php
  require_once 'header.php';

  if (!$loggedin) die("</div></body></html>");

  echo "<h3>Your Profile</h3>";

  $result = queryMysql("SELECT * FROM profiles WHERE user='$user'");

  if (isset($_POST['text']))
  {
    $text = sanitizeString($_POST['text']);
    $text = preg_replace('/\s\s+/', ' ', $text);

    if ($result->num_rows)
      queryMysql("UPDATE profiles SET text='$text' where user='$user'");
    else queryMysql("INSERT INTO profiles VALUES('$user', '$text')");
  }
  else
  {
    if ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_ASSOC);
      $text = stripslashes($row['text']);
    }
    else $text = "";
  }

  $text = stripslashes(preg_replace('/\s\s+/', ' ', $text));

  if (isset($_FILES['image']['name']))
  {
    $saveto = "$user.jpg";
    move_uploaded_file($_FILES['image']['tmp_name'], $saveto);
    $typeok = TRUE;

    switch($_FILES['image']['type'])
    {
```

```
      case "image/gif": $src = imagecreatefromgif($saveto); break;
      case "image/jpeg": // как для обычного, так
                         // и для прогрессивного jpeg-формата
      case "image/pjpeg": $src = imagecreatefromjpeg($saveto); break;
      case "image/png": $src = imagecreatefrompng($saveto); break;
      default: $typeok = FALSE; break;
    }

    if ($typeok)
    {
      list($w, $h) = getimagesize($saveto);

      $max = 100;
      $tw = $w;
      $th = $h;

      if ($w > $h && $max < $w)
      {
        $th = $max / $w * $h;
        $tw = $max;
      }
      elseif ($h > $w && $max < $h)
      {
        $tw = $max / $h * $w;
        $th = $max;
      }
      elseif ($max < $w)
      {
        $tw = $th = $max;
      }

      $tmp = imagecreatetruecolor($tw, $th);
      imagecopyresampled($tmp, $src, 0, 0, 0, 0, $tw, $th, $w, $h);
      imageconvolution($tmp, array(array(-1, -1, -1),
        array(-1, 16, -1), array(-1, -1, -1)), 8, 0);
      imagejpeg($tmp, $saveto);
      imagedestroy($tmp);
      imagedestroy($src);
    }
  }

  showProfile($user);

echo <<<_END
      <form data-ajax='false' method='post'
        action='profile.php' enctype='multipart/form-data'>
      <h3>Enter or edit your details and/or upload an image</h3>
      <textarea name='text'>$text</textarea><br>
      Image: <input type='file' name='image' size='14'>
      <input type='submit' value='Save Profile'>
      </form>
    </div><br>
  </body>
</html>
_END;
?>
```

Рис. 27.4. Редактирование профиля пользователя

Файл members.php

С помощью файла `members.php`, код которого показан в примере 27.9, пользователи сайта смогут найти других участников сети и добавить их в список своих друзей (или удалить их оттуда, если они уже числились друзьями). У этой программы есть два режима: первый выдает список всех участников и их отношений к вам, а второй показывает пользовательские профили.

Просмотр профилей пользователей

Сначала следует код для последнего режима, где проверяется переменная `view`, извлеченная из массива `$_GET`. Если она существует, значит пользователь хочет просмотреть чей-то профиль, поэтому программа задействует функцию `showProfile`, а также предоставляет две ссылки для друзей и сообщений пользователей.

Добавление и удаление друзей

Затем проверяются две `$_GET`-переменные `add` и `remove`. Если в одной и в другой имеются значения, то это будет имя того пользователя, которого нужно либо добавить в список друзей, либо удалить из него. Для этого выполняется поиск записи пользователя в MySQL-таблице `friends` с последующей вставкой имени пользователя в таблицу или удалением его из этой таблицы.

Разумеется, каждая отправленная переменная сначала проходит обезвреживающую обработку, осуществляемую функцией `sanitizeString`, которая призвана обеспечить ее безопасное использование в MySQL.

Вывод списка всех участников

В последнем блоке кода выдается SQL-запрос на вывод списка всех имен пользователей. Перед выводом заголовка страницы количество возвращенных имен присваивается переменной `$num`.

Затем с помощью цикла `for` осуществляется последовательный перебор всех участников с извлечением сведений о них и дальнейшим поиском их в таблице `friends`. В процессе поиска определяется, проявляют ли они интерес к дружбе с пользователем или проявляет ли пользователь свой интерес к дружбе с ними.

Если обнаруживается взаимный интерес, такие участники классифицируются как взаимные друзья.

Переменная `$t1` имеет ненулевое значение в том случае, когда пользователь проявил интерес к дружбе с другим участником, а переменная `$t2` имеет ненулевое значение, когда другой участник заинтересовался дружбой с пользователем. В зависимости от значений этих переменных текст, отображаемый после каждого имени пользователя, показывает степень взаимоотношений его владельца с текущим пользователем, если таковые имеются.

Кроме того, для отображения взаимоотношений используются соответствующие значки. Двунаправленная стрелка означает, что пользователи являются взаимными друзьями. Стрелка влево показывает, что пользователь проявил интерес к дружбе с другим участником. Стрелка вправо показывает, что дружбой с пользователем заинтересовался другой участник.

И наконец, в зависимости от заинтересованности пользователя в дружбе с другим участником ему предоставляется ссылка для добавления этого участника в список друзей или для удаления его оттуда.

При вызове в браузере программы из примера 27.9 будет выведена страница, показанная на рис. 27.5. Обратите внимание на то, как пользователю предлагается заинтересоваться дружбой с тем участником, к которому еще не проявлен интерес (`follow`), но, если участник уже проявил интерес к дружбе с пользователем, напротив его имени появляется ссылка reap (Ответить), позволяющая ответить ему взаимностью. Если пользователь уже заинтересовался дружбой с другим участником, он может выбрать ссылку drop (Отклонить) для удаления этой заинтересованности.

Рис. 27.5. Использование модуля участников

Пример 27.9. members.php

```
<?php
  require_once 'header.php';

  if (!$loggedin) die("</div></body></html>");

  if (isset($_GET['view']))
  {
    $view = sanitizeString($_GET['view']);

    if ($view == $user) $name = "Your";
    else                $name = "$view's";

    echo "<h3>$name Profile</h3>";
    showProfile($view);
    echo "<a class='button' data-transition='slide'
          href='messages.php?view=$view'>View $name messages</a>";
    die("</div></body></html>");
  }

  if (isset($_GET['add']))
  {
    $add = sanitizeString($_GET['add']);

    $result = queryMysql("SELECT * FROM friends
      WHERE user='$add' AND friend='$user'");
    if (!$result->num_rows)
      queryMysql("INSERT INTO friends VALUES ('$add', '$user')");
  }
  elseif (isset($_GET['remove']))
  {
    $remove = sanitizeString($_GET['remove']);
    queryMysql("DELETE FROM friends WHERE user='$remove' AND friend='$user'");
  }

  $result = queryMysql("SELECT user FROM members ORDER BY user");
  $num = $result->num_rows;

  echo "<h3>Other Members</h3><ul>";

  for ($j = 0 ; $j < $num ; ++$j)
  {
    $row = $result->fetch_array(MYSQLI_ASSOC);
    if ($row['user'] == $user) continue;

    echo "<li><a data-transition='slide' href='members.php?view=" .
      $row['user'] . "'>" . $row['user'] . "</a>";
    $follow = "follow";

    $result1 = queryMysql("SELECT * FROM friends WHERE
      user='" . $row['user'] . "' AND friend='$user'");
    $t1      = $result1->num_rows;
    $result1 = queryMysql("SELECT * FROM friends WHERE
      user='$user' AND friend='" . $row['user'] . "'");
    $t2      = $result1->num_rows;

    if (($t1 + $t2) > 1) echo " &harr; is a mutual friend";
```

```
    elseif ($t1)         echo " &larr; you are following";
    elseif ($t2)       { echo " &rarr; is following you";
                         $follow = "recip"; }

    if (!$t1) echo " [<a data-transition='slide'
      href='members.php?add=" . $row['user'] . "'>$follow</a>]";
    else echo " [<a data-transition='slide'
      href='members.php?remove=" . $row['user'] . "'>drop</a>]";
  }
?>
    </ul></div>
  </body>
</html>
```

На рабочем сервере могут быть тысячи или даже сотни тысяч пользователей, поэтому, скорее всего, возникнет потребность в существенной модификации этой программы, чтобы включить поддержку поиска в тексте About me (Обо мне), поддержку разбиения выводимой информации на страницы и т. д.

Файл friends.php

Код модуля `friends.php`, показывающий друзей пользователя наряду со степенью проявления интереса к дружбе, приведен в примере 27.10. Этот код выполняет исследование таблицы `friends`, похожее на исследование, которое проводится в программе `members.php`, но применительно к одному пользователю. Затем этот модуль показывает всех друзей пользователя и проявляющих интерес к дружбе, а также тех участников, к дружбе с которыми пользователь сам проявляет интерес.

Все люди, проявляющие интерес к дружбе, сохраняются в массиве с именем `$followers`, а все люди, к дружбе с которыми проявляется интерес, сохраняются в массиве с именем `$following`. Затем для извлечения всех людей, которые проявляют интерес к дружбе и к которым проявлен взаимный интерес, применяется следующий весьма лаконичный фрагмент кода:

```
$mutual = array_intersect($followers, $following);
```

Функция `array_intersect` извлекает всех участников, являющихся общими для обоих массивов, и возвращает новый массив, который содержит только этих людей. Затем этот массив сохраняется в переменной `$mutual`. Теперь можно воспользоваться функцией `array_diff` для каждого из массивов `$followers` и `$following`, чтобы в них содержались только те люди, которые не являются взаимными друзьями:

```
$followers = array_diff($followers, $mutual);
$following = array_diff($following, $mutual);
```

В результате этого в массиве `$mutual` будут содержаться только взаимные друзья, в массиве `$followers` — только люди, проявляющие интерес к дружбе (без взаимных

друзей), а в массиве `$following` — только люди, к дружбе с которыми проявляется интерес (также без взаимных друзей).

При наличии этих массивов упрощается решение задачи отдельного отображения каждой категории участников. Результат этого решения показан на рис. 27.6. Функция PHP `sizeof` возвращает количество элементов в массиве. Здесь я использую ее только для вызова кода, когда размер массива не является нулевым (то есть друзья данного типа существуют).

Рис. 27.6. Отображение друзей пользователя и заинтересованности во взаимной дружбе

Обратите внимание на то, как с помощью переменных $name1, $name2 и $name3 в соответствующих местах код может сообщить о том, что вы (пользователь) смотрите собственный список друзей, используя слова Your (Ваши) и You are (Вы), вместо того чтобы просто отобразить имя пользователя.

Если хотите вывести на экран информацию о профиле пользователя, можно убрать символы комментария из закомментированной строки кода.

Пример 27.10. friends.php

```
<?php
  require_once 'header.php';

  if (!$loggedin) die("</div></body></html>");

  if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
  else                      $view = $user;

  if ($view == $user)
  {
    $name1 = $name2 = "Your";
    $name3 =          "You are";
  }
  else
  {
    $name1 = "<a data-transition='slide'
              href='members.php?view=$view'>$view</a>'s";
    $name2 = "$view's";
    $name3 = "$view is";
  }

  // Снимите комментарий со следующей строки кода,
  // если хотите, чтобы здесь был показан профиль пользователя
  // showProfile($view);

  $followers = array();
  $following = array();

  $result = queryMysql("SELECT * FROM friends WHERE user='$view'");
  $num = $result->num_rows;

  for ($j = 0 ; $j < $num ; ++$j)
  {
    $row = $result->fetch_array(MYSQLI_ASSOC);
    $followers[$j] = $row['friend'];
  }

  $result = queryMysql("SELECT * FROM friends WHERE friend='$view'");
  $num = $result->num_rows;

  for ($j = 0 ; $j < $num ; ++$j)
  {
    $row           = $result->fetch_array(MYSQLI_ASSOC);
    $following[$j] = $row['user'];
  }

  $mutual    = array_intersect($followers, $following);
```

```
  $followers = array_diff($followers, $mutual);
  $following = array_diff($following, $mutual);
  $friends   = FALSE;
  echo "<br>";

  if (sizeof($mutual))
  {
    echo "<span class='subhead'>$name2 mutual friends</span><ul>";
    foreach($mutual as $friend)
      echo "<li><a data-transition='slide'
            href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
  }

  if (sizeof($followers))
  {
    echo "<span class='subhead'>$name2 followers</span><ul>";
    foreach($followers as $friend)
      echo "<li><a data-transition='slide'
            href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
  }

  if (sizeof($following))
  {
    echo "<span class='subhead'>$name3 following</span><ul>";
    foreach($following as $friend)
      echo "<li><a data-transition='slide'
            href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
  }

  if (!$friends) echo "<br>You don't have any friends yet.<br><br>";

  echo "<a data-role='button' data-transition='slide'
        href='messages.php?view=$view'>View $name2 messages</a>";
?>
    </div>
  </body>
</html>
```

Файл messages.php

Код последнего из основных модулей, `messages.php`, показан в примере 27.11. Этот модуль начинает работу с проверки наличия отправленного сообщения в элементе POST-массива с ключом `'text'`. Если сообщение имеется, оно вставляется в таблицу `messages`. Одновременно с этим сохраняется значение элемента с ключом `pm`. Значение этого элемента свидетельствует об открытом или закрытом статусе сообщения. Нуль представляет открытое, а единица — закрытое сообщение.

Затем отображаются пользовательский профиль и форма для ввода сообщения, а также переключатели для выбора между отправкой закрытого (`private`) или открытого (`public`) сообщения. После этого показываются все сообщения: если они имеют статус открытого сообщения, их могут просматривать все пользователи, а закрытые сообщения могут просматривать только отправитель и получатель. Все это управляется двумя запросами к базе данных MySQL. В дополнение к этому, когда сообщение имеет статус закрытого, оно представлено словом `whispered` («прошептал») и отображается курсивом.

И наконец, программа отображает две ссылки: для обновления сообщений в том случае, когда другой пользователь за время просмотра опубликовал новое сообщение, и для просмотра друзей пользователя. Здесь опять применяется прием с переменными `$name1` и `$name2`, чтобы при просмотре вашего собственного профиля вместо имени пользователя отображалось слово Your (Ваши).

Пример 27.11. messages.php

```
require_once 'header.php';

if (!$loggedin) die("</div></body></html>");

if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
else                      $view = $user;

if (isset($_POST['text']))
{
  $text = sanitizeString($_POST['text']);

  if ($text != "")
  {
    $pm = substr(sanitizeString($_POST['pm']),0,1);
    $time = time();
    queryMysql("INSERT INTO messages VALUES(NULL, '$user',
      '$view', '$pm', $time, '$text')");
  }
}

if ($view != "")
{
  if ($view == $user) $name1 = $name2 = "Your";
  else
  {
    $name1 = "<a href='members.php?view=$view'>$view</a>'s";
    $name2 = "$view's";
  }

  echo "<h3>$name1 Messages</h3>";
  showProfile($view);

  echo <<<_END
    <form method='post' action='messages.php?view=$view'>
      <fieldset data-role="controlgroup" data-type="horizontal">
        <legend>Type here to leave a message</legend>
```

```
        <input type='radio' name='pm' id='public' value='0' checked='checked'>
        <label for="public">Public</label>
        <input type='radio' name='pm' id='private' value='1'>
        <label for="private">Private</label>
      </fieldset>
      <textarea name='text'></textarea>
      <input data-transition='slide' type='submit' value='Post Message'>
    </form><br>
_END;

    date_default_timezone_set('UTC');

    if (isset($_GET['erase']))
    {
      $erase = sanitizeString($_GET['erase']);
      queryMysql("DELETE FROM messages WHERE id=$erase AND recip='$user'");
    }

    $query = "SELECT * FROM messages WHERE recip='$view' ORDER BY time DESC";
    $result = queryMysql($query);
    $num = $result->num_rows;

    for ($j = 0 ; $j < $num ; ++$j)
    {
      $row = $result->fetch_array(MYSQLI_ASSOC);

      if ($row['pm'] == 0 || $row['auth'] == $user ||
          $row['recip'] == $user)
      {
        echo date('M jS \'y g:ia:', $row['time']);
        echo " <a href='messages.php?view=" . $row['auth'] .
             "'>" . $row['auth']. "</a> ";

        if ($row['pm'] == 0)
          echo "wrote: &quot;" . $row['message'] . "&quot; ";
        else
          echo "whispered: <span class='whisper'>&quot;" .
            $row['message']. "&quot;</span> ";

        if ($row['recip'] == $user)
          echo "[<a href='messages.php?view=$view" .
               "&erase=" . $row['id'] . "'>erase</a>]";

        echo "<br>";
      }
    }
  }

  if (!$num)
    echo "<br><span class='info'>No messages yet</span><br><br>";

  echo "<br><a data-role='button'
        href='messages.php?view=$view'>Refresh messages</a>";
?>
    </div><br>
  </body>
</html>
```

Результат запуска этой программы в браузере показан на рис. 27.7. Обратите внимание на то, как пользователям, просматривающим собственные сообщения, предоставляется ссылка для того, чтобы можно было стереть любое из нежелательных сообщений. Следует также отметить присущую jQuery Mobile реализацию стилевого оформления переключателей, предназначенных для выбора между отправкой частного или общедоступного сообщения. Принцип того, как это работает, объясняется в главе 22.

Рис. 27.7. Модуль передачи сообщений

Файл logout.php

Модуль `logout.php`, код которого показан в примере 27.12, является завершающим ингредиентом рецепта нашей социальной сети. Он отображает страницу выхода с сайта, которая закрывает сессию и удаляет любые связанные с ней данные и cookie-файлы. Результат вызова этой программы можно увидеть на рис. 27.8, где на этот раз к пользователю обращена просьба щелкнуть на ссылке. Эта ссылка приведет его на главную страницу, предназначенную для пользователей, еще не вошедших на сайт, с которой удалены все ссылки в верхней части экрана, предназначенные для вошедших на сайт пользователей. Разумеется, вы можете создать код на JavaScript или на PHP, который сразу перенаправил бы пользователя на эту страницу, чтобы очистить страницу выхода с сайта от ненужных элементов.

Рис. 27.8. Страница выхода

Пример 27.12. logout.php

```
<?php
  require_once 'header.php';

  if (isset($_SESSION['user']))
  {
    destroySession();
    echo "<br><div class='center'>You have been logged out. Please
         <a data-transition='slide' href='index.php'>click here</a>
         to refresh the screen.</div>";
  }
```

```
  else echo "<div class='center'>You cannot log out because
            you are not logged in</div>";
?>
    </div>
  </body>
</html>
```

Файл styles.css

Таблица стилей, используемая для этого проекта, показана в примере 27.13. В нем содержится несколько наборов объявлений.

- `*` — с помощью универсального селектора устанавливаются применяемые в проекте по умолчанию семейство шрифтов и размер шрифта.
- `body` — задается ширина окна проекта, горизонтальная центровка, указывается цвет фона и задается граница окна.
- `html` — устанавливается цвет фона блока HTML.
- `img` — задаются граница, тень и правое поле для всех изображений.
- `.username` — задается центрирование имени пользователя, выбираются семейство шрифтов, размер, цвет, фон и отступы, с которыми это имя отображается.
- `.info` — задействуется для отображения важной информации: для ее элементов, использующих этот класс, устанавливаются цвета фона и текста, применяются граница и отступы.
- `.center` — используется для центрирования содержимого `<div>`-элемента.
- `.subhead` — выделяется блок текста.
- `.taken`, `.available`, `.error` и `.whisper` — устанавливаются цвета и стили шрифтов, используемых для отображения различных типов информации.
- `#logo` — задействуется для стилевого оформления текста логотипа, применяемого в качестве запасного варианта в случае использования браузера, не поддерживающего HTML5 и невозможности в силу этого создания логотипа на холсте.
- `#robin` — задействуется для выравнивания изображения в заголовке страницы.
- `#used` — задействуется для того, чтобы отдалить от расположенного выше поля тот элемент, который заполняется в результате асинхронного вызова `checkuser.php`, в том случае, когда имя пользователя уже занято.

Пример 27.13. Файл styles.css

```
* {
  font-family:verdana,sans-serif;
  font-size :14pt;
}

body {
  width :700px;
  margin :20px auto;
  background:#f8f8f8;
```

```
  border :1px solid #888;
}

html {
  background:#fff
}

img {
  border :1px solid black;
  margin-right :15px;
  -moz-box-shadow :2px 2px 2px #888;
  -webkit-box-shadow:2px 2px 2px #888;
  box-shadow :2px 2px 2px #888;
}

.username {
  text-align :center;
  background :#eb8;
  color :#40d;
  font-family:helvetica;
  font-size :20pt;
  padding :4px;
}

.info {
  font-style :italic;
  margin :40px 0px;
  text-align :center;
  }

.center {
  text-align:center;
}

.subhead {
  font-weight:bold;
}

.taken, .error {
  color:red;
}

.available {
  color:green;
}

.whisper {
  font-style:italic;
  color :#006600;
}

#logo {
  font-family:Georgia;
  font-weight:bold;
  font-style :italic;
  font-size :97px;
```

```
  color :red;
}

#robin {
  position :relative;
  border :0px;
  margin-left :-6px;
  margin-right :0px;
  top :17px;
  -moz-box-shadow :0px 0px 0px;
  -webkit-box-shadow:0px 0px 0px;
  box-shadow :0px 0px 0px;
}

#used {
  margin-top:50px;
}
```

Файл javascript.js

И наконец, разберем файл JavaScript (пример 27.14), в котором содержатся функции `O`, `S` и `C`, используемые кодом по всей книге.

Пример 27.14. Файл javascript.js

```
function O(i)
{
  return typeof i == 'object' ? i : document.getElementById(i)
}

function S(i)
{
  return O(i).style
}

function C(i)
{
  return document.getElementsByClassName(i)
}
```

Ну вот, собственно, и все. Если вы создадите что-нибудь на основе этого кода или других примеров, приведенных в издании, или извлечете пользу от них каким-либо другим образом, я буду рад тому, что мне удалось вам в чем-то помочь. И спасибо за чтение этой книги!

Но перед тем, как закрыть книгу и приступить к практическому применению на просторах Всемирной паутины только что полученных навыков, пожалуйста, просмотрите следующие за этой главой приложения. В них содержится множество дополнительной информации, которая может оказаться полезной для вас.

A Ответы на контрольные вопросы

Глава 1

1. Веб-сервер (такой как Apache), язык сценариев на стороне сервера (PHP), база данных (MySQL) и язык сценариев на стороне клиента (JavaScript).
2. *Язык гипертекстовой разметки* — HyperText Markup Language: сама веб-страница, включая текст и теги разметки.
3. Как практически все процессоры баз данных, MySQL воспринимает команды на *структурированном языке запросов* — Structured Query Language (SQL). SQL является способом общения с MySQL любого пользователя (а также PHP- программы).
4. Сценарии PHP работают на сервере, а сценарии JavaScript — на машине клиента. У языка PHP есть средства общения с базой данных, позволяющие хранить и извлекать данные, но его средствами невозможно провести быстрое и динамическое изменение веб-страницы, просматриваемой пользователем. У языка JavaScript имеются совершенно противоположные достоинства и недостатки.
5. CSS означает *каскадные таблицы стилей* — Cascading Style Sheets: правила задания стилей и разметки, применяемые к элементам HTML-документа.
6. Наверное, наиболее интересными новыми элементами в HTML5 являются `<audio>`, `<video>` и `<canvas>`. Хотя есть и многие другие, например `<article>`, `<summary>`, `<footer>` и т. д.
7. Некоторые из этих технологий подконтрольны компаниям, собирающим сведения об ошибках и занимающимся их устранением, как и любые другие компании, производящие программные продукты. Но программы с открытым кодом также зависят от сообщества разработчиков, поэтому ваш отчет об ошибке может быть рассмотрен любым пользователем, хорошо разбирающимся в коде. Возможно, когда-нибудь и вам доведется исправлять ошибки, допущенные в инструментарии, имеющем открытый код.

8. Она позволяет разработчикам сконцентрироваться на создании основной функции веб-сайта или веб-приложения, возложив на среду задачу обеспечения наиболее эффективной работы и отображения информации, независимо от используемой платформы (будь то Linux, macOS, Windows, iOS или Android), размеров экрана или типа задействованного браузера.

Глава 2

1. WAMP означает *Windows*, *Apache*, *MySQL* и *PHP*, в то время как *M* в MAMP означает *Mac* вместо Windows, а *L* в LAMP — *Linux*. Все эти аббревиатуры ссылаются на полноценные решения для хостинга динамических веб-страниц.
2. Оба адреса — и `127.0.0.1`, и `http://localhost` — являются способами ссылки на локальный компьютер. При правильной настройке WAMP или MAMP для вызова исходной страницы на локальном сервере в адресную строку браузера можно вводить любой из этих адресов.
3. FTP означает *протокол передачи файло*в — File Transfer Protocol. Программа FTP используется для передачи файлов между клиентом и сервером в обе стороны.
4. Чтобы обновить файлы, их нужно отправить на удаленный сервер с помощью FTP-программы, что может существенно увеличить время разработки, если это действие повторяется многократно на протяжении одного рабочего сеанса.
5. Специализированные редакторы программ обладают множеством автоматических функций и способны подсвечивать проблемные участки кода даже до его запуска на выполнение.

Глава 3

1. Используется тег `<?php ... ?>`, который может быть сокращен до пары тегов `<? ... ?>`, но так делать не рекомендуется.
2. Для отдельной строки комментария можно воспользоваться сочетанием символов `//`, а комментарии, занимающие несколько строк, можно заключить в пары символов `/* ... */`.
3. Все PHP-инструкции должны заканчиваться точкой с запятой (`;`).
4. Все имена PHP-переменных, исключая имена констант, должны начинаться с символа `$`.
5. Переменная содержит значение, которое может быть строкой, числом или другими данными.
6. Выражение `$variable = 1` является инструкцией присваивания, а группа символов `==` в выражении `$variable == 1` является оператором сравнения. Инструкцию `$variable = 1` следует использовать для присваивания значения переменной

`$variable`. Инструкцию `$variable == 1` необходимо применять в последующих строках программы для определения того, равно значение переменной `$variable` единице или нет. Если для сравнения воспользоваться по ошибке инструкцией `$variable = 1`, скорее всего, произойдут два нежелательных события: переменной `$variable` будет присвоено значение `1` и будет неизменно возвращаться истинное значение, независимо от предыдущего значения этой переменной.

7. В PHP дефисы зарезервированы для операторов вычитания, уменьшения на единицу и смены знака числа на минус. Если бы в именах переменных можно было использовать дефисы, это затруднило бы интерпретацию такой конструкции, как `$current-user`, что в любом случае приводило бы к двусмысленности программного кода.
8. Да, имена переменных чувствительны к регистру букв. Поэтому `$This_Variable` и `$this_variable` являются совершенно разными переменными.
9. Пробелы в именах переменных недопустимы, поскольку собьют с толку парсер (синтаксический анализатор) PHP. Вместо них следует использовать знаки подчеркивания (_).
10. Чтобы перевести значение переменной из одного типа в другой, нужно сослаться на эту переменную, и PHP автоматически преобразует ее тип.
11. Разница между `++$j` и `$j++` ни в чем не проявится до тех пор, пока значение переменной `$j` не будет проходить проверку, присваиваться другой переменной или передаваться функции в качестве параметра. В таких случаях использование выражения `++$j` приводит к увеличению значения `$j` на единицу до выполнения проверки или другой инструкции, а применение выражения `$j++` приводит к тому, что сначала выполняется инструкция, а затем значение переменной `$j` увеличивается на единицу.
12. В большинстве случаев, когда не нужно соблюдать приоритетность, операторы `&&` и `and` являются взаимозаменяемыми. При необходимости соблюдения приоритетности у оператора `&&` более высокий уровень приоритета, чем у оператора `and`.
13. Чтобы создать многострочную команду `echo` или инструкцию присваивания, можно воспользоваться кавычками или конструкцией `<<< _END ... _END;`. В последнем случае закрывающий тег должен быть единственным в своей строке, без чего-либо, что находилось бы перед ним или после него.
14. Значения констант нельзя переопределять, потому что, будучи единожды определенными, они сохраняют свое значение до прекращения работы программы.
15. Для изменения исходного предназначения кавычек можно воспользоваться сочетанием `\'` для отключения одинарной кавычки или сочетанием `\"` для отключения двойной кавычки.
16. Команды `echo` и `print` похожи друг на друга, за исключением того, что `print` является функцией PHP и воспринимает один аргумент, а `echo` является конструкцией, которая может воспринимать несколько аргументов.

17. Функции предназначены для выделения отдельных частей кода в самостоятельные изолированные блоки, на которые можно ссылаться по одному лишь имени функции.
18. Объявив переменную глобальной, ее можно сделать доступной для всех частей PHP-программы.
19. Если данные сгенерированы внутри функции, их можно передать всей остальной программе путем возвращения значения с помощью инструкции `return` или изменения значения глобальной переменной.
20. При объединении строки с числом получается еще одна строка.

Глава 4

1. В PHP `TRUE` представляет значение `1`, а `FALSE` представляет значение `NULL`, которое можно рассматривать как «ничто» и которое выводится на экран как пустая строка.
2. Самые простые формы выражений — это литералы, к которым относятся числа и строки, а также переменные, которые просто вычисляются в самих себя.
3. Разница между унарными, бинарными и трехкомпонентными операторами состоит в количестве необходимых им операндов (им требуется соответственно один, два и три операнда).
4. Наилучший способ установки собственной приоритетности операторов состоит в заключении тех подвыражений, которым нужно придать более высокий уровень приоритета, в круглые скобки.
5. Взаимосвязанность операторов относится к направлению обработки выражения (слева направо или справа налево).
6. Оператор тождественности используется в том случае, когда нужно обойти присущее PHP автоматическое изменение типа операнда (которое называется также приведением типов).
7. К условным инструкциям относятся `if`, `switch` и оператор `?:`.
8. Чтобы пропустить выполнение текущей итерации цикла и перейти к следующей, используется инструкция `continue`.
9. Циклы, в которых применяется инструкция `for`, считаются более мощными, чем циклы `while`, так как они поддерживают два дополнительных параметра, которые управляют работой цикла.
10. Большинство условных выражений в инструкциях `if` и `while` являются литералами (или логическими выражениями), поэтому они инициируют выполнение кода только в том случае, если вычисляются как `TRUE`. Числовые выражения инициируют выполнение, когда их значение является ненулевым. Строковые выражения инициируют выполнение, когда вычисляются как непустая строка. Значение `NULL` вычисляется как ложное, поэтому не инициирует выполнение кода.

Глава 5

1. Функции избавляют от необходимости многократно копировать или переписывать одни и те же фрагменты кода, объединяя набор инструкций и позволяя использовать для его вызова простое имя.
2. По умолчанию функции могут возвращать одно значение. Но применение массивов, ссылок и глобальных переменных позволяет возвращать любое количество значений.
3. При обращении к переменной по имени, например при присваивании ее значения другой переменной или передаче ее функции, значение переменной копируется. При этом изменение значения копии не приводит к изменению исходного значения. Но если вы ссылаетесь на переменную, используется только указатель (или ссылка) на ее значение, поэтому на одно и то же значение ссылается сразу несколько имен. Изменение значения по ссылке приводит к изменению исходного значения.
4. Под областью видимости понимаются части программы, из которых может быть получен доступ к переменной. Например, переменная с глобальной областью видимости может быть доступна из всех частей PHP-программы.
5. Чтобы включить один файл в состав другого, можно воспользоваться инструкциями `include` или `require` или их более безопасными вариантами `include_once` и `require_once`.
6. Функция является набором инструкций, на который производится ссылка по имени и который может принимать и возвращать значения. Объект может состоять из нуля или нескольких функций (которые применительно к нему называются методами), а также из переменных (называемых свойствами объекта). Все они объединяются в одно образование.
7. Для создания в PHP нового объекта используется ключевое слово `new`:

   ```
   $object = new Class;
   ```

8. Для создания подкласса используется ключевое слово `extends`, которое является частью следующей синтаксической конструкции:

   ```
   class Подкласс extends Родительский_класс
   ```

9. Чтобы при создании объекта он был проинициализирован, можно вызвать инициализирующую часть кода путем создания внутри класса метода-конструктора под названием `__construct` и поместить в него ваш код.
10. Явного объявления свойств внутри класса не требуется, поскольку они после первого же применения будут объявлены неявным образом. Но явное объявление считается правилом хорошего тона, потому что улучшает читаемость кода, упрощает его отладку и приносит особую пользу тем людям, которым приходится обслуживать ваш код.

Глава 6

1. Числовой массив может быть проиндексирован с помощью чисел или числовых переменных. Ассоциативный массив использует для индексации элементов буквенно-цифровые идентификаторы.
2. Основное преимущество ключевого слова `array` состоит в том, что оно дает возможность присваивать массиву сразу несколько значений без повторения имени этого массива.
3. Как функция `each`, так и структура организации цикла `foreach...as` возвращают элементы из массива. Обе приступают к работе с начала массива и увеличивают значение указателя на единицу, обеспечивая при следующем вызове или итерации возвращение следующего элемента, и обе возвращают `FALSE` при достижении конца массива. Разница между ними в том, что функция `each` возвращает только один элемент, поэтому обычно помещается в цикл. Конструкция `foreach...as` уже является циклом, выполняемым снова и снова до тех пор, пока не закончатся элементы массива или пока цикл не будет прерван явным образом.
4. Для создания многомерного массива нужно элементам основного массива присвоить значения, представляющие собой дополнительные массивы.
5. Для подсчета количества элементов в массиве может использоваться функция `count`.
6. Функция `explode` предназначена для извлечения частей строки, которые разделены идентификатором, например для извлечения слов, разделенных в предложении пробелами.
7. Чтобы вернуть внутренний указатель текущего элемента массива PHP на первый элемент этого массива, вызывается функция `reset`.

Глава 7

1. Для отображения числа с плавающей точкой следует использовать спецификатор преобразования `%f`.
2. Для приема строки `"Happy Birthday"` и вывода строки `**Happy` можно воспользоваться инструкцией:

   ```
   printf: printf("%'*7.5s", "Happy Birthday");.
   ```

3. Для выдачи информации из `printf` не в браузер, а в переменную следует воспользоваться альтернативной функцией `sprintf`.
4. Чтобы создать отметку времени UNIX для времени и даты, представленных в виде `7:11am May 2nd, 2016`, нужно воспользоваться командой:

   ```
   $timestamp = mktime(7, 11, 0, 5, 2, 2016);.
   ```

5. Чтобы открыть файл в режиме чтения и записи с усечением его размера и установкой указателя на начало файла, следует в функции `fopen` воспользоваться режимом доступа к файлу `w+`.
6. Для удаления файла `file.txt` нужно применить такую PHP-команду:
```
unlink('file.txt');
```
7. Для чтения целиком всего файла используется функция `file_get_contents`. Эта же функция, если ей предоставить URL-адрес, позволяет прочитать файл из Интернета.
8. Сведения о файлах, загруженных на сервер, содержатся в ассоциативном PHP-массиве `$_FILES`.
9. Системные команды можно запускать с помощью PHP-функции `exec`.
10. В HTML5 можно использовать либо стиль тегов XHTML 1.0 (например, `<hr />`), либо стиль стандарта HTML4 (например, `<hr>`). Выбор полностью зависит от стиля, используемого вашей компанией.

Глава 8

1. В MySQL точка с запятой используется для разделения или завершения команд. Если забыть ввести этот символ, то MySQL выдаст приглашение и будет ожидать его ввода.
2. Для просмотра доступных баз данных следует набрать команду `SHOW databases`. Для просмотра таблиц, использующихся в базе данных, нужно набрать команду `SHOW tables`. (Команды нечувствительны к регистру букв.)
3. Для создания нового пользователя применяется команда `GRANT`:
```
GRANT PRIVILEGES ON newdatabase.* TO 'newuser'@'localhost'
  IDENTIFIED BY 'newpassword';
```
4. Для просмотра структуры таблицы нужно набрать команду `DESCRIBE` *имя_таблицы*.
5. Индекс в MySQL предназначен для существенного сокращения времени доступа к базе данных за счет добавления метаданных к таблице в одном или нескольких ключевых столбцах, в которых после индексации можно проводить быстрый поиск для обнаружения в таблице нужной строки.
6. Индекс `FULLTEXT` позволяет запросам, в которых используется естественный язык, находить ключевые слова, если они имеются в столбце или столбцах, проиндексированных в режиме `FULLTEXT`. Способ поиска во многом похож на тот, для которого задействуется поисковый механизм.
7. К стоповым относятся слова, имеющие настолько широкое распространение, что нет смысла включать их в индекс `FULLTEXT` или использовать при поиске. Но все же они включаются в поиск, когда входят в состав большой строки, заключенной в кавычки.
8. По сути, спецификатор `SELECT DISTINCT` воздействует только на отображение, выбирая всего одну строку и исключая все ее дубликаты. Спецификатор

GROUP BY не исключает, а объединяет все строки, у которых есть одно и то же значение в столбце. Поэтому спецификатор GROUP BY хорошо справляется с такими операциями, как COUNT, ведущей подсчеты в группе строк. А спецификатор SELECT DISTINCT для таких целей не годится.

9. Для возвращения только тех строк, в которых в каком-нибудь месте столбца author таблицы classics содержится слово Langhorne, предназначена такая команда:

```
SELECT * FROM classics WHERE author LIKE "%Langhorne%";
```

10. Для того чтобы можно было объединить две таблицы, у них должен быть общим хотя бы один столбец, например номер ID или, как в случае с таблицами classics и customers, столбец isbn.

Глава 9

1. Под словом *«отношение»* понимается связь между двумя элементами данных, обладающими какими-нибудь взаимными ассоциациями, например книга и ее автор или книга и покупатель, который ее приобрел. Реляционная (то есть учитывающая отношения) база данных, например MySQL, специализируется на хранении и извлечении подобных отношений.
2. К процессу удаления повторяющихся данных и оптимизации таблиц применяется термин *«нормализация»*.
3. Существует три правила первой нормальной формы.
   - В ней не должно быть никаких повторяющихся столбцов, содержащих одни и те же типы данных.
   - Каждая позиция пересечения столбца и строки должна содержать только одно значение.
   - Для уникальной идентификации каждой строки должен использоваться первичный ключ.
4. Чтобы таблица отвечала требованиям второй нормальной формы, столбцы, в которых данные повторяются в нескольких строках, должны быть перемещены в собственные таблицы.
5. При отношении «один ко многим» первичный ключ из таблицы со стороны «один» должен быть добавлен в качестве отдельного столбца (внешнего ключа) к таблице на стороне «многие».
6. Для создания базы данных, в которой имеется отношение «многие ко многим», нужно создать промежуточную таблицу, содержащую ключи из двух других таблиц. В результате эти две другие таблицы смогут ссылаться друг на друга посредством третьей таблицы.
7. Чтобы инициировать MySQL-транзакцию, используется либо команда BEGIN, либо команда START TRANSACTION. Для прекращения транзакции и отмены всех действий выдается команда ROLLBACK. Для завершения транзакции и совершения всех действий выдается команда COMMIT.

8. Для изучения подробностей работы запроса можно воспользоваться командой `EXPLAIN`.
9. Для создания в файле `publications.sql` резервной копии базы данных `publications` можно воспользоваться такой командой:

```
mysqldump -u пользователь -pпароль publications > publications.sql
```

Глава 10

1. Для подключения к базе данных MySQL с помощью `mysqli` следует вызвать метод `mysqli`, передав ему имя хоста, имя пользователя, пароль и имя базы данных. В случае успеха будет возвращен объект подключения.
2. Для отправки запроса к MySQL с помощью `mysqli` нужно сначала убедиться в том, что объект подключения к базе данных уже создан, а затем вызвать метод `query` этого объекта, передав ему строку запроса.
3. При возникновении ошибки `mysqli` в свойстве `error` объекта подключения содержится сообщение об ошибке. Если ошибка возникла при подключении к базе данных, сообщение об ошибке будет содержаться в свойстве `connect_error`.
4. Для определения количества строк, возвращенных запросом `mysqli`, нужно воспользоваться свойством `num_rows` объекта `result`.
5. Для извлечения из набора результатов `mysqli` конкретной строки нужно воспользоваться методом `data_seek` объекта `result`, передав ему номер строки (номера строк начинаются с нуля); затем для получения требуемой информации нужно вызвать `fetch_array` или другой метод извлечения данных. При получении всех результатов этого делать не требуется.
6. Для обезвреживания символов в строках можно вызвать метод `real_escape_string` объекта подключения `mysqli`, передав ему строку, в которой обезвреживаются символы. Конечно, в целях обеспечения безопасности лучше всего использовать подготовленные инструкции.
7. Если пренебречь закрытием объектов, созданных методами `mysqli`, ваши программы подвергнутся риску возникновения дефицита памяти, особенно на сайтах с высоким уровнем трафика. Если в программе имеется ошибка хода ее выполнения, то закрытием объектов вы исключаете также случайное получение старых результатов.

Глава 11

1. Для передачи данных формы PHP-программе GET-методом используется массив `$_GET`, а POST-методом — массив `$_POST`.
2. Несмотря на то что оба этих поля при заполнении формы воспринимают текст, разница между ними заключается в том, что в текстовом поле может содержаться только одна строка, а в текстовой области может быть несколько строк и в ней осуществляется перенос слов на новую строку.

3. Для предоставления пользователю возможности выбора в веб-форме трех взаимоисключающих вариантов следует задействовать переключатели, поскольку флажки допускают множественный выбор.
4. Для отправки из веб-формы группы значений при использовании только одного имени поля следует воспользоваться не обычным именем поля, а именем массива с квадратными скобками, например `choices[]`. Тогда каждое значение помещается в массив, длина которого будет соответствовать количеству отправленных элементов.
5. Чтобы отправить данные поля формы, не отображая их на экране браузера, следует поместить эти данные в скрытое поле, в котором задан атрибут `type="hidden"`.
6. Элемент формы, а также сопровождающие его текст или графику можно заключить в теги `<label>` и `</label>` и гарантировать тем самым возможность выбора щелчком кнопкой мыши на всей области объединенного элемента.
7. Чтобы можно было превратить HTML в такой формат, который может быть отображен на экране, но не сможет интерпретироваться браузером как код HTML, используется PHP-функция `htmlentities`.
8. Помочь пользователям в заполнении полей данными, которые уже были где-то отправлены, можно с помощью атрибута `autocomplete`, который приводит к появлению предложений с возможными значениями.
9. Чтобы при отправке формы не было пропущенных данных, к обязательным для заполнения элементам ввода нужно применить атрибут `required`.

Глава 12

1. Cookie должны быть переданы до кода HTML веб-страницы, потому что они отсылаются в виде составной части заголовков.
2. Для сохранения cookie на машине браузера применяется функция `set_cookie`.
3. Для удаления cookie нужно его выдать заново, но при этом установить срок истечения действия на уже прошедшее время.
4. При HTTP-аутентификации имя пользователя и пароль сохраняются в элементах массива `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW]`.
5. Функция `password_hash` считается мощным средством защиты, потому что это односторонняя функция, превращающая строку в длинную строку, представляющую собой шестнадцатеричное число, не поддающуюся обратному преобразованию, из-за чего ее крайне трудно взломать, при условии, что от пользователей требуется задание стойкого пароля (например, длиной не менее восьми символов, включая вставленные в произвольном месте цифры и знаки пунктуации).
6. При подмешивании произвольных данных, прежде чем преобразовать строку с помощью функции `hash` (с чем PHP должен справиться за вас), в нее добавляются лишние символы (известные только программисту). Тем самым

гарантируется, что у пользователей с одиноковым паролем не будет одинакового хеша и не будут использоваться заранее вычисленные хеш-таблицы.

7. PHP-сессия — это группа уникальных для текущего пользователя переменных, передаваемых вместе с последовательными запросами с таким расчетом, чтобы переменные оставались доступными при посещении пользователем разных страниц.
8. Чтобы инициировать PHP-сессию, используется функция `session_start`.
9. Хищение сессии происходит в том случае, когда взломщик каким-то образом добывает ID существующей сессии и пытается ее захватить.
10. Фиксация сессии заключается в попытке злоумышленника заставить пользователя войти в систему, используя неверный ID сессии, взламывая таким образом безопасность подключения.

Глава 13

1. В качестве контейнера для кода JavaScript используются теги `<script>` и `</script>`.
2. По умолчанию информация, выводимая кодом JavaScript, будет добавлена к той части документа, в которой находится этот код. Если он находится в заголовке, она будет добавлена в заголовок, а если в теле документа, то в тело.
3. Код JavaScript из других файлов может быть включен в ваш документ либо путем копирования и последующей вставки этого кода, либо более распространенным способом — включением его в качестве составляющей тега `<script src='filename.js'>`.
4. В JavaScript эквивалентом PHP-команд `echo` и `print` служит функция (или метод) `document.write`.
5. Чтобы создать комментарий в JavaScript, нужно перед текстом однострочного комментария поставить символы `//`, а многострочный комментарий поместить между символами `/*` и `*/`.
6. Оператором объединения строк в JavaScript служит знак «плюс» (`+`).
7. Внутри функции JavaScript можно определить переменную с локальной областью видимости, поставив в начале первой инструкции присваивания значения этой переменной ключевое слово `var`.
8. Чтобы во всех основных браузерах отобразить URL-адрес, присвоенный ссылке, имеющей ID со значением `thislink`, можно воспользоваться двумя следующими командами:

```
document.write(document.getElementById('thislink').href)
document.write(thislink.href)
```

9. Команды для загрузки предыдущей страницы, ссылка на которую хранится в истории браузера, имеют следующий вид:

```
history.back()
history.go(-1)
```

10. Для замены текущего документа главной страницей сайта oreilly.com можно воспользоваться такой командой:

```
document.location.href = 'http://oreilly.com'
```

Глава 14

1. Наиболее заметной разницей между логическими значениями в PHP и JavaScript является то, что PHP распознает ключевые слова `TRUE`, `true`, `FALSE` и `false`, а в JavaScript поддерживаются только ключевые слова `true` и `false`. Кроме того, в PHP `TRUE` имеет значение `1`, а `FALSE` — значение `NULL`, в то время как в JavaScript они представлены значениями `true` и `false`, которые могут быть возвращены в виде строк.
2. В отличие от PHP, в JavaScript для определения имени переменной никакие символы (вроде `$`) не используются. Имена переменных в JavaScript могут начинаться и далее состоять из любых букв в верхнем или нижнем регистре, а также из символов подчеркивания, в имена также могут включаться цифры, но только не в качестве первого символа.
3. Разница между унарными, бинарными и трехкомпонентными операторами состоит в количестве необходимых для них операндов (им требуются один, два и три операнда соответственно).
4. Наилучший способ установки собственной приоритетности операторов состоит в заключении части выражения, которая должна быть вычислена в первую очередь, в круглые скобки.
5. Оператор тождественности используется в тех случаях, когда нужно обойти присущее JavaScript автоматическое изменение типа операнда.
6. Самыми простыми формами выражений являются литералы (такие как числа и строки) и переменные, которые просто вычисляются в самих себя.
7. К трем условным инструкциям относятся `if`, `switch` и оператор `?`.
8. Большинство условных выражений в инструкциях `if` и `while` являются литералами (или логическими выражениями), поэтому инициируют выполнение кода только в том случае, если вычисляются как `true`. Числовые выражения инициируют выполнение, когда результатом их вычисления является ненулевое значение. Строковые выражения инициируют выполнение, когда вычисляются как непустая строка. Значение `NULL` вычисляется как ложное, поэтому не инициирует выполнение кода.
9. Циклы, в которых используется инструкция `for`, считаются более мощными по сравнению с циклами `while`, потому что они поддерживают два дополнительных параметра, управляющих работой цикла.
10. Инструкция `with` воспринимает в качестве своего параметра объект. При ее использовании объект указывается только один раз, а затем внутри блока `with` устанавливается связь этого объекта с каждой инструкцией.

Глава 15

1. Имена функций и переменных в JavaScript чувствительны к регистру используемых в них букв. Имена `Count`, `count` и `COUNT` представляют совершенно разные переменные.
2. Для создания функции, которая воспринимает и обрабатывает неограниченное количество параметров, доступ к параметрам организуется через массив `arguments`, являющийся составной частью всех функций.
3. Один из способов возвращения из функции нескольких значений заключается в помещении всех этих значений в массив с последующим его возвращением.
4. При определении класса для ссылки на текущий объект используется ключевое слово `this`.
5. Методы класса не обязательно должны определяться внутри определения самого класса. Если метод класса определяется за пределами конструктора, имя этого метода должно быть присвоено объекту `this` внутри определения класса.
6. Новый объект создается с помощью ключевого слова `new`.
7. Доступность свойства или метода может быть обеспечена всем объектам класса без тиражирования этого свойства или метода внутри объекта путем использования ключевого слова `prototype` для создания единственного экземпляра, который затем передается по ссылке всем объектам класса.
8. Для создания многомерного массива внутри основного массива помещается подмассив.
9. Синтаксис, который следует использовать для создания ассоциативного массива, имеет структуру *`ключ : значение`*, заключенную в фигурные скобки, как показано в следующем примере:

```
assocarray =
{
  "forename" : "Paul",
  "surname"  : "McCartney",
  "group"    : "The Beatles"
}
```

10. Инструкция, используемая для сортировки в убывающем порядке массива, состоящего из чисел, будет иметь следующий вид:

```
numbers.sort(function(a,b){return b - a})
```

Глава 16

1. Отправить данные формы на проверку до их отправки на сервер можно добавлением к тегу `<form>` JavaScript-атрибута `onSubmit`. Функция проверки должна возвращать `true`, если форма должна быть отправлена на сервер, или `false`, если она не прошла проверку.

2. Для проверки соответствия строки регулярному выражению в JavaScript используется метод `test`.
3. Регулярные выражения, соответствующие символам, не использующимся в словах, могут иметь вид `/[^\w]/`, `/[\W]/`, `/[^a-zA-Z0-9_]/` и т. д.
4. Для проверки соответствия как слову *fox*, так и слову *fix* можно воспользоваться регулярным выражением `/f[oi]x/`.
5. Регулярное выражение, соответствующее любому отдельному слову, за которым следует символ, не использующийся в словах, может иметь следующий вид: `/\w+\W/g`.
6. Функция JavaScript, использующая регулярное выражение для проверки наличия слова *fox* в строке `The quick brown fox`, может иметь такой вид:

```
document.write(/fox/.test("The quick brown fox"))
```

7. Функция PHP, использующая регулярное выражение для замены всех экземпляров слова the в строке `The cow jumps over the moon` словом my, может иметь следующий вид:

```
$s=preg_replace("/the/i", "my", "The cow jumps over the moon");
```

8. Для предварительного заполнения полей формы значениями применяется HTML-атрибут `value`, который помещается в тег `<input>` и принимает там форму `value="`*`значение`*`"`.

Глава 17

1. Необходимость функции для создания новых объектов `XMLHTTPRequest` обусловлена тем, что браузеры Microsoft используют два разных метода их создания, а все остальные основные браузеры применяют третий, совершенно иной метод. За счет функции, тестирующей используемый браузер, можно обеспечить работу кода на всех основных браузерах.
2. Цель применения конструкции `try...catch` состоит в настройке на перехват ошибки при выполнении кода, находящегося внутри инструкции `try`. Если его выполнение вызовет ошибку, то вместо выдачи общей ошибки будет выполнен код блока `catch`.
3. У объекта `XMLHTTPRequest` имеется шесть свойств и шесть методов (см. табл. 17.1 и 17.2).
4. Определить завершение асинхронного вызова можно по значению `4`, которое примет свойство `readyState`.
5. Когда асинхронный вызов успешно завершится, принадлежащее объекту свойство `status` получит значение `200`.
6. Значение, возвращенное успешно завершенным асинхронным вызовом, содержится в свойстве `responseText` объекта `XMLHTTPRequest`.

7. DOM-дерево, созданное из XML, возвращенного успешно завершенным асинхронным вызовом, хранится в свойстве `responseXML` объекта `XMLHTTPRequest`.
8. Для указания функции обратного вызова, обрабатывающей ответы асинхронных вызовов, нужно присвоить имя функции свойству `onreadystatechange` объекта `XMLHTTPRequest`. Можно также воспользоваться безымянной встроенной функцией.
9. Для инициирования асинхронного запроса вызывается метод `send` объекта `XMLHTTPRequest`.
10. Основное различие между GET- и POST-запросами асинхронных вызовов состоит в том, что GET-запросы присоединяют данные к URL-адресу, а POST-запросы передают данные в качестве параметра метода `send` и требуют правильной формы заголовков, отправляемых в первоочередном порядке.

Глава 18

1. Чтобы импортировать одну таблицу стилей в другую, используется инструкция `@import`, например:

```
@import url('styles.css');
```

2. Чтобы импортировать таблицу стилей в документ, можно воспользоваться HTML-тегом `<link>`, например:

```
<link rel='stylesheet' href='styles.css'>
```

3. Чтобы непосредственно встроить стиль в элемент, применяется атрибут `style`, например:

```
<div style='color:blue;'>
```

4. Разница между идентификатором CSS и классом CSS заключается в том, что идентификатор применяется только к одному элементу, а класс можно применить ко многим элементам.
5. В CSS-объявлениях для имен идентификаторов в качестве префикса используется символ решетки (`#`), а для имен классов — символ точки (`.`), например `#myid` и `.myclass`.
6. Точка с запятой (`;`) в CSS используется в качестве разделителя объявлений.
7. Чтобы добавить к таблице стилей комментарий, его нужно поставить между маркерами открытия и закрытия комментария: `/*` и `*/`.
8. В CSS указать на соответствие любому элементу можно с помощью универсального селектора `*`.
9. Чтобы выбрать в CSS группу разных элементов и (или) типов элементов, между каждым элементом, идентификатором, классом и т. д. нужно ставить запятую.

10. Чтобы одно CSS-объявление из пары, имеющей одинаковые приоритеты, получило преимущество над другим, к нему добавляют объявление `!important`, например:

```
p{ color:#ff0000 !important; }
```

Глава 19

1. CSS3-операторы `^=`, `$=` и `*=` в указанном порядке соответствуют началу, концу и любой части строки.
2. Для указания размера фонового изображения предназначено свойство `background-size`, например:

```
background-size:800px 600px;
```

3. Радиус границы можно указать с помощью свойства `border-radius`, например:

```
border-radius:20px;
```

4. Перетекание текста по нескольким колонкам можно задать с помощью свойств `column-count`, `column-gap` и `column-rule` или их вариантов, характерных для того или иного браузера, например:

```
column-count: 3;
    column-gap :1em;
    column-rule :1px solid black;
```

5. Четырьмя функциями, с помощью которых можно указать CSS-цвета, являются `hsl`, `hsla`, `rgb` и `rgba`. Например:

```
color:rgba(0%,60%,40%,0.4);
```

6. Чтобы создать под каким-нибудь текстом серую тень с диагональным отступом вправо и вниз на 5 пикселов и с размытостью 3 пиксела, можно воспользоваться следующим объявлением:

```
text-shadow: 5px 5px 3px #888;
```

7. Показать с помощью многоточия, что текст усечен, можно таким объявлением:

```
text-overflow:ellipsis;
```

8. Чтобы включить в состав своей веб-страницы веб-шрифты Google, нужно сначала их выбрать с сайта http://fonts.google.com, затем скопировать предоставляемый тег `<link>` в раздел `<head>` своего HTML-документа. Это будет выглядеть примерно так:

```
<link href='http://fonts.googleapis.com/css?family=Lobster'
      rel='stylesheet' type='text/css'>
```

   Затем можно сослаться на шрифт в CSS-объявлении, например в таком:

```
h1{ font-family:'Lobster', arial, serif; }
```

9. Для поворота объекта на 90° нужно воспользоваться следующим CSS-объявлением:

```
transform:rotate(90deg);
```

10. Чтобы указать переход объекта таким образом, чтобы при изменении любого из его свойств переход осуществлялся сразу в линейном режиме в течение 0,5 с, нужно воспользоваться следующим объявлением:

```
transition:all .5s linear;
```

Глава 20

1. Функция `O` возвращает объект по его идентификатору, функция `S` возвращает свойство стиля объекта, а функция `C` возвращает массив всех объектов, к которым обращается заданный класс.
2. Изменить CSS-атрибут объекта можно с помощью функции `setAttribute`, например:

```
myobject.setAttribute('font-size', '16pt')
```

   Можно также изменить атрибут непосредственно (что обычно и делается), используя немного измененные имена там, где это требуется, например:

```
myobject.fontSize = '16pt'
```

3. Свойствами, предоставляющими доступную в окне браузера ширину и высоту, являются `window.innerHeight` и `window.innerWidth`.
4. Задать какие-нибудь действия при прохождении указателя мыши над объектом, а затем при выходе за границы объекта можно, привязав код, совершающий эти действия, к событиям `onmouseover` и `onmouseout`.
5. Для создания нового элемента нужно воспользоваться таким кодом, как, например:

```
elem = document.createElement('span')
```

   Для добавления нового элемента к DOM применяется, например, такой код:

```
document.body.appendChild(elem)
```

6. Чтобы сделать элемент невидимым, нужно установить для его свойства `visibility` значение `hidden` (а для возвращения ему видимости нужно задать значение `visible`). Чтобы сжать элемент до нулевых размеров, следует выбрать для его свойства `display` значение `none` (а для восстановления его размеров — значение `block`).
7. Чтобы задать одиночное событие в будущем времени, нужно вызвать функцию `setTimeout`, передав ей код или имя функции для выполнения и значение задержки в миллисекундах.
8. Для установки повторяющегося события через указанный интервал времени нужно вызвать функцию `setInterval`, передав ей код или имя функции для выполнения и значение задержки между повторениями в миллисекундах.
9. Чтобы освободить элемент от его места на веб-странице, позволив ему перемещаться, нужно установить для его свойства `position` значение `'relative'`, `'absolute'` или `'fixed'`. Для восстановления элемента на его исходном месте этому свойству нужно присвоить значение `'static'`.

10. Для получения скорости анимации 50 кадров в секунду нужно установить задержку между прерываниями равной 20 мс. Для вычисления этого значения необходимо разделить 1000 мс на желаемую скорость анимации.

Глава 21

1. Обычно в качестве фабричного метода для создания объектов jQuery используется символ `$`. Вместо него можно воспользоваться методом по имени `jQuery`.
2. Чтобы создать ссылку на минимизированную версию выпуска 3.2.1 jQuery, получаемую из Google CDN, можно воспользоваться следующим кодом HTML:

```
<script src='http://ajax.googleapis.com
/ajax/libs/jquery/3.2.1/jquery.min.js'></script>
```

3. Для фабричного метода jQuery `$` приемлемы CSS-селекторы, применяемые с целью создания jQuery-объекта из соответствующих этим селекторам элементов.
4. Для получения значения CSS-свойства используется метод `css`, которому передается название свойства. Для установки значения свойства этому методу передаются название свойства и значение.
5. Для прикрепления метода к событию щелчка на элементе с идентификатором `elem` с целью медленного исчезновения этого элемента можно воспользоваться следующим кодом:

```
$('#elem').click(function() { $(this).hide('slow') } )
```

6. Для применения к элементу эффекта анимации его свойству `position` нужно присвоить значение `fixed`, `relative` или `absolute`.
7. Одновременного (или последовательного — в случае применения анимации) запуска сразу нескольких методов можно добиться, связав их в цепочку с помощью символов точки:

```
$('#elem').css('color', 'blue').css('background',
  'yellow').slideUp('slow')
```

8. Для извлечения объекта элемента из объекта, выбранного средствами jQuery, можно указать его индекс в квадратных скобках, например:

```
$('#elem')[0]
```

   или же воспользоваться методом `get`:

```
$('#elem').get(0)
```

9. Для отображения одноуровневого элемента, следующего непосредственно перед элементом с идентификатором `news`, полужирным шрифтом, можно воспользоваться такой инструкцией:

```
$('#news').prev().css('font-weight', 'bold')
```

10. Средствами jQuery Get-запрос по технологии асинхронного обмена данными можно сделать с помощью метода `$.get`:

```
$.get('http://server.com/ajax.php?do=this', function(data) {
  alert('The server said: ' + data) } )
```

Глава 22

1. Использование CDN для доставки файлов означает, что вам не нужно полагаться на собственную (или на клиентскую) загрузку сети, что может сэкономить деньги. Кроме того, можно создать у пользователя ощущение ускорения работы, поскольку после загрузки браузером файла та же самая его версия может быть заново загружена локально из кэша. Недостаток заключается в том, что ваш веб-сайт или веб-приложение не смогут выполняться в локальной среде, если браузер пользователя в текущий момент не подключен к интернету.
2. Чтобы определить страницу содержимого для jQuery Mobile, ее нужно заключить в `<div>`-элемент с атрибутом `data-role`, имеющим значение `page`.
3. Тремя основными частями, из которых складывается страница jQuery, являются ее заголовок, содержимое и нижний колонтитул. Для их обозначения эти части помещаются в `<div>`-элементы, которые, соответственно, имеют атрибуты `data-role` с присвоенными значениями `header`, `content` и `footer`. Эти три элемента должны быть дочерними по отношению к родительскому `<div>`-элементу, рассмотренному в вопросе 2.
4. Чтобы в один HTML-документ поместить несколько страниц jQuery Mobile, в него можно включить несколько родительских `<div>`-элементов, имеющих атрибут `data-role` со значением `page`, каждый из которых будет содержать три дочерних `<div>`-элемента, рассмотренных в вопросе 3. Чтобы ссылаться на эти страницы, каждому из родительских элементов следует присвоить уникальный идентификатор (например, `id="news"`), на который затем можно будет сослаться из любого места HTML-документа, используя привязку, например `<a href="#news">`.
5. Чтобы предотвратить асинхронную загрузку веб-страницы, можно либо свойству `data-ajax` привязки (`anchor`) или формы (`form`) задать значение `false`, привоив их атрибуту `rel` значение `external`, либо предоставить значение их атрибуту `target`.
6. Чтобы настроить страничный переход по гипертекстовой ссылке на применение эффекта `flip`, нужно задать ее атрибуту `data-transition` значение `flip` (или же воспользоваться любым другим поддерживаемым значением для других доступных эффектов перехода, например `data-transition="pop"`).
7. Чтобы страница показывалась в виде диалогового окна, ее можно загрузить, задав ее атрибуту `data-rel` значение `dialog`.
8. Чтобы придать обычной гипертекстовой ссылке внешний вид кнопки, нужно задать ее атрибуту `data-role` значение `button`.
9. Чтобы добиться линейного показа элемента jQuery Mobile, нужно задать его атрибуту `data-inline` значение `true`.
10. Чтобы добавить к кнопке значок, нужно предоставить имя известного значка jQuery Mobile в качестве значения ее атрибута `data-icon`, например `data-icon="gear"`.

Глава 23

1. Новым элементом, появившимся в HTML5 и позволяющим рисовать графические изображения на веб-странице, является элемент холста, создаваемый с помощью тега `<canvas>`.
2. Для доступа ко многим улучшенным функциям HTML5, например к холстам или геолокации, нужно использовать код JavaScript.
3. Для внедрения аудио и видео в веб-страницу нужно применять теги `<audio>` или `<video>`.
4. В HTML5 появилось локальное хранилище, предлагающее более высокие возможности по доступу к локальному пользовательскому пространству памяти по сравнению с cookie, ограниченными объемом хранимых данных.
5. В HTML5 для выполнения нужных вам фоновых задач можно установить рабочие веб-процессы, представляющие собой простые разделы кода JavaScript.

Глава 24

1. Для создания в HTML элемента холста нужно добавить тег `<canvas>` и указать идентификатор, который может использоваться кодом JavaScript для доступа к этому холсту:

   ```
   <canvas id='mycanvas'>
   ```

2. Чтобы дать коду JavaScript доступ к элементу холста, нужно обеспечить предоставление элементу идентификатора, например `mycanvas`, а затем воспользоваться функцией `document.getElementdByld` (или функцией `O` из файла `OSC.js`, расположенного с архивом примеров на сайте, сопутствующем книге) для возвращения объекта, представляющего элемент. А для извлечения двумерного содержимого холста нужно вызвать для объекта метод `getContext`:

   ```
   canvas  = document.getElementByld('mycanvas')
   context = canvas.getContext('2d')
   ```

3. Для указания начала пути холста нужно вызвать для объекта `context` метод `beginPath`. После создания пути его нужно закрыть, вызвав для объекта `context` метод `closePath`:

   ```
   context.beginPath()
     // Сюда помещаются команды создания пути
   context.closePath()
   ```

4. Извлечь данные с холста можно с помощью метода `toDataURL`, присвоив потом эти данные свойству `src` объекта `image`:

   ```
   image.src = canvas.toDataURL()
   ```

5. Для создания градиентной заливки (радиальной или линейной) более чем с двумя цветами нужно с помощью цветовых опорных точек указать заранее созданному объекту `gradient` все требуемые цвета и назначить каждому из них

начальную точку в виде долевого значения от полного градиента (в диапазоне от `0` до `1`):

```
gradient.addColorStop(0,    'green')
gradient.addColorStop(0.3,  'red')
gradient.addColorStop(0,79, 'orange')
gradient.addColorStop(1,    'brown')
```

6. Для настройки ширины линий при рисовании нужно присвоить значение свойству `lineWidth` объекта `context`:

```
context.lineWidth = 5
```

7. Чтобы обеспечить рисование только внутри определенной области, можно создать путь, а затем вызвать метод `clip`.
8. Сложная кривая с двумя воображаемыми точками притяжения называется кривой Безье. Для ее создания нужно вызвать метод `bezierCurveTo`, предоставив для точек притяжения две пары координат x и y, за которыми необходимо указать еще одну пару координат для конечной точки кривой. После этого будет создана кривая от текущего места рисования и до указанного места назначения.
9. Метод `getImageData` возвращает массив, состоящий из пиксельных данных указанной области с элементами, последовательно содержащими значения красной, зеленой и синей составляющей цвета пиксела, а также значение его альфа-прозрачности. Следовательно, для каждого пиксела возвращается четыре элемента данных.
10. Методу `transform` передаются шесть аргументов (или параметров) в следующем порядке: горизонтальное масштабирование, горизонтальный наклон, вертикальный наклон, вертикальное масштабирование, горизонтальное перемещение, вертикальное перемещение. Следовательно, аргументы, применяемые к масштабированию, имеют в этом списке номера 1 и 4.

Глава 25

1. Для вставки аудио и видео в документ HTML5 предназначены теги `<audio>` и `<video>`.
2. Чтобы гарантировать максимальную возможность проигрывания аудио на всех платформах, нужно использовать как минимум два кодека: PCM и любой из других или Vorbis и любой из других.
3. Для проигрывания медиа в HTML5 и для постановки его на паузу можно вызывать методы `play` и `pause` элемента `<audio>` или элемента `<video>`.
4. Чтобы поддержать проигрывание медиа в браузерах, не работающих с HTML5, можно вставить Flash-аудио- или видеопроигрыватель в любой элемент `<audio>` или `<video>`. В том случае если не будет поддерживаться проигрывание HTML5- медиа, он будет активизирован.

5. Чтобы гарантировать максимальную возможность проигрывания видео на всех платформах, нужно использовать кодек MP4/H.264 и кодек Ogg/Theora или VP8 для поддержки браузера Opera.

Глава 26

1. Для запроса у браузера геолокационных данных вызывается следующий метод, которому передаются имена двух функций, написанных вами для обработки доступа или отказа в получении данных:

   ```
   navigator.geolocation.getCurrentPosition(granted, denied)
   ```

2. Чтобы определить возможность поддержки браузером локального хранилища, нужно протестировать наличие у объекта `localStorage` свойства `typeof`:

   ```
   if (typeof localStorage == 'undefined')
     // Локальное хранилище недоступно
   ```

3. Чтобы удалить все данные локального хранилища для текущего домена, можно вызвать метод `localStorage.clear`.

4. Для связи рабочих веб-процессов с основной программой проще всего воспользоваться для отправки информации методом `postMessage`. Для извлечения этой информации программа прикрепляет к объекту рабочего веб-процесса событие `onmessage`.

5. Чтобы предотвратить запуск рабочего веб-процесса, следует вызвать метод `terminate` объекта `worker`, например `worker.terminate()`.

6. Предотвратить действие по умолчанию, не позволяющее осуществлять перетаскивание для событий, управляющих операциями перетаскивания, можно путем вызова в ваших обработчиках событий `ondragover` и `ondrop` метода `preventDefault` в отношении объекта события.

7. Чтобы сделать обмен сообщениями между документами более безопасным, нужно при отправке сообщений всегда предоставлять идентификатор домена:

   ```
   postMessage(message, 'http://mydomain.com')
   ```

   а при их получении проверять этот идентификатор:

   ```
   if (event.origin) != 'http://mydomain.com') // Запрещение
   ```

8. Можно также зашифровать данные или сделать обмен данными непонятным для непосвященных, чтобы воспрепятствовать вредоносной инъекции или перехвату.

9. Микроданные предназначены для того, чтобы сделать информацию понятнее для компьютерных программ, например для поисковых движков.

Б Интернет-ресурсы

В этом приложении перечислены полезные сайты, предоставляющие материал, использованный в данной книге, и другие ресурсы для совершенствования ваших веб-приложений.

Сайты, относящиеся к PHP

- http://ampps.com;
- http://codewalkers.com;
- http://easyphp.org;
- http://forums.devshed.com;
- http://hotscripts.com/category/php/;
- http://htmlgoodies.com/beyond/php/;
- http://php.net;
- http://phpbuilder.com;
- http://phpfreaks.com;
- http://phpunit.de;
- http://w3schools.com/php/;
- http://zend.com.

Сайты, относящиеся к MySQL

- http://launchpad.net/mysql;
- http://mysql.com;
- http://php.net/mysql;
- http://planetmysql.org;

- http://oracle.com/us/sun;
- http://w3schools.com/PHP/php_mysql_i.

Сайты, относящиеся к JavaScript

- http://developer.moziNa.org/en/JavaScript;
- http://dynamicdrive.com;
- http://javascript.about.com;
- http://javascript.com;
- http://javascriptkit.com;
- http://w3schools.com/JS;
- http://www.webreference.com/js.

Сайты, относящиеся к CSS

- http://cssbasics.com;
- http://css-discuss.incutio.com/wiki/Print_Stylesheets;
- http://freehtmlvalidator.com;
- http://quirksmode.org/css/quirksmode.html.

Сайты, относящиеся к HTML5

- http://caniuse.com;
- http://html5demos.com;
- http://html5doctor.com;
- http://html5readiness.com;
- http://html5test.com;
- http://htmlvalidator.com;
- http://modernizr.com.

Сайты, относящиеся к асинхронному обмену данными

- http://ajax.asp.net;
- http://ajaxian.com;
- http://developer.mozilla.org/en/AJAX;

- http://dojotoolkit.org;
- http://jquery.com;
- http://jquerymobile.com;
- http://mootools.net;
- http://openjs.com;
- http://prototypejs.org.

Сайты с разнообразными ресурсами

- http://apachefriends.org;
- http://easyphp.org;
- http://eclipse.org;
- http://editra.org;
- http://mamp.info/en;
- http://programmingforums.org;
- http://putty.org;
- http://sourceforge.net/projects/glossword.

B Стоповые слова MySQL для FULLTEXT

В этом приложении содержится более 500 стоповых слов (stopwords), упоминавшихся в пункте «Создание индекса FULLTEXT» на с. 223. Стоповыми называются слова, которые считаются настолько распространенными, что не могут представлять ценности для поиска или хранения в индексе FULLTEXT. Теоретически игнорирование этих слов слабо влияет на результаты большинства поисков в режиме FULLTEXT, но позволяет существенно сократить объем и повысить эффективность работы баз данных MySQL. Здесь приведены слова в нижнем регистре, но этот перечень касается также слов, представленных в верхнем и смешанном регистрах.

A

A's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, asking, associated, at, available, away, awfully.

B

Be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by.

C

C'mon, c's, came, can, can't, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, couldn't, course, currently.

D

Definitely, described, despite, did, didn't, different, do, does, doesn't, doing, don't, done, down, downwards, during.

E

Each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except.

F

Far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore.

G

Get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings.

H

Had, hadn't, happens, hardly, has, hasn't, have, haven't, having, he, he's, hello, help, hence, her, here, here's, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however.

I

I'd, i'll, i'm, i've, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, isn't, it, it'd, it'll, it's, its, itself.

J

Just.

K

Keep, keeps, kept, know, knows, known.

L

Last, lately, later, latter, latterly, least, less, lest, let, let's, like, liked, likely, little, look, looking, looks, ltd.

M

Mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself.

N

Name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere.

O

Obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own.

P

Particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides.

Q

Que, quite, qv.

R

Rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right.

S

Said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, shouldn't, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure.

T

T's, take, taken, tell, tends, th, than, thank, thanks, thanx, that, that's, thats, the, their, theirs, them, themselves, then, thence, there, there's, thereafter, thereby, therefore, therein, theres, thereupon, these, they, they'd, they'll, they're, they've, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two.

U

Un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually.

V

Value, various, very, via, viz, vs.

W

Want, wants, was, wasn't, way, we, we'd, we'll, we're, we've, welcome, well, went, were, weren't, what, what's, whatever, when, whence, whenever, where, where's, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, who's, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, won't, wonder, would, would, wouldn't.

Y

Yes, yet, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves.

Z

Zero.

Г Функции MySQL

За счет функций, встроенных в MySQL, существенно сокращается время выполнения сложных запросов и упрощается их конструкция. Если есть желание получить более полную информацию обо всех доступных функциях, можете обратиться к материалам сайтов по следующим URL-адресам:

- строковые функции: tinyurl.com/phpstringfuncs;
- функции даты и времени: tinyurl.com\phpdateandtime.

Но для ускорения далее приводятся описания наиболее востребованных функций MySQL.

Строковые функции

Перечислим основные функции для работы со строками.

- `CONCAT(str1, str2, ...)`

  Возвращает результат объединения *str1*, *str2* и любых других параметров (или `NULL`, если все аргументы имеют значение `NULL`). Если какой-нибудь из аргументов имеет двоичную форму, то результат будет возвращен в виде двоичной последовательности. В противном случае результат будет в виде строки, не имеющей двоичного формата. Например, следующий код возвращает строку MySQL:

  ```
  SELECT CONCAT('My', 'S', 'QL');
  ```

- `CONCAT_WS(separator, str1, str2, ...)`

  Эта функция работает так же, как и `CONCAT`, за исключением того, что между объединяемыми элементами она вставляет разделитель. Если разделитель имеет значение `NULL`, то результат тоже будет `NULL`, но значения `NULL` могут ис-

пользоваться и в качестве других аргументов, которые в таком случае будут пропущены. Следующий код возвращает строку `Truman,Harry,S`:

```
SELECT CONCAT_WS(',', 'Truman', 'Harry', 'S');
```

- `LEFT(str, len)`

  Возвращает *len* крайних слева символов из строки *str* (или `NULL`, если какой-нибудь из аргументов имеет значение `NULL`). Следующий код возвращает строку `Chris`:

```
SELECT LEFT('Christopher Columbus', '5');
```

- `RIGHT(str, len)`

  Возвращает *len* крайних справа символов из строки *str* (или `NULL`, если какой-нибудь из аргументов имеет значение `NULL`). Следующий код возвращает строку `Columbus`:

```
SELECT RIGHT('Christopher Columbus', '8');
```

- `MID(str, pos, len)`

  Возвращает до *len* символов из строки *str*, начиная с позиции *pos*. Если аргумент *len* опущен, то возвращаются все символы до конца строки. Для аргумента *pos* можно использовать отрицательное значение, тогда он будет представлять позицию символа, вычисляемую с конца строки. Первой позицией в строке является `1`. Следующий код возвращает строку `"stop"`:

```
SELECT MID('Christopher Columbus', '5', '4');
```

- `LENGTH(str)`

  Возвращает длину строки *str* в байтах. Учтите, что при встрече многобайтовых символов учитываются все их байты. Если нужно узнать количество символов в строке, необходимо воспользоваться функцией `CHAR_LENGTH`. Следующий код возвращает значение `15`:

```
SELECT LENGTH('Mark Zuckerberg');
```

- `LPAD(str, len, padstr)`

  Возвращает строку *str*, дополненную до длины Ten символами *padstr*, добавляемыми в начало строки. Если строка *str* длиннее, чем *len*, то строка возвращается усеченной до *len* символов. Этот код:

```
SELECT LPAD('January', '8', ' ');
SELECT LPAD('February', '8', ' ');
SELECT LPAD('March', '8', ' ');
SELECT LPAD('April', '8', ' ');
SELECT LPAD('May', '8', ' ');
```

  вернет следующие строки:

```
 January
February
   March
   April
     May
```

  Обратите внимание на то, как все строки были дополнены до восьми символов.

- RPAD(*str*, *len*, *padstr*)

  Эта функция работает так же, как и функция LPAD, за исключением того, что она возвращает строку, дополненную символами не слева, а справа. Следующий код возвращает строку "Hi!!!":

  ```
  SELECT RPADC'Hi', '5', '!');
  ```

- LOCATE(*substr*, *str*, *pos*)

  Возвращает позицию первой же встреченной подстроки *substr* в строке *str*. Если функции передан параметр *pos*, то поиск начинается с позиции *pos*. Если *substr* не была найдена в строке *str*, возвращается значение 0.

  Следующий код возвращает значения 5 и 11, поскольку при вызове первой функции возвращается позиция первого встреченного слова unit, а вторая функция начинает поиск только с седьмого символа и поэтому выводит позицию второго появления этого слова в строке:

  ```
  SELECT LOCATE('unit'. 'Community unit');
  SELECT LOCATE('unit', 'Community unit', 7);
  ```

- LOWER(*str*)

  Эта функция является прямой противоположностью функции UPPER. Она возвращает строку *str*, все буквы которой переводятся в нижний регистр. Следующий код возвращает строку "queen elizabeth ii":

  ```
  SELECT LOWER('Queen Elizabeth II');
  ```

- UPPER(*str*)

  Эта функция является прямой противоположностью функции LOWER. Она возвращает строку *str*, все буквы которой переводятся в верхний регистр. Следующий код возвращает строку "I CAN'T HELP SHOUTING":

  ```
  SELECT UPPER('I can't help shouting');
  ```

- QUOTE(*str*)

  Возвращает строку, помещенную в кавычки, которая будет готова к использованию в инструкции SQL, для чего в ней отключаются все неоднозначно толкуемые символы. Возвращаемая строка заключается в одинарные кавычки, а перед всеми имеющимися в ней одинарными кавычками, обратными слешами, ASCII-символами NUL и Ctrl-Z устанавливается обратный слеш. Если аргумент имеет значение NULL, возвращаемое значение является словом NULL, не заключенным в кавычки. Код примера возвращает следующую строку:

  ```
  'I\'m hungry'
  ```

  Обратите внимание на то, как символ одинарной кавычки (') был заменен символами /'.

  ```
  SELECT QUOTE("I'm hungry");
  ```

- REPEAT(*str*, *count*)

  Возвращает строку, содержащую *count* копий строки *str*. Если *count* меньше единицы, возвращается пустая строка. Если какой-нибудь из параметров имеет значение NULL, возвращается NULL. Следующий код возвращает строки "Ho Ho Ho" и "Merry Christmas":

- `REPLACE(str, from, to)`

  Возвращает строку *str*, в которой все появления строки *from* заменены строкой *to*. При проведении поиска подстроки *from* поиск и замена чувствительны к регистру. Следующий код возвращает строку `"Cheeseburger and Coke"`:

  ```
  SELECT REPLACE('Cheeseburger and Fries', 'Fries', 'Coke');
  ```

- `TRIM([specifier remove FROM] str)`

  Возвращает строку *str*, из которой удалены все префиксы и (или) суффиксы, имеющие значение *remove*. В качестве *specifier* может быть указан один из спецификаторов: `BOTH` (оба), `LEADING` (ведущие) или `TRAILING` (замыкающие). Если спецификатор не указан, предполагается спецификатор `BOTH`. Строка *remove* является необязательным параметром, и при ее отсутствии удаляются пробелы. Следующий код возвращает строки `No Padding` и `Hello`:

  ```
  SELECT TRIM(' No Padding ');
  SELECT TRIM(LEADING '_' FROM ' Hello ');
  ```

- `LTRIM(str)`

  Возвращает строку *str*, у которой удалены все начальные пробелы. Следующий код возвращает строку `"No Padding "`:

  ```
  SELECT LTRIM('     No Padding     ');
  ```

- `RTRIM(str)`

  Возвращает строку *str*, у которой удалены все замыкающие пробелы. Следующий код возвращает строку `"  No Padding"`:

  ```
  SELECT RTRIM('      No Padding      ');
  ```

Функции для работы с датами

Даты — важная составная часть большинства баз данных. При проведении финансовых транзакций должны записываться даты, при повторных выставлениях счетов должны учитываться сроки истечения действия кредитных карт и т. д. Поэтому неудивительно, что в MySQL имеется широкий спектр функций для работы с датами.

- `CURDATE()`

  Возвращает текущую дату в формате `YYYY-MM-DD` или `YYYMMDD` в зависимости от того, в каком контексте используется функция: строковом или числовом. 2 мая 2018 года следующий код вернул бы значения `2018-05-02` и `20180502`:

  ```
  SELECT CURDATE();
  SELECT CURDATE() + 0;
  ```

- `DATE(expr)`

  Извлекает дату из выражения `DATETIME`, переданного в аргументе *expr*. Следующий код возвращает значение `1961-05-02`:

- `DATE_ADD(date, INTERVAL expr unit)`

  Возвращает результат добавления выражения *expr*, в котором к дате применяется единица измерения *unit*. Аргумент *date* является стартовой датой или

значением DATETIME, а *expr* для отрицательных интервалов может начинаться с минуса (-). В табл. ПГ.1 показаны типы интервалов, поддерживаемые и ожидаемые в качестве значений *expr*. Обратите внимание на приведенные в этой таблице примеры, которые показывают, в каких случаях значение *expr* должно быть заключено в кавычки, чтобы база данных MySQL смогла их правильно интерпретировать (при любых сомнениях лучше добавить кавычки, которые не помешают работе).

Таблица ПГ.1. Ожидаемые значения expr

Тип	Ожидаемое значение expr	Пример
MICROSECOND	МИКРОСЕКУНДЫ	111111
SECOND	СЕКУНДЫ	11
MINUTE	МИНУТЫ	11
HOUR	ЧАСЫ	11
DAY	ДНИ	11
WEEK	НЕДЕЛИ	11
MONTH	МЕСЯЦЫ	11
QUARTER	КВАРТАЛЫ	1
YEAR	ГОДЫ	11
SECOND_MICROSECOND	'СЕКУНДЫ.МИКРОСЕКУНДЫ'	11.22
MINUTE_MICROSECOND	'МИНУТЫ.МИКРОСЕКУНДЫ'	11.22
MINUTE_SECOND	'МИНУТЫ:СЕКУНДЫ'	'11:22'
HOUR_MICROSECOND	'ЧАСЫ.МИКРОСЕКУНДЫ'	11.22
HOUR_SECOND	'ЧАСЫ:МИНУТЫ:СЕКУНДЫ'	'11:22:33'
HOUR_MINUTE	'ЧАСЫ:МИНУТЫ'	'11:22'
DAY_MICROSECOND	'ДНИ.МИКРОСЕКУНДЫ'	11.22
DAY_SECOND	'ДНИ ЧАСЫ:МИНУТЫ:СЕКУНДЫ'	'11 22:33:44'
DAY_MINUTE	'ДНИ ЧАСЫ:МИНУТЫ'	'11 22:33'
DAY_HOUR	'ДНИ ЧАСЫ'	'11 22'
YEAR_MONTH	'ГОДЫ-МЕСЯЦЫ'	'11-2'

Для вычитания интервала дат можно также воспользоваться функцией DATE_SUB. Но функциями DATE_ADD и DATE_SUB можно вообще не пользоваться, поскольку MySQL допускает непосредственные арифметические операции с датами. Данный код:

```
SELECT DATE_ADD('1975-01-01', INTERVAL 77 DAY);
SELECT DATE SUB('1982-07-04', INTERVAL '3-11' YEAR MONTH);
SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;
SELECT '2000-01-01' - INTERVAL 1 SECOND;
```

возвращает следующие значения:

```
1975-03-19 1978-08-04 2019-01-01 00:00:00 1999-12-31 23:59:59
```

Обратите внимание на то, как в последних двух командах используются непосредственные арифметические операции с датами без обращения к функциям.

- `DATE_FORMAT(`*`date`*`,` *`format`*`)`

  Эта функция возвращает значение даты *`date`*, отформатированное в соответствии со строкой форматирования *`format`*. В табл. ПГ.2 показаны спецификаторы, которые можно указывать в строке форматирования *`format`*. Учтите, что символ `%` нужно ставить так, как показано в таблице, то есть впереди каждого спецификатора. Следующий код возвращает заданную дату и время в виде `Friday May 4th 2018 03:02 AM`:

```
SELECT DATE_FORMAT('2018-05-04 03:02:01', '%W %M %D %Y %h:%i %p');
```

Таблица ПГ.2. Спецификаторы, использующиеся в функции DATE_FORMAT

Спецификатор	Описание
%a	Сокращенное название дня недели (Sun — Sat)
%b	Сокращенное название месяца (Jan — Dec)
%c	Месяц в числовом формате (0-12)
%D	День месяца с английским суффиксом (0th, 1st, 2nd, 3rd...)
%d	День месяца в числовом формате (00–31)
%e	День месяца в числовом формате (0–31)
%f	Микросекунды (000000–999999)
%H	Час, две цифры (00–23)
%h	Час, две цифры (01–12)
%I	Час (01–12)
%i	Минуты в числовом формате, две цифры (00–59)
%j	День года (001–366)
%k	Час (0–23)
%l	Час (1–12)
%M	Название месяца (January — December)
%m	Месяц в числовом формате, две цифры (00–12)
%p	AM или PM (до или после полудня)
%r	Время в 12-часовом формате (hh:mm:ss, за которыми следует AM или PM)
%S	Секунды, две цифры (00–59)
%s	Секунды, две цифры (00–59)
%T	Время в 24-часовом формате (hh:mm:ss)
%U	Неделя (00–53), когда первым днем недели считается воскресенье
%u	Неделя (00–53), когда первым днем недели считается понедельник

Продолжение ⇩

Таблица ПГ.2 (продолжение)

Спецификатор	Описание
%V	Неделя (00–53), когда первым днем недели считается воскресенье; используется со спецификатором %X
%v	Неделя (00–53), когда первым днем недели считается понедельник; используется со спецификатором %x
%W	Название дня недели (Sunday — Saturday)
%w	День недели (0 — воскресенье — 6 — суббота)
%X	Год для недели, в которой первым днем считается воскресенье, в числовом формате, четыре цифры; используется вместе с %V
%x	Год для недели, в которой первым днем считается понедельник, в числовом формате, четыре цифры; используется вместе с %v
%Y	Год в числовом формате, четыре цифры
%y	Год в числовом формате, две цифры
%%	Символ % как таковой

- `DAY(`*`date`*`)`

  Возвращает для даты *`date`* день месяца в диапазоне от 1 до 31 или возвращает `0` для дат, содержащих нулевую составляющую дней, таких как `0000-00-00` или `2018-00-00`. Для возвращения таких же значений можно также воспользоваться функцией `DAYOFMONTH`. Следующий код возвращает значение `3`:

  ```
  SELECT DAY('2018-02-03');
  ```

- `DAYNAME(`*`date`*`)`

  Возвращает название дня недели для даты *`date`*. Следующий код возвращает строку `Saturday`:

  ```
  SELECT DAYNAME('2018-02-03');
  ```

- `DAYOFWEEK(`*`date`*`)`

  Возвращает номер дня недели для даты *`date`* в диапазоне от `1` для воскресенья до `7` для субботы. Следующий код возвращает значение `7`:

  ```
  SELECT DAYOFWEEK('2018-02-03');
  ```

- `DAYOFYEAR(`*`date`*`)`

  Возвращает день года для даты *`date`* в диапазоне от `1` до `366`. Следующий код возвращает значение `34`:

- `LAST_DAY(`*`date`*`)`

  Возвращает последний день месяца для заданной в формате `DATETIME` даты *`date`*. Если аргумент имеет неправильный формат, возвращает `NULL`. Данный код:

  ```
  SELECT LAST_DAY('2018-02-03');
  SELECT LAST_DAY('2018-03-11');
  SELECT LAST_DAY('2018-04-26');
  ```

возвращает следующие значения:

```
2018-02-28
2018-03-31
2018-04-30
```

- MAKEDATE(*year*, *dayofyear*)

  Возвращает дату, соответствующую предоставленному году *year* и дню года *dayofyear*. Если *dayofyear* имеет нулевое значение, результат будет в виде значения NULL. Следующий код возвращает дату 2018-10-01:

  ```
  SELECT MAKEDATE(2018,274);
  ```

- MONTH(*date*)

  Возвращает месяц даты *date* в диапазоне от 1 до 12 с января по декабрь. Для дат, у которых часть, относящаяся к месяцу, имеет нулевое значение, например 0000-00-00 или 2018-00-00, возвращает 0. Следующий код возвращает значение 7:

  ```
  SELECT MONTH('2018-07-11');
  ```

- MONTHNAME(*date*)

  Возвращает полное название месяца для даты *date*. Следующий код возвращает строку July:

  ```
  SELECT MONTHNAME('2018-07-11');
  ```

- SYSDATE()

  Возвращает текущую дату и время в виде значения в формате либо YYYY-MM-DD HH:MM:SS, либо YYYYMMDDHHMMSS, в зависимости от того, в каком контексте используется функция: строковом или числовом. Аналогичным образом работает функция NOW, за исключением того, что она возвращает время и дату только на момент запуска текущей инструкции, а функция SYSDATE возвращает время и дату именно на момент вызова самой функции. 19 декабря 2018 года в 19:11:13 следующий код вернет значения 2018-12-19 19:11:13 и 20181219191113:

  ```
  SELECT SYSDATE();
  SELECT SYSDATE() + 0;
  ```

- YEAR(*date*)

  Возвращает год для даты *date* в диапазоне от 1000 до 9999 или 0, если дата задана как 0000-00-00. Следующий код возвращает год 1999:

  ```
  SELECT YEAR('1999-08-07');
  ```

- WEEK(*date* [, *mode*])

  Возвращает номер недели для даты *date*. Если функции передан необязательный параметр *mode*, то возвращенный номер недели будет модифицирован в соответствии с описанием, приведенным в табл. ПГ.3. Можно также воспользоваться функцией WEEKOFYEAR, работа которой эквивалентна работе функции WEEK при использовании режима 3. Следующий код возвращает номер недели, равный 14:

  ```
  SELECT WEEK('2018-04-04', 1);
  ```

Таблица ПГ.3. Режимы работы, поддерживаемые функцией WEEK

Режим	Первый день недели	Диапазон	Когда неделя 1 — это первая неделя...
0	Воскресенье	0–53	С воскресеньем в этом году
1	Понедельник	0–53	С более чем тремя днями в этом году
2	Воскресенье	1–53	С воскресеньем в этом году
3	Понедельник	1–53	С более чем тремя днями в этом году
4	Воскресенье	0–53	С более чем тремя днями в этом году
5	Понедельник	0–53	С понедельником в этом году
6	Воскресенье	1–53	С более чем тремя днями в этом году
7	Понедельник	1–53	С понедельником в этом году

- `WEEKDAY(`*`date`*`)`

  Возвращает номер дня недели для даты *`date`* в диапазоне от `0` (понедельник) до `6` (суббота). Следующий код возвращает значение `2`:

  ```
  SELECT WEEKDAY('2018-04-04');
  ```

Функции для работы с временем

Иногда приходится работать не с датой, а с временем, и MySQL предоставляет для этого большое количество функций. Далее перечисляются наиболее востребованные из них.

- `CURTIME(`*`fsp`*`)`

  Возвращает текущее время (в часовом поясе пользователя) в виде значения, имеющего формат `HH:MM:SS` или `HHMMSS` в зависимости от того, в каком контексте используется функция: строковом или числовом. Если предоставлен необязательный аргумент *`fsp`* со значением в диапазоне от `0` до `6`, в возвращаемое значение включается дробная секундная часть с указанным в этом аргументе количеством цифр. При текущем времени 11:56:23 следующий код возвращает значения `11:56:23` и `115623.000000`:

  ```
  SELECT CURTIMEO SELECT CURTIMEO + 0;
  ```

- `HOUR(`*`time`*`)`

  Возвращает значение часа для времени *`time`*. Следующий код возвращает значение `11`:

  ```
  SELECT HOUR('11:56:23');
  ```

- `MINUTE(`*`time`*`)`

  Возвращает значение минуты для времени *`time`*. Следующий код возвращает значение `56`:

  ```
  SELECT MINUTE('11:56:23');
  ```

- SECOND(*time*)

  Возвращает значение секунды для времени *time*. Следующий код возвращает значение 23:

  ```
  SELECT SECOND('11:56:23');
  ```

- MAKETIME(*hour*, *minute*, *second*)

  Возвращает значение времени, вычисленное на основе аргументов часа *hour*, минуты *minute* и секунды *second*. Следующий код возвращает время 11:56:23:

  ```
  SELECT MAKETIME(11, 56, 23);
  ```

- TIMEDIFF(*expr1*, *expr2*)

  Возвращает разницу между *expr1* и *expr2* (*expr1* – *expr2*) в виде значения времени. Оба аргумента должны быть выражениями одинакового типа в формате TIME или DATETIME. Следующий код возвращает значение 01:37:38:

  ```
  SELECT TIMEDIFF('2000-01-01 01:02:03', '1999-12-31 23:24:25');
  ```

- UNIX_TIMESTAMP([*date*])

  Эта функция при вызове без необязательного аргумента *date* возвращает в формате беззнакового целого числа то количество секунд, которое прошло с нуля часов, нуля минут и нуля секунд универсального синхронного времени (UTC) 1 января 1970 года. Если функции передается параметр *date*, то возвращаемое значение содержит количество секунд, которое прошло с 1970 года по указанную дату *date*. Эта команда не будет возвращать одинаковое время для всех, поскольку переданные ей данные интерпретируются как местное время (заданное часовым поясом пользователя). Следующий код возвращает значение 946684800 (количество секунд, которое прошло до начала нового тысячелетия), а затем возвращает отметку времени TIMESTAMP, представляющую текущее время системы UNIX на момент запуска функции:

  ```
  SELECT UNIX_TIMESTAMP('2000-01-01');
  SELECT UNIX TIMESTAMP();
  ```

- FROM_UNIXTIME(*unix_timestamp* [, *format*])

  Возвращает параметр *unix_timestamp* либо в формате строки YYYY-MM-DD HH:MM:SS, либо в формате числа с плавающей точкой YYYYMMDDHHMMSS в зависимости от того, в каком контексте используется функция: строковом или числовом. Если задан необязательный параметр *format*, результат форматируется в соответствии со спецификаторами, показанными в табл. ПГ.2. Точное возвращаемое значение будет зависеть от местного времени пользователя. Следующий код возвращает строки 2000-01-01 00:00:00 и Saturday January 1st 2000 12:00 AM:

  ```
  SELECT FROM_UNIXTIME(946684800);
  SELECT FROM_UNIXTIME(946684800, '%W %M %D %Y %h:%i %p');
  ```

Д Селекторы, объекты и методы jQuery

В главе 21 заложены хорошие основы для использования библиотеки jQuery JavaScript. Чтобы помочь вам приступить к использованию jQuery и получить от нее наивысшую отдачу, здесь приводится полный перечень селекторов, объектов и методов, используемых в этой библиотеке. Для рассмотрения некоторых из них в данной книге места не нашлось, но вы уже готовы к их использованию, поскольку обладаете достаточным уровнем знаний для их правильного применения.

И все же имейте в виду, что иногда в библиотеку добавляются новые свойства, устраняются прежние недочеты, а некоторые функции могут быть не рекомендованы к дальнейшему применению или удалены. Быть в курсе новейших разработок, нерекомендуемых или удаленных функций (не упомянутых в данном приложении) и самых последних выпусков jQuery можно, изучая сайты http://jquery.com/ и http://api.jquery.com/.

Селекторы jQuery

`('*')`

Выбор всех элементов.

`('элемент')`

Выбор всех элементов с заданным именем тега.

`('#идентификатор')`

Выбор конкретного элемента с заданным атрибутом идентификатора (ID).

`('.класс')`

Выбор всех элементов с заданным классом.

`('селектор1, селектор2… селекторN')`

Выбор объединенных результатов всех указанных селекторов.

`('прародитель потомок')`

Выбор всех элементов, являющихся потомками указанного прародителя.

`('prev + next')`

Выбор всех следующих элементов, соответствующих селектору *next*, которые предшествуют непосредственно элементу *prev*, находящемуся с ними на одном уровне.

`('prev ~ siblings')`

Выбор всех элементов, находящихся на том же уровне, что и элемент *prev*, следующих непосредственно за ним, имеющих тот же самый родительский элемент и соответствующих фильтрующему селектору *siblings*.

`('parent > child')`

Выбор всех непосредственных дочерних элементов, указанных селектором *child*, относящихся к тем элементам, которые указаны селектором *parent*.

`[name]`

Выбор элементов с указанным именем атрибута *name* с любым значением этого атрибута.

`[name\='value']`

Выбор элементов с указанным именем атрибута *name* со значением этого атрибута, равным строке *value* или начинающимся с этой строки и продолжающимся знаком дефиса (-).

`[name*='value']`

Выбор элементов с указанным именем атрибута *name* со значением, содержащим заданную подстроку *value*.

`[name~='value']`

Выбор элементов с указанным именем атрибута *name* со значением, содержащим заданное слово *value*, отделенное от других слов пробелами.

`[name$='value']`

Выбор элементов с указанным именем атрибута *name* со значением, оканчивающимся на заданную строку *value*. Сравнение чувствительно к регистру букв.

`[name='value']`

Выбор элементов с указанным именем атрибута *name* со значением, точно соответствующим конкретно указанной строке *value*.

`[name!='value']`

Выбор элементов, у которых либо нет указанного имени атрибута *name*, либо он есть, но не с конкретным значением *value*.

`[name*='value']`

Выбор элементов с указанным именем атрибута *name* со значением, в точности начинающимся с заданной строки *value*.

`[name='value'][name2='value2']`

Соответствует элементам, которые соответствуют всем указанным фильтрам атрибутов.

`:animated`

Выбор всех элементов, находящихся в стадии применения к ним эффектов анимации на момент запуска селектора.

`:button`

Выбор всех элементов типа `button`.

`:checkbox`

Выбор всех элементов ввода типа `checkbox`.

`:checked`

Соответствует всем выбранным элементам или элементам с установленными флажками.

`:contains(text)`

Выбор всех элементов, содержащих указанный текст *text*.

`:disabled`

Выбор всех элементов, находящихся в неактивном состоянии.

`:empty`

Выбор всех элементов, не имеющих дочерних элементов (включая текстовые узлы).

`:enabled`

Выбор всех элементов, находящихся в активном состоянии.

`:eq(n)`

Выбор внутри соответствующего набора элементов элемента с индексом *n*.

`:even`

Выбор элементов с четными номерами позиций (с учетом индексации, начинающейся с нуля). См. также `:odd`.

`:file`

Выбор всех элементов ввода типа `file`.

`:first-child`

Выбор всех элементов, являющихся первыми дочерними элементами своего родителя.

`:first-of-type`

Выбор всех элементов, являющихся первыми среди одноуровневых элементов с одинаковыми названиями.

`:first`

Выбор первого соответствующего элемента.

`:focus`

Выбор элемента, если он в данный момент находится в фокусе.

`:gt(`*`index`*`)`

Выбор в соответствующем наборе всех элементов с индексом выше заданного индекса *`index`*.

`:has(`*`selector`*`)`

Выбор элемента, содержащего хотя бы один элемент, соответствующий указанному селектору *`selector`*.

`:header`

Выбор всех элементов, являющихся заголовками, например, `h1`, `h2`, `h3` и т. д.

`:hidden`

Выбор всех невидимых (скрытых) элементов.

`:image`

Выбор всех элементов типа `image`.

`:input`

Выбор всех элементов `input`, `textarea`, `select` и `button`.

`:lang(`*`language`*`)`

Выбор всех элементов указанного языка *`language`*.

`:last-child`

Выбор всех элементов, являющихся последними дочерними элементами своего родителя.

`:last-of-type`

Выбор всех элементов, являющихся последними среди одноуровневых элементов с одинаковыми названиями элемента.

`:last`

Выбор последнего соответствующего селектору элемента.

`:lt(`*`index`*`)`

Выбор в соответствующем наборе элементов тех из них, у которых индекс ниже заданного индекса *`index`*.

`:not(`*`selector`*`)`

Выбор всех элементов, не соответствующих указанному селектору *`selector`*.

`:nth-child(`*`n`*`)`

Выбор всех элементов, являющихся *n*-ми дочерними элементами своих родителей.

`:nth-last-child(`*`n`*`)`

Выбор всех элементов, являющихся *n*-ми дочерними элементами своих родителей, вычисляемых от последнего элемента к первому.

`:nth-last-of-type(`*`n`*`)`

Выбор всех элементов, являющихся *n*-ми дочерними элементами своих родителей по отношению к одноуровневым с ними элементам с тем же именем элемента при ведении отсчета от конца к началу.

`:nth-of-type(`*`n`*`)`

Выбор всех элементов, являющихся *n*-ми дочерними элементами своего родителя по отношению к одноуровневым элементам с одним и тем же названием.

`:odd`

Выбор элементов с нечетными номерами позиций (с учетом индексации, начинающейся с нуля). См. также `:even`.

`:only-child`

Выбор всех элементов, являющихся единственным дочерним элементом своих родителей.

`:only-of-type`

Выбор всех элементов, не имеющих одноуровневых элементов с таким же названием элемента.

`:parent`

Выбор всех элементов, у которых имеется хотя бы один дочерний узел (будь то элемент или текст).

`:password`

Выбор всех элементов ввода типа `password`.

`:radio`

Выбор всех элементов ввода типа `radio`.

`:reset`

Выбор всех элементов ввода типа `reset`.

`:root`

Выбор элемента, являющегося для документа корневым.

`:selected`

Выбор всех элементов ввода, которые были помечены на веб-странице в качестве выбранных.

`:submit`

Выбор всех элементов ввода типа `submit`.

`:target`

Выбор всех целевых элементов, обозначенных в URI документа идентификатором фрагмента.

`:text`

Выбор всех элементов ввода типа `text`.

`:visible`

Выбор всех видимых на экране элементов.

Объекты jQuery

`event.currentTarget`

Текущий элемент DOM в фазе всплытия события.

`event.data`

Необязательный объект данных, передаваемый методу события при привязке текущего исполняемого обработчика.

`event.delegateTarget`

Элемент, к которому был прикреплен вызванный в данный момент обработчик события jQuery.

`event.metaKey`

Показывает, была ли нажата клавиша META в момент выдачи события.

`event.namespace`

Пространство имен, указанное при инициировании события.

`event.pageX`

Позиция указателя мыши относительно левого края документа.

`event.pageY`

Позиция указателя мыши относительно верхнего края документа.

`event.relatedTarget`

Другой DOM-элемент, вовлеченный в событие, если таковой имеется.

`event.result`

Последнее значение, возвращенное обработчиком события, запущенным данным событием, если только оно не было равно `undefined`.

`event.target`

DOM-элемент, инициировавший событие.

`event.timeStamp`

Разница в миллисекундах между временем создания события браузером и 1 января 1970 года.

`event.type`

Дескриптор события.

`event.which`

Для событий клавиатуры или мыши это свойство показывает конкретную нажатую клавишу или кнопку.

`.jquery`

Строка, содержащая номер версии jQuery.

`jQuery.cssHooks`

Непосредственный перехват внутри jQuery для переопределения способа извлечения или установки конкретных свойств CSS, приведения к общему виду имен свойств CSS или создания специально определяемых свойств.

`jQuery.cssNumber`

Содержит все свойства CSS, которые могут применяться без единицы измерения. Метод `css` использует этот объект, чтобы увидеть, может ли он добавить `px` к значениям без указания единицы измерения.

`jQuery.ready`

Promise-подобный (или так называемый тогдашний) объект, разрешение которого выполняется по готовности документа.

`jQuery.fx.off`

Глобальное отключение всех эффектов анимации.

`length`

Количество элементов объекта jQuery.

Методы jQuery

`$`

Возвращение коллекции соответствующих элементов, либо найденных в DOM на основе переданного аргумента (аргументов), либо созданных путем передачи строки HTML.

`add`

Добавление элементов к набору соответствующих элементов.

`addBack`

Добавление предыдущего набора элементов в стек текущего набора с возможностью фильтрации с помощью селектора.

`addClass`

Добавление указанного класса (классов) к каждому набору соответствующих элементов.

`after`

Вставка содержимого, указанного параметром, после каждого элемента в наборе соответствующих элементов.

`ajaxComplete`

Регистрация обработчика, вызываемого при завершении Ajax-запросов.

`ajaxError`

Регистрация обработчика, вызываемого при завершении Ajax-запросов ошибкой.

`ajaxSend`

Прикрепление функции, выполняемой перед отправкой Ajax-запроса.

`ajaxStart`

Регистрация обработчика, вызываемого при запуске первого Ajax-запроса.

`ajaxStop`

Регистрация обработчика, вызываемого при завершении всех Ajax-запросов.

`ajaxSuccess`

Прикрепление функции, выполняемой при успешном завершении Ajax-запроса.

`animate`

Выполнение эффекта анимации, специально определенного в отношении набора CSS-свойств.

`append`

Вставка содержимого, указанного параметром, в конец каждого элемента в наборе соответствующих элементов.

`appendTo`

Добавление каждого элемента набора соответствующих элементов к концу целевого элемента.

`attr`

Получение значения атрибута первого элемента в наборе соответствующих элементов или установка одного или нескольких атрибутов для каждого соответствующего элемента.

`before`

Вставка содержимого, указанного параметром, перед каждым элементом в наборе соответствующих элементов.

`blur`

Привязка обработчика событий к событию JavaScript `blur` или инициирование этого события в отношении элемента.

`callbacks.add`

Добавление функции обратного вызова или коллекции таких функций к списку функций обратного вызова.

`callbacks.disable`

Отключение списка функций обратного вызова от их дальнешего выполнения.

`callbacks.disabled`

Определение отключенного состояния списка функций отбратного вызова.

`callbacks.empty`

Удаление всех функций обратного вызова из списка.

`callbacks.fire`

Вызов всех функций обратного вызова с заданными аргументами.

`callbacks.fired`

Определение факта хотя бы однократного вызова функций обратного вызова.

`callbacks.fireWith`

Вызов всех имеющихся в списке функций обратного вызова с заданным контекстом и аргументами.

`callbacks.has`

Определение факта прикрепления к списку каких-либо функций обратного вызова или (если функция обратного вызова предоставлена в качестве аргумента) факта наличия предоставленной функции в списке.

`callbacks.lock`

Блокировка списка функций обратного вызова в его текущем состоянии.

`callbacks.locked`

Определение факта заблокированности списка функций обратного вызова.

`callbacks.remove`

Удаление функции обратного вызова или коллекции таких функций из их списка.

`change`

Привязка обработчика событий к событию JavaScript `change` или инициирование этого события в отношении элемента (в случае вызова без аргумента).

`children`

Получение дочернего элемента для каждого элемента в наборе соответствующих элементов с возможностью фильтрации с помощью селектора.

`clearQueue`

Удаление из очереди всех функций, которые еще не были запущены.

`click`

Привязка обработчика событий к событию JavaScript `click` или инициирование этого события в отношении элемента (в случае вызова без аргумента).

`clone`

Создание глубокой копии набора соответствующих элементов.

`closest`

Получение для каждого элемента в наборе первого элемента, соответствующего селектору, путем тестирования самого элемента и обхода вверх его прародителей в DOM-дереве.

`contents`

Получение дочернего элемента для каждого элемента в наборе соответствующих элементов, включая текстовые узлы и узлы комментариев.

`css`

Получение значения стилевого свойства для первого элемента в наборе соответствующих элементов или (в случае вызова без дополнительного аргумента) установка одного или нескольких CSS-свойств для каждого соответствующего элемента.

`data`

Сохранение произвольных данных, связанных с соответствующими элементами, или возвращение значения в поименованном хранилище данных для первого элемента в наборе соответствующих элементов.

`dblclick`

Привязка обработчика событий к событию JavaScript `dblclick` или инициирование этого события в отношении элемента.

`deferred.always`

Добавление обработчиков, вызываемых, когда в объекте `Deferred` хранятся сведения либо об успешном выполнении задачи, либо о возникновении ошибки при ее выполнении.

`deferred.catch`

Добавление обработчиков для вызова в случае, если объект `Deferred` отклонен.

`deferred.done`

Добавление обработчиков, вызываемых, когда в объекте `Deferred` хранятся сведения об успешном выполнении задачи.

`deferred.fail`

Добавление обработчиков, вызываемых, когда в объекте `Deferred` хранятся сведения о возникновении ошибки при выполнении задачи.

`deferred.notify`

Вызов функции `progressCallbacks` в отношении объекта `Deferred` с заданными аргументами.

`deferred.notifyWith`

Вызов функции `progressCallbacks` в отношении объекта `Deferred` с заданными контекстом и аргументами.

`deferred.progress`

Добавление обработчиков, вызываемых, когда объект `Deferred` генерирует уведомление о ходе выполнения.

`deferred.promise`

Возвращение объекта `Promise`, являющегося заместителем объекта `Deferred`.

`deferred.reject`

Перевод объекта `Deferred` в состояние невыполнения задачи и вызов любой функции `failCallbacks` с заданными аргументами.

`deferred.rejectWith`

Перевод объекта `Deferred` в состояние невыполнения задачи и вызов любой функции `failCallbacks` с заданными контекстом и аргументами.

`deferred.resolve`

Перевод объекта `Deferred` в состояние выполнения задачи и вызов любой функции `doneCallbacks` с заданными аргументами.

`deferred.resolveWith`

Перевод объекта `Deferred` в состояние выполнения задачи и вызов любой функции `doneCallbacks` с заданными контекстом и аргументами.

`deferred.state`

Определение текущего состояния объекта `Deferred`.

`deferred.then`

Добавление обработчиков, вызываемых при переводе объекта `Deferred` в состояние выполнения задачи, невыполнения задачи или нахождении его в состоянии генерирования уведомление о ходе выполнения задачи.

`delay`

Установка таймера на задержку выполнения следующей функции из очереди функций.

`dequeue`

Выполнение следующей функции из очереди для соответствующих элементов.

`detach`

Удаление набора соответствующих элементов из DOM.

`each`

Обход элементов объекта jQuery с выполнением функции в отношении каждого соответствующего элемента.

`empty`

Удаление из DOM всех дочерних узлов, относящихся к набору соответствующих элементов.

`end`

Завершение последней операции фильтрации в текущей цепочке и возвращение соответствующих элементов к их предыдущему состоянию.

`eq`

Сокращение набора соответствующих элементов до одного с указанным индексом.

`event.isDefaultPrevented`

Возвращение истинного значения, если в отношении данного объекта события когда-либо вызывался метод `preventDefault`.

`event.isImmediatePropagationStopped`

Возвращение истинного значения, если в отношении данного объекта события когда-либо вызывался метод `stopImmediatePropagation`.

`event.isPropagationStopped`

Возвращение истинного значения, если в отношении данного объекта события когда-либо вызывался метод `stopPropagation`.

`event.preventDefault`

При вызове данного метода действие по умолчанию для данного события не запускается.

`event.stopImmediatePropagation`

Воспрепятствование выполнению оставшихся обработчиков и всплытию события вверх по DOM-дереву.

`event.stopPropagation`

Воспрепятствование всплытию события по DOM-дереву и уведомлению о событии любых родительских обработчиков.

`fadeIn`

Постепенное проявление соответствующих элементов на экране.

`fadeOut`

Постепенное растворение соответствующих элементов на экране.

`fadeTo`

Настройка прозрачности соответствующих элементов.

`fadeToggle`

Вывод на экран или скрытие соответствующих элементов путем анимации их прозрачности.

`filter`

Сокращение набора соответствующих элементов до тех, которые соответствуют селектору или проходят функциональный тест.

`find`

Получение потомков каждого элемента в текущем наборе соответствующих элементов, отфильтрованных селектором, объектом jQuery или элементом.

`finish`

Остановка текущей запущенной анимации, удаление всех эффектов анимации, находящихся в очереди, и завершение всех эффектов анимации для соответствующих элементов.

`first`

Сокращение набора соответствующих элементов до первого элемента в наборе.

`focus`

Привязка обработчика событий к событию JavaScript focus или инициирование этого события в отношении элемента (при вызове без аргумента).

`focusin`

Привязка обработчика событий к событию JavaScript `focusin`.

`focusout`

Привязка обработчика событий к событию JavaScript `focusout`.

`get`

Извлечение DOM-элементов, соответствующих объекту jQuery

`has`

Сокращение набора соответствующих элементов до тех элементов, у которых имеется потомок, соответствующий селектору или DOM-элементу.

`hasClass`

Определение факта присваивания любому из соответствующих элементов заданного класса.

`height`

Получение текущей вычисленной высоты первого элемента в наборе соответствующих элементов или установка высоты каждого соответствующего элемента (при вызове с еще одним аргументом).

`hide`

Скрытие соответствующих элементов.

`hover`

Привязка одного или двух обработчиков к соответствующим элементам для их выполнения при входе указателя мыши в границы элемента или при его выходе за эти границы.

`html`

Получение HTML-содержимого первого элемента в наборе соответствующих элементов или установка HTML-содержимого для каждого соответствующего элемента (при вызове с еще одним аргументом).

`index`

Поиск заданного элемента среди соответствующих элементов.

`innerHeight`

Получение текущей вычисленной высоты первого элемента в наборе соответствующих элементов (включая поля, но не границы) или (при вызове с еще одним аргументом) установка внутренней высоты каждого соответствующего элемента.

`innerWidth`

Получение текущей вычисленной ширины (включая поля, но не границы) первого элемента в наборе соответствующих элементов или (при вызове с еще одним аргументом) установка внутренней ширины каждого соответствующего элемента.

`insertAfter`

Вставка каждого элемента в наборе соответствующих элементов после целевого элемента.

`insertBefore`

Вставка каждого элемента в наборе соответствующих элементов перед целевым элементом.

`is`

Проверка текущего соответствующего селектору, элементу или jQuery-объекту набора элементов и возвращение истинного значения, если хотя бы один из этих элементов соответствует заданным аргументам.

`jQuery`

Возвращение коллекции соответствующих элементов, либо найденных в DOM на основе переданных аргументов (или аргумента), либо созданных путем передачи строки HTML.

`jQuery.ajax`

Выполнение асинхронного HTTP-запроса (Ajax-запроса).

`jQuery.ajaxPrefilter`

Обработка специальных Ajax-настроек или изменение существующих настроек перед отправкой каждого запроса и их обработкой методом `$.ajax`.

`jQuery.ajaxSetup`

Установка значений по умолчанию для будущих Ajax-запросов. Использование данного метода не рекомендуется.

`jQuery.ajaxTransport`

Создание объекта, занимающегося фактической передачей Ajax-данных.

`jQuery.Callbacks`

Многоцелевой списочный объект функций обратного вызова, предоставляющий эффективный способ управления списками функций обратного вызова.

`jQuery.contains`

Проверка того, является ли DOM-элемент потомком другого DOM-элемента.

`jQuery.data`

Сохранение произвольных данных, связанных с конкретным элементом и/или возвращение установленного значения.

`jQuery.Deferred`

Функция-конструктор, возвращающая способный встраиваться в цепочку вспомогательный объект с методами регистрации нескольких функций обратного вызова в очередях функций обратного вызова и менять состояние успеха или неудачи любой синхронной или асинхронной функции.

`jQuery.dequeue`

Выполнение следующей функции в очереди для соответствующего элемента.

`jQuery.each`

Универсальная функция-итератор, которая может использоваться для беспрепятственной итерации как через объекты, так и через массивы. Массивы и подобные массивам объекты, имеющие свойство длины (такие как объект аргументов функций), проходят поэлементный перебор с использованием числового индекса от `0` до длины `-1`. Другие объекты подвергаются поэлементному перебору посредством их поименованных свойств.

`jQuery.error`

Передача методу строки и выдача исключения, содержащего эту строку.

`jQuery.escapeSelector`

Обезвреживание символов, имеющих в селекторе CSS специальное значение.

`jQuery.extend`

Объединение содержимого двух или более объектов в первом объекте.

`jQuery.fn.extend`

Объединение содержимого объекта в jQuery-прототип для предоставления новых методов экземпляра jQuery.

`jQuery.get`

Загрузка данных с сервера с использованием Get-запроса HTTP.

`jQuery.getJSON`

Загрузка данных, закодированных в формате JSON, с сервера с использованием Get-запроса HTTP.

`jQuery.getScript`

Загрузка файла JavaScript с сервера с использованием Get-запроса HTTP с последующим выполнением этого файла.

`jQuery.globalEval`

Глобальное выполнение кода JavaScript.

`jQuery.grep`

Поиск элементов массива, удовлетворяющих функции фильтрации. На исходный массив не влияет.

`jQuery.hasData`

Определение наличия у элемента jQuery связанных с ним данных.

`jQuery.holdReady`

Удерживание или освобождение выполнения jQuery-события `ready`.

`jQuery.htmlPrefilter`

Изменение и фильтрация строк HTML, переданных посредством jQuery-методов работы со строками.

`jQuery.inArray`

Поиск указанного значения в массиве и возвращение его индекса (или `-1`, если значение не найдено).

`jQuery.isArray`

Определение, является ли аргумент массивом.

`jQuery.isEmptyObject`

Проверка, что объект пуст (не содержит исчисляемых свойств).

`jQuery.isFunction`

Определение, является ли переданный аргумент объектом функции JavaScript.

`jQuery.isNumeric`

Определение, является ли аргумент числом.

`jQuery.isPlainObject`

Проверка того факта, что объект является простым объектом (созданным с помощью `{}` или `new Object`).

`jQuery.isWindow`

Определение, является ли аргумент окном.

`jQuery.isXMLDoc`

Определение, находится ли DOM-узел внутри XML-документа (или является ли он XML-документом).

`jQuery.makeArray`

Преобразование похожего на массив объекта в настоящий массив JavaScript.

`jQuery.map`

Преобразование всех элементов массива или объекта в новый массив элементов.

`jQuery.merge`

Слияние содержимого двух массивов в первый массив.

`jQuery.noConflict`

Передача управления jQuery от переменной с именем `$` переменной с другим именем.

`jQuery.noop`

Пустая функция.

`jQuery.now`

Возвращение числа, представляющего текущее время.

`jQuery.param`

Создание сериализованного представления массива или объекта, пригодного к использованию в строке запроса URL-адреса или в Ajax-запросе.

`jQuery.parseHTML`

Синтаксический разбор строки с целью получения массива DOM-узлов.

`jQuery.parseJSON`

Передача методу правильно оформленной JSON-строки и возвращение им результирующего объекта JavaScript.

`jQuery.parseXML`

Синтаксический разбор строки с целью получения XML-документа.

`jQuery.post`

Загрузка данных с сервера с использованием POST-запроса HTTP.

`jQuery.proxy`

Получение функции и возвращение новой функции, которая всегда будет иметь конкретный контекст.

`jQuery.queue`

Демонстрация или (при вызове с еще одним аргументом) манипуляция очередью функций, выполняемых в отношении соответствующего элемента.

`jQuery.readyException`

Обработка ошибок, возникающих синхронно в функциях, заключенных в jQuery.

`jQuery.removeData`

Удаление ранее сохраненного фрагмента данных.

`jQuery.speed`

Создание объекта, содержащего набор свойств, готовых к использованию в определении пользовательских анимаций.

`jQuery.trim`

Удаление пробельных символов в начале и конце строки.

`jQuery.type`

Определение внутреннего JavaScript-класса объекта.

`jQuery.uniqueSort`

Сортировка массива DOM-элементов на месте с удалением дубликатов. Работает только в отношении массивов DOM-элементов, исключая строковые или числовые массивы.

`jQuery.when`

Предоставление способа выполнения функций обратного вызова на основе от нуля и более объектов, обычно объектов `Deferred`, представляющих асинхронные события.

`keydown`

Привязка обработчика событий к событию JavaScript `keydown` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`keypress`

Привязка обработчика событий к событию JavaScript `keypress` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`keyup`

Привязка обработчика событий к событию JavaScript `keyup` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`last`

Сокращение набора соответствующих элементов до последнего элемента в наборе.

`load`

Загрузка данных с сервера и помещение возвращенного HTML в соответствующий элемент.

`map`

Пропуск каждого элемента в соответствующем наборе через функцию с созданием нового объекта jQuery, содержащего возвращенные значения.

`mousedown`

Привязка обработчика событий к событию JavaScript `mousedown` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mouseenter`

Привязка обработчика события, инициируемого при входе указателя мыши в область элемента, или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mouseleave`

Привязка обработчика события, инициируемого при выходе указателя мыши из области элемента, или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mousemove`

Привязка обработчика событий к событию JavaScript `mousemove` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mouseout`

Привязка обработчика событий к событию JavaScript `mouseout` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mouseover`

Привязка обработчика событий к событию JavaScript `mouseover` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`mouseup`

Привязка обработчика событий к событию JavaScript `mouseup` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`next`

Получение одноуровневых элементов, непосредственно следующих за каждым элементом набора соответствующих элементов. Если предоставлен селектор, следующий одноуровневый элемент извлекается только при его соответствии селектору.

`nextAll`

Получение всех следующих одноуровневых элементов каждого элемента из набора соответствующих элементов с возможностью фильтрации с помощью селектора.

`nextUntil`

Получение всех следующих одноуровневых элементов вплоть до каждого элемента, соответствующего селектору, DOM-узлу или переданному объекту jQuery, исключая сам этот элемент.

`not`

Удаление элементов из набора соответствующих элементов.

`off`

Удаление обработчика события.

`offset`

Получение текущих координат первого элемента или (при вызове с аргументом) установка координат каждого элемента в наборе соответствующих элементов относительно документа.

`offsetParent`

Получение ближайшего прародительского элемента с заданным типом позиционирования.

`on`

Прикрепление функции обработчика события для одного или нескольких событий для выбранных элементов.

`one`

Прикрепление обработчика к событию для элементов. Обработчик выполняется не более одного раза для каждого элемента каждого типа событий.

`outerHeight`

Получение текущей вычисленной высоты первого элемента в наборе соответствующих элементов, включая поля и границы, а также, возможно, отступы.

`outerWidth`

Получение текущей вычисленной ширины первого элемента в наборе соответствующих элементов, включая поля и границы.

`parent`

Получение родительского элемента для каждого элемента в текущем наборе соответствующих элементов с возможностью фильтрации с помощью селектора.

`parents`

Получение прародительских элементов для каждого элемента в текущем наборе соответствующих элементов с возможностью фильтрации с помощью селектора.

`parentsUntil`

Получение прародительских элементов для каждого элемента в текущем наборе соответствующих элементов вплоть до элемента, соответствующего селектору, DOM-узлу или объекту jQuery, исключая сам этот элемент.

`position`

Получение текущих координат первого элемента в наборе соответствующих элементов относительно смещения родительского элемента.

`prepend`

Вставка содержимого, указанного параметром, в начало каждого элемента в наборе соответствующих элементов.

`prependTo`

Вставка каждого элемента в наборе соответствующих элементов в начало целевого элемента.

`prev`

Получение всех непосредственно предшествующих одноуровневых элементов каждого элемента из набора соответствующих элементов с возможностью фильтрации с помощью селектора.

`prevAll`

Получение всех предшествующих одноуровневых элементов каждого элемента из набора соответствующих элементов с возможностью фильтрации с помощью селектора.

`prevUntil`

Получение всех предшествующих одноуровневых элементов до элемента, соответствующего селектору, DOM-узлу или jQuery-объекту, за исключением самого этого элемента.

`promise`

Возвращение объекта `Promise` для наблюдения за тем, завершены ли все действия конкретного типа, привязанные к коллекции, независимо от того, выстроены они в очередь или нет.

`prop`

Получение значения свойства первого элемента в наборе соответствующих элементов или (при вызове с еще одним аргументом) установка одного или нескольких свойств для каждого соответствующего элемента.

`pushStack`

Добавление коллекции DOM-элементов в стек jQuery.

`queue`

Показ очереди функций, выполняемых в отношении соответствующих элементов, или (при вызове с еще одним аргументом) манипулирование этой очередью.

`ready`

Определение функции, выполняемой, когда DOM полностью загружена.

`remove`

Удаление набора соответствующих элементов из DOM.

`removeAttr`

Удаление атрибута из каждого элемента в наборе соответствующих элементов.

`removeClass`

Удаление одного класса, нескольких классов или всех классов из каждого элемента в наборе соответствующих элементов.

`removeData`

Удаление ранее сохраненного фрагмента данных.

`removeProp`

Удаление свойства для набора соответствующих элементов.

`replaceAll`

Замена каждого целевого элемента набором соответствующих элементов.

`replaceWith`

Замена каждого элемента из набора соответствующих элементов с предоставлением нового содержимого и возвращением набора удаленных элементов.

`resize`

Привязка обработчика событий к событию JavaScript `resize` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`scroll`

Привязка обработчика событий к событию JavaScript `scroll` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`scrollLeft`

Получение текущей горизонтальной позиции полосы прокрутки для первого элемента в наборе соответствующих элементов или (при вызове с аргументом) установка горизонтальной позиции полосы прокрутки для каждого соответствующего элемента.

`scrollTop`

Получение текущей вертикальной позиции полосы прокрутки для первого элемента в наборе соответствующих элементов или (при вызове с аргументом) установка вертикальной позиции полосы прокрутки для каждого соответствующего элемента.

`select`

Привязка обработчика событий к событию JavaScript `select` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`serialize`

Преобразование набора элементов формы в строку для последующей отправки.

`serializeArray`

Преобразование набора элементов формы в массив имен и значений.

`show`

Отображение соответствующих элементов.

`siblings`

Получение одноуровневых элементов каждого элемента в наборе соответствующих элементов с возможностью фильтрации с помощью селектора.

`slice`

Сокращение набора соответствующих элементов до поднабора в указанном диапазоне индексов.

`slideDown`

Отображение соответствующих элементов путем их выскальзывания из-под границ.

`slideToggle`

Отображение или скрытие соответствующих элементов путем их выскальзывания за границы или из-под этих границ.

`slideUp`

Скрытие соответствующих элементов путем их выскальзывания за свои границы.

`stop`

Остановка текущей запущенной анимации в отношении соответствующих элементов.

`submit`

Привязка обработчика событий к событию JavaScript `submit` или (при вызове без аргумента) инициирование этого события в отношении элемента.

`text`

Получение объединенного текстового содержимого каждого элемента в наборе соответствующих элементов или (при вызове с аргументом) установка текстового содержимого соответствующих элементов.

`toArray`

Извлечение всех элементов, содержащихся в наборе jQuery в виде массива.

`toggle`

Отображение или скрытие соответствующих элементов.

`toggleClass`

Добавление или удаление одного или нескольких классов из каждого элемента в наборе соответствующих элементов в зависимости либо от присутствия класса, либо от значения аргумента состояния.

`trigger`

Выполнение всех обработчиков и правил поведения, прикрепленных к соответствующим элементам для заданного типа события.

`triggerHandler`

Выполнение всех обработчиков, прикрепленных к элементу для события.

`unwrap`

Удаление родительских элементов набора соответствующих элементов из DOM с оставлением соответствующих элементов на их местах.

`val`

Получение текущего значения первого элемента в наборе соответствующих элементов или (при вызове с аргументом) установка значения каждого соответствующего элемента.

`width`

Получение текущей вычисленной ширины первого элемента в наборе соответствующих элементов или (при вызове с аргументом) установка ширины для каждого соответствующего элемента.

`wrap`

Заключение каждого элемента в наборе соответствующих элементов в HTML-структуру.

`wrapAll`

Заключение всех элементов в наборе соответствующих элементов в HTML-структуру.

`wrapInner`

Заключение содержимого каждого элемента в наборе соответствующих элементов в HTML-структуру.

Робин Никсон

Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5

5-е издание

Перевел с английского *Н. Вильчинский*

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*О. Сивченко*
Ведущий редактор	*Н. Гринчик*
Художественный редактор	*С. Заматевская*
Корректор	*Т. Радецкая*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 10.2018. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 17.10.18. Формат 70×100/16. Бумага офсетная. Усл. п. л. 65,790. Тираж 1000. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87