Дженнифер Роббинс

Независимо от того, новичок вы или опытный веб-дизайнер, эта книга научит вас основам современного веб-дизайна

HTML5, CSS3 и JavaScript

ИСЧЕРПЫВАЮЩЕЕ РУКОВОДСТВО

ЭКСМО

Дженнифер Роббинс

HTML5, CSS3 и JavaScript

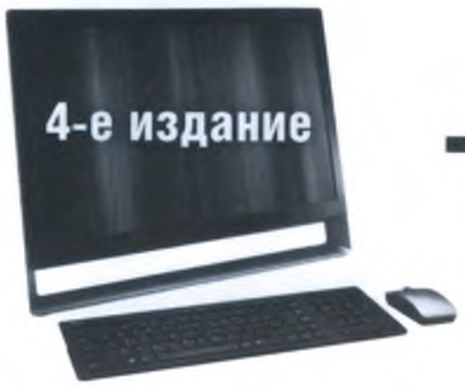
ИСЧЕРПЫВАЮЩЕЕ РУКОВОДСТВО

ЭКСМО
2014

Дженнифер Роббинс

HTML5, CSS3 и JavaScript

ИСЧЕРПЫВАЮЩЕЕ РУКОВОДСТВО



+

ЭКСМО
2014

УДК 004.42
ББК 32.973.26
Р 58

Jennifer Niederst Robbins
Learning Web Design, 4th Edition
Authorized Russian translation of the English edition of Learning Web Design, 4th Edition (ISBN 9781449319274)

Роббинс Дж.

Р 58 HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Дженнифер Роббинс ; [пер. с англ. М. А. Райтман]. — 4-е издание. — М. : Эксмо, 2014. — 528 с. + DVD. — (Мировой компьютерный бестселлер).

В этой книге вы найдете все, что необходимо знать для создания отличных веб-сайтов. Начав с изучения принципов функционирования Интернета и веб-страниц, к концу книги вы освоите приемы создания сложных сайтов, включая таблицы стилей CSS и графические файлы, и научитесь размещать страницы во Всемирной паутине. Книга включает упражнения, с помощью которых вы освоите разнообразные техники работы с современными веб-стандартами (включая HTML5 и CSS3).

На диске — описанные в книге программы и примеры.

УДК 004.42
ББК 32.973.26

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Роббинс Дженнифер

HTML5, CSS3 и JavaScript
ИСЧЕРПЫВАЮЩЕЕ РУКОВОДСТВО
(орыс тілінде)

Директор редакции *Е. Капьёв*
Ответственный редактор *В. Обручев*
Художественный редактор *Г. Федотов*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86, 8 (495) 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86, 8 (495) 956-39-21
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Қазақстан Республикасында дистрибьютор және өнім бойынша
арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 2 51 59 89,90,91,92, факс: 8 (727) 251 58 12 вн. 107; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить по адресу: http://eksmo.ru/certification/

Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Подписано в печать 21.01.2014.
Формат 84x108$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 55,44.
Тираж 2000 экз. Заказ 561.

Отпечатано с готовых файлов заказчика
в ОАО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, г. Ульяновск, ул. Гончарова, 14

ISBN 978-5-699-67603-3

ISBN 978-5-699-67603-3

ОГЛАВЛЕНИЕ

ЧАСТЬ II РАЗМЕТКА HTML ДЛЯ СТРУКТУРИЗАЦИИ

СЛЕДУЮЩИЕ ГЛАВЫ И ПРИЛОЖЕНИЯ НАХОДЯТСЯ НА ДИСКЕ, ПРИЛАГАЕМОМ К КНИГЕ

ЧАСТЬ V JAVASCRIPT ДЛЯ ПОВЕДЕНИЯ

ПРЕДИСЛОВИЕ

Веб-дизайн потрясающе изменчив. Как только начало казаться, что ситуация вокруг принятия веб-стандартов создателями браузеров и сообществом веб-разработчиков успокоилась, появился «мобильный Интернет», вновь перевернувший все. С выпуском смартфонов и планшетов Всемирная паутина пробивает себе дорогу на маленькие экраны портативных устройств, где никогда раньше не использовалась. В результате веб-дизайнеры и программисты столкнулись с серьезными трудностями, пытаясь найти способы, как сделать взаимодействие с сайтами приятным для пользователя, независимо от того, с каких устройств к ним получен доступ.

На момент написания этой книги многие из проблем, например, как визуализировать изображение на нужном устройстве, еще находились в стадии обсуждения. Это невероятно активный период для веб-дизайна, полный экспериментов и совместной работы. Он чем-то напоминает мне дни становления Всемирной паутины в 1993 году, когда я только начинала карьеру веб-дизайнера. Столько всего еще предстоит понять! Так много возможностей! И, если честно, сейчас нелегко писать книгу обо всех этих меняющихся технологиях. Поэтому я сделала все возможное, чтобы выделить развивающиеся области и предоставить ссылки на онлайн-ресурсы, чтобы держать вас в курсе.

ПРИМЕРЫ НА ДИСКЕ

Все примеры для выполнения упражнений из книги вы найдете на диске, прилагающемся к книге.

Кроме того, стали доступны два новых стандарта — HTML5 (пятая основная версия языка гипертекстовой разметки) и CSS3 (каскадные таблицы стилей, уровень 3). В разделе этой книги, посвященном HTML, отражен современный стандарт HTML5. Рассмотрены уже готовые к применению разделы разрабатывающегося стандарта CSS3 и присутствует глава, описывающая, как добиться интерактивности с помощью переходов и преобразований. Сегодня инструменты позволяют сделать гораздо больше и эффективнее, чем всего несколько лет назад.

Наконец, поскольку JavaScript — важная часть процесса веб-разработки, в книгу включены две главы, в которых представлен этот язык и несколько способов его использования. Я не специалист по JavaScript, но, к счастью, нашла эксперта в этой области.

Главы, посвященные JavaScript, написал Мэтт «Wilto» Маркус, дизайнер и разработчик в компании Filament Group, член команды jQuery Mobile и технический редактор издания A List Apart.

Эта книга рассматривает специфические проблемы и вопросы, возникающие у новичков, опытных графических дизайнеров, програм-

мистов, офисных служащих и всех, кто хочет научиться веб-дизайну. Я постаралась изложить здесь опыт моего преподавания на курсах для начинающих, добавила упражнения и тесты, чтобы вы могли получить практический опыт и проверить свои успехи.

Читаете ли вы эту книгу отдельно или в качестве дополнения к курсу веб-дизайна, я надеюсь, она поможет вам начать работу и получить удовольствие от процесса.

Структура книги

Книга состоит из пяти разделов, в каждом из которых рассматриваются важные аспекты веб-разработки.

Часть I. Начало работы

В ней закладываются основы всех рассматриваемых в книге тем. Она начинается со значимых общих сведений о среде веб-разработки: чем вы можете заниматься, какие технологии изучить и какие инструменты использовать. Вы сразу начнете работать с HTML и CSS и узнаете в общих чертах, как функционируют веб-страницы. Я также познакомлю вас с некоторыми важными понятиями, которые помогут вам начать думать о дизайне, как современный веб-дизайнер.

Часть II: Разметка HTML для структуризации

Здесь в подробностях раскрывается работа всех элементов и атрибутов, доступных для создания семантической структуры контента, в том числе и новых элементов, введенных в HTML5. Я рассмотрю разметку текста, ссылок, изображений, таблиц и форм. Часть II завершается всесторонним обсуждением языка HTML5 и его отличий от предыдущих стандартов.

Часть III: Правила CSS для представления

В этой части вы перейдете от изучения основ использования каскадных таблиц стилей для изменения внешнего вида текста к созданию многоколоночных макетов и добавлению на страницу синхронизированной по времени анимации и интерактивности. Здесь также рассматриваются распространенные методы использования CSS, в том числе описывается, как создать страницу с помощью адаптивного веб-дизайна.

Часть IV: Создание графических изображений для Всемирной паутины

Здесь описаны различные форматы файлов, подходящие для использования во Всемирной паутине, и способы их оптимизации с целью максимально уменьшить размер файла.

Часть V: JavaScript для поведения

Мэтт Маркус начинает свою часть с краткого описания синтаксиса языка JavaScript, чтобы вы могли отличить переменную от функции. Вы также познакомитесь с некоторыми способами использования

Специальные обозначения текста в этой книге

В этой книге используются следующие обозначения текста:

Курсив

Используется для выделения терминов, имен файлов и каталогов, а также для привлечения внимания.

Полужирный

Используется для выделения адресов веб-сайтов и электронной почты.

`Листинг`

Моноширинным шрифтом выделены примеры кода.

`Полужирный листинг`

Моноширинный полужирный шрифт используется для привлечения внимания в примерах кода.

`Курсивный листинг`

Моноширинный шрифт с курсивом используется для выделения заполнителей для значений атрибутов и свойств таблиц стилей.

языка JavaScript, в том числе сценариев *объектной модели документа* (*Document Object Model, DOM*), а также существующих инструментов JavaScript таких, как полизаполнители и библиотеки, которые позволят вам быстро приступить к работе с JavaScript, даже если вы еще не готовы писать собственный код с нуля.

Благодарности

Я хочу поблагодарить моего редактора, Саймона Сент-Лорена (Simon St. Laurent), с которым мне посчастливилось работать над совместными проектами, и я надеюсь на дальнейшее сотрудничество. Спасибо моему соавтору, Мэтту Маркусу (Mat Marquis) (**matmarquis.com**), за то, что он смог показать интересные стороны языка JavaScript и не терял хорошего настроения при работе со мной.

Множество умных и замечательных людей поддерживали меня при написании этой книги. Я хочу поблагодарить моих основных технических рецензентов, Аарона Густафсона (Aaron Gustafson) (**easy-designs.net**), Джоэль Марш (Joel Marsh) (**thehipperelement.com**) и Мэтта Мензера (Matt Menzer), за то, что уделили так много времени проверке глав этой книги.

Также спасибо следующим людям за их точные рецензии: Энтони Калзадилле (Anthony Calzadilla), Дэнни Чэпману (Danny Chapman), Мэтту Хофи (Matt Haughey), Джералду Льюису (Gerald Lewis), Джейсону Паменталу (Jason Pamental) и Стефани Ригер (Stephanie Rieger).

К счастью, я знакома со множеством экспертов в данной области, чьи книги, статьи, презентации, слайды и личное общение подпитывали мою работу. Я не смогла бы закончить ее без помощи этих гениев: Дэна Седерхольма (Dan Cederholm), Джоша Кларка (Josh Clark), Энди Клэрка (Andy Clarke), Криса Койера (Chris Coyier), Брэда Фроста (Brad Frost), Лизы Гарднер (Lyza Gardner), Джейсона Гризби (Jason Grigsby), Стефана Хэя (Stephen Hay), Скотта Джела (Scott Jehl), Скотта Дженсона (Scott Jenson), Тима Кадлека (Tim Kadlec), Джереми Кейта (Jeremy Keith), Сандерса Кляйнфильда (Sanders Kleinfeld), Питера-Пола Коха (Peter-Paul Koch), Брюса Лоусона (Bruce Lawson), Итана Маркотта (Ethan Marcotte), Эрика Мейера (Eric Meyer), Карен МакГрейн (Karen McGrane), Шелли Пауэрс (Shelley Powers), Брайана Ригера (Bryan Rieger), Стефании Ригер (Stephanie Rieger), Реми Шарпа (Remy Sharp), Люка Вроблевски (Luke Wroblewski) и Джеффри Зелдмана (Jeffrey Zeldman).

Книга пишется всем миром, поэтому я хочу поблагодарить за их вклад Мелани Ярбро (Melanie Yarbrough) (выпускающий редактор и корректор), Женевьеву Д'Энтремон (Genevieve d'Entremont) (литературный редактор), Ребекку Демарест (Rebecca Demarest) (создание рисунков), компанию Ньюген (Newgen) (создание макета страницы), Эллен Траумен Зейг (Ellen Troutmen Zeig) (предметный указатель), Рэнди Комера (Randy Comer) (дизайн обложки книги) и Рона Билоде (Ron Bilodeau) (дизайн книги).

Наконец, я хочу поблагодарить Эди Фридман (Edie Freedman) (самую лучшую начальницу) за ее терпение, когда я с головой уходила в процесс написания книги. И моих дорогих, любимых Джефа и Арно. Я счастлива наконец сказать: «Я вернулась!»

Об авторе

Дженнифер Роббинс начала работать в области веб-дизайна в 1993 году в качестве графического дизайнера первого коммерческого веб-сайта системы Global Network Navigator. Помимо этого издания, она также является автором книг «Web Design in a Nutshell» и «HTML5 Pocket Reference» (доступной также в виде приложения для операционной системы iOS), опубликованных издательством O'Reilly. В прошлом Дженнифер выступала на многих конференциях, в том числе на «Seybold» и «South By Southwest» и преподавала начальный курс веб-дизайна в университете Johnson and Wales University в Провиденсе, штат Род-Айленд. В данный момент она занимается созданием цифровых продуктов для компании «O'Reilly Media», где уделяет основное внимание информационной архитектуре, интерактивному дизайну и созданию веб-сайтов, приложений и электронных книг, приятных в использовании. В свободное время Дженнифер любит что-нибудь мастерить, слушать инди-рок, готовить и заниматься детьми.

Использование примеров кода

Цель этой книги — помочь вам в работе. Можно использовать приведенный в книге код в собственных программах и документации. Вам не нужно запрашивать разрешение издательства, если заимствуете небольшие фрагменты кода. Например, для написания программы, в которой используется несколько фрагментов кода из данной книги, вам не потребуется разрешение. Для продажи и распространения оптического диска с примерами необходимо получить разрешение.

Если вы цитируете книгу и приводите пример кода, отвечая на вопрос, разрешение не нужно, а вот для включения значительного количества примеров кода из этой книги в документацию по собственному продукту оно вам потребуется.

Нам будет приятно, если вы сошлетесь на эту книгу как на источник, но это необязательно. Ссылка обычно содержит название книги, фамилию и имя автора, сокращение от города издания, название издательства и год. Например: Дженнифер Роббинс. Веб-дизайн: HTML, CSS и JavaScript. М.: Эксмо, 2013.

Если использование примеров кода требует получения разрешения, обращайтесь к нам по адресу **info@eksmo.ru**.

НАЧАЛО РАБОТЫ ЧАСТЬ I

В этой части

Я хочу лишь блог!

Вам необязательно становиться веб-дизайнером, чтобы начать публиковать свои изречения и рисунки во Всемирной паутине. Вы можете создать собственный *блог*, используя одну из бесплатных или недорогих платформ. Эти ресурсы предоставляют шаблоны, что избавляет от необходимости изучать язык HTML (хотя и это не повредит). Ниже представлены некоторые из популярных платформ блогов:

- WordPress (**www.wordpress.com**)
- Blogger (**www.blogger.com**)
- Tumblr (**www.tumblr.com**)
- Squarespace (**www.squarespace.com**) — еще один сервис, позволяющий создавать сайты простым перетаскиванием и предлагающий услуги хостинга. Его возможности не ограничиваются только созданием блогов.

НА ЗАМЕТКУ

Термин «веб-дизайн» применим к целому ряду дисциплин, в том числе к:

- Визуальному (графическому) дизайну
- Проектированию пользовательского интерфейса и опыта взаимодействия
- Производству веб-документации и таблиц стилей
- Написанию сценариев и программированию
- Контент-стратегиям
- Созданию мультимедийных элементов

С чего начать?

Ваша индивидуальная отправная точка, без сомнения, зависит от вашего опыта и целей. Однако изучение принципов функционирования Интернета и веб-страниц — отличный первый шаг для каждого. Эта книга предоставляет такие базовые знания. Овладев основами, вы сможете посетить множество ресурсов во Всемирной паутине и приобрести книги для более глубокого изучения определенных областей. Существует множество уровней вовлечения в веб-дизайн — от создания персонального веб-сайта до превращения разработки страниц в полномасштабную карьеру. Вам может понравиться быть специалистом широкого профиля или профессионалом в чем-то одном, например в сфере разработки на платформе Flash.

Если вы относитесь к веб-дизайну как к хобби или хотите опубликовать один или два веб-проекта, то самостоятельное обучение (такое как чтение этой книги) в сочетании с использованием доступных шаблонов и надежных инструментов веб-дизайна (например, программы Dreamweaver корпорации Adobe) — это, возможно, все, что вам нужно для выполнения поставленной задачи. Многие программы повышения квалификации предлагают вводные курсы по веб-дизайну и производству веб-сайтов.

Если же вы интересуетесь веб-дизайном для построения карьеры, то должны довести свои навыки до профессионального уровня. Работодатели могут не требовать сертификатов курсов по веб-дизайну, но они ожидают увидеть примеры работающих сайтов в подтверждение ваших навыков и опыта. Это могут быть результаты учебных заданий, персональные проекты или корпоративные ресурсы. Важно, чтобы они выглядели профессионально и строились с использованием четких, правильно функционирующих HTML-кода, таблиц стилей и, возможно, закадровых сценариев. Получение работы начального уровня и работа в составе команды — отличный способ узнать, как построены большие сайты, и решить, какими аспектами веб-дизайна вы хотели бы заниматься.

Задачи веб-дизайнера

Со временем термин «веб-дизайн» превратился в общее понятие, описывающее процесс, который фактически охватывает много различных дисциплин, от графического дизайна до разметки документа и серьезного программирования. В этом разделе описываются некоторые наиболее распространенные сферы деятельности.

Если вы самостоятельно проектируете небольшой веб-сайт, придется быть мастером на все руки. Хорошая новость заключается в том, что вы, вероятно, этого не заметите. Ежедневное содержание домашнего хозяйства требует, чтобы вы были и поваром, и уборщиком, и бухгалтером, и дипломатом, и садовником, и рабочим-строителем — но для вас это только обязанности, которые нужно делать по дому. Таким же образом в качестве самостоятельного веб-дизайнера вы станете и графическим дизайнером,

и сценаристом, и верстальщиком HTML-кода, и информационным архитектором, но для вас это будет выглядеть только как «создание веб-страниц».

Если вы не заинтересованы стать независимым веб-дизайнером на все руки, то можете выбрать специализацию и работать в составе команды или как фрилансер (по договору).

Выполнение задач, для решения которых у вас нет навыков, можно поручить приглашенным специалистам. Например, я создаю веб-сайты с 1993 года и все еще нанимаю программистов и мультимедийных разработчиков, когда клиенты требуют интерактивные элементы. Это позволяет сосредоточиться на тех частях проекта, которые я могу сделать качественно (в моем случае это организация контента, проектирование интерфейса и визуальный дизайн).

Крупномасштабные веб-сайты почти всегда создаются командой людей, насчитывающей от нескольких сотрудников до сотен человек. При таком сценарии каждый член команды сосредотачивается только на одном аспекте создания сайта. Вы можете просто приспособить свои навыки (написание текстов, владение программой Photoshop, программирование и т. д.) и интересы к новой среде.

Я разделила множество ролей и обязанностей, обычно понимаемых под словом «веб-дизайн», на четыре очень широкие категории: дизайн, разработка, контент-стратегии и создание мультимедийных элементов.

Дизайн

Дизайн! Звучит довольно просто, но даже это требование разделено на ряд специальностей, когда дело доходит до создания сайтов. Ниже представлено несколько профессий, связанных с проектированием сайта. Но имейте в виду, что дисциплины часто пересекаются, и человек, называющий себя дизайнером, нередко несет ответственность за несколько (если не за все) из этих обязанностей.

Проектирование взаимодействия, опыта взаимодействия и пользовательского интерфейса

Часто под дизайном мы подразумеваем внешний вид. Но во Всемирной паутине сначала обращают внимание на то, как сайт работает. Прежде чем выбирать цвета и шрифты, важно определить цели сайта, как он будет использоваться и как посетители будут по нему перемещаться. Эти задачи относятся к таким видам профессиональной деятельности, как *проектирование взаимодействия* (*IxD*-проектирование*), *проектирование пользовательского интерфейса* (*UI**-проектирование*) и *проектирование опыта взаимодействия* (*UX***-проектирование*). У этих обязанностей много общего, и нередко один человек или команда выполняют все три. Цель *проектировщика взаимодействия* — сделать сайт как можно более простым, эффективным и приятным в использовании. С проектированием взаимодействия тесно связан дизайн пользовательских интерфейсов, который обычно более сосредоточен на

* IxD — сокращение от англ. Interaction Design — проектирование взаимодействия (примеч. ред.)

** UI — сокращение от англ. User Interface — пользовательский интерфейс (примеч. ред.)

*** UX — сокращение от англ. User eXperience — опыт взаимодействия (примеч. ред.)

функциональной организации страницы, а также конкретных инструментах (кнопках, ссылках, меню и так далее), с помощью которых пользователи перемещаются по контенту или выполняют задачи.

Более новая профессия в мире веб-дизайна — *проектировщик опыта взаимодействия*. Такой специалист использует более целостный подход, чтобы гарантировать положительный опыт взаимодействия от работы с сайтом. Проектирование опыта взаимодействия требует глубокого понимания пользователей и их потребностей на основе наблюдений и интервью.

По словам Дональда Нормана, который ввел этот термин, проектирование опыта взаимодействия включает в себя все аспекты взаимодействия пользователя с продуктом: как он воспринимается, изучается и используется. Для веб-сайта или приложения это подразумевает графический дизайн, пользовательский интерфейс, качество контента и содержащееся в нем сообщение, а также общую производительность сайта.

К числу документов, создаваемых проектировщиками взаимодействия, пользовательского интерфейса или опыта взаимодействия пользователя, относятся:

Изучение пользователей и отчеты о тестировании

Понимание потребностей, желаний и ограничений пользователей играет основную роль в успехе дизайна сайта или веб-приложения. Этот подход к дизайну с учетом потребностей пользователей называется *дизайном, ориентированным на пользователя*, и является основным направлением современного дизайна. Дизайн сайта часто начинается с изучения пользователей, в том числе посредством интервью и наблюдений, чтобы лучше понять, как сайт может решить проблемы или как он будет использоваться. Обычно дизайнеры проводят пользовательское тестирование на каждом этапе процесса разработки, чтобы обеспечить необходимый уровень практичности своего дизайна. Если пользователи с трудом могут выяснить, где найти контент или как перейти к следующему шагу процесса, стоит вернуться к проектированию.

Каркас сайта

Каркас сайта отображает структуру веб-страницы, используя только контуры для каждого типа контента и виджетов (рис. 1.1). Задача каркаса — сообщить, как распределяется пространство на экране, и показать, где находятся функциональные элементы и контент (например, навигация, поле поиска, элементы формы и так далее) без каких-либо декоративных или графических элементов. Как правило, он сопровождается инструкциями о том, как все должно действовать, чтобы команда разработчиков знала, что следует проектировать.

Схема сайта

Схема сайта показывает структуру сайта в целом и то, как отдельные страницы связаны друг с другом. На рис. 1.2 представлена простая схема веб-сайта. А некоторые схемы могут занимать целые стены!

Раскадровки и пользовательские блок-схемы

Раскадровка отслеживает путь обычного пользователя (*персоны* на жаргоне проектировщиков опыта взаимодействия) через сайт или приложение. Этот путь обычно включает в себя сценарий и «сцены», состоящие из изображений

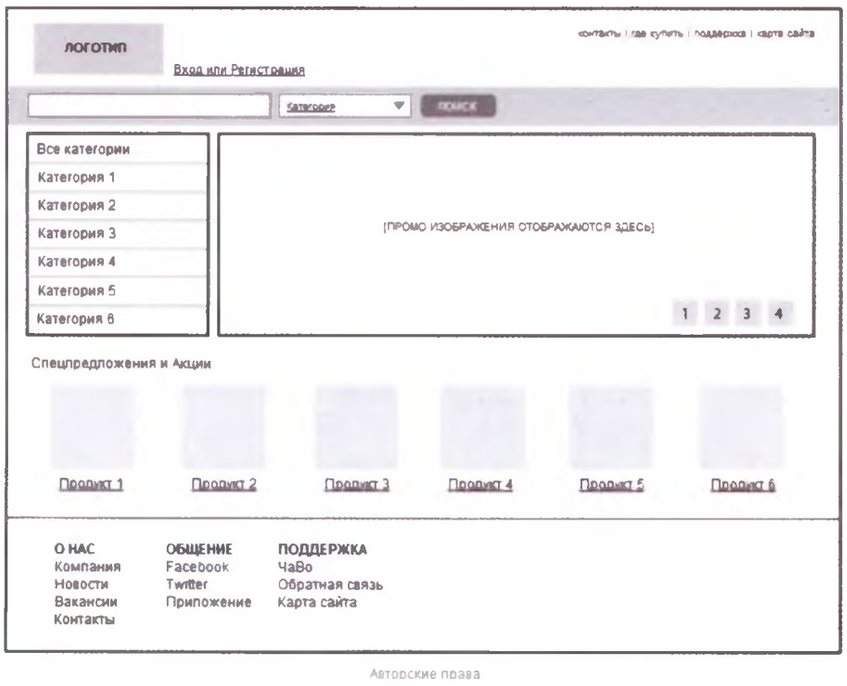


Рис. 1.1. *Схема каркаса*

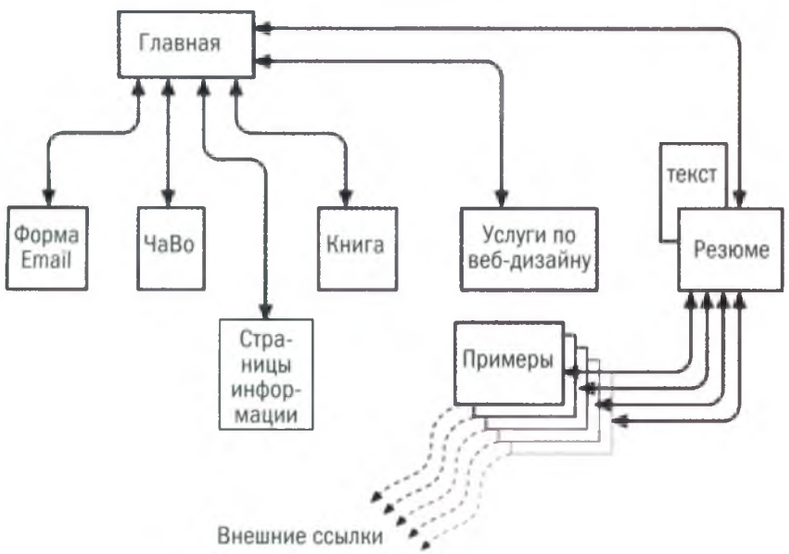


Рис. 1.2. *Простая схема веб-сайта*

экрана или взаимодействий пользователя с экраном. Раскадровка призвана продемонстрировать шаги, необходимые для выполнения задач, возможные варианты, а также вводит некоторые стандартные типы страниц. На рис. 1.3 представлена простая раскадровка. Пользовательская блок-схема — еще один способ показать, как связаны между собой части сайта или приложения. Его основной принцип заключается в том, чтобы сосредотачиваться на технических деталях, а не описывать последовательность событий. Например такое-то действие пользователя запускает функцию на сервере. Обычно дизайнеры создают пользовательские блок-схемы для пошаговых процессов, таких как регистрация участников сообщества или онлайн-платежи.

Style Tiles

Другой подход к созданию «look and feel» сайта — использование стиля Style Tiles, который содержит примеры цветовых схем, элементы брендинга, контента и пользовательского интерфейса, а также палитры настроения, без применения их к конкретным макетам страницы. Идея состоит в том, чтобы прийти к общему знаменателю относительно согласованного визуального языка сайта. См. сайт **styletil.es**.

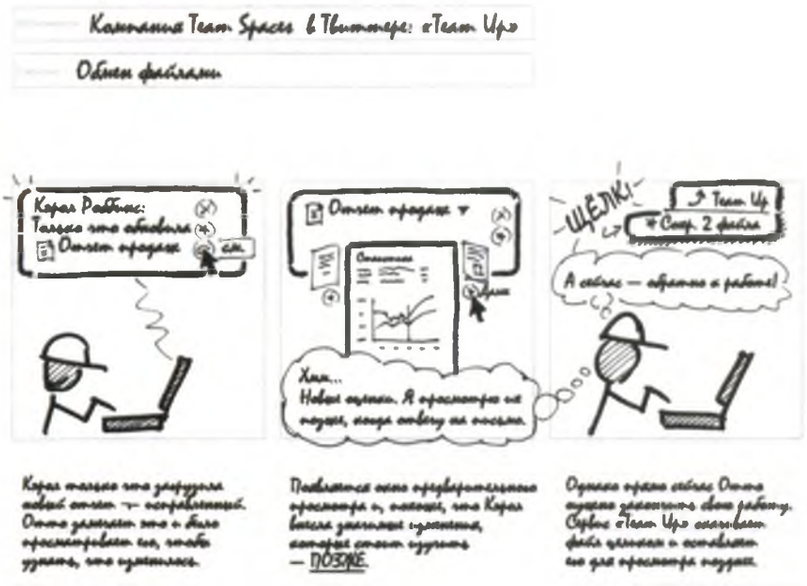


Рис. 1.3. Типичная раскадровка [предоставленная компанией Adaptive Path и нарисованная Брэндоном Шоером]

Рис. 1.4. Эскизы «look and feel» простого веб-сайта

Графический дизайн

Поскольку Всемирная паутина — визуальная среда, веб-страницы требуют особого внимания к проектированию их внешнего вида. Графический дизайнер создает «look and feel» — внешний вид и поведение сайта — логотипы, графики, шрифты, цвета, макеты и т. д., чтобы обеспечить положительное первое впечатление и согласованность с брендом и миссией организации, которую он представляет. Графические дизайнеры обычно создают эскизы возможных вариантов внешнего вида сайта, как показано на рис. 1.4. Кроме того, они могут быть ответственны за подготовку графических файлов, оптимизированных для доставки через Интернет (о техниках оптимизации изображений читайте в главе 19).

Если вы планируете профессионально заняться разработкой визуального дизайна коммерческих сайтов, я настоятельно рекомендую пройти курсы графического дизайна, а также изучить программу Adobe Photoshop (стандарт в данной области) или Adobe Fireworks.

Если вы уже работаете графическим дизайнером, то легко приспособите навыки под веб-дизайн, хотя это не значит, что вам не нужно хорошо разбираться в HTML, CSS и других веб-технологиях. Поскольку на большинстве веб-сайтов присутствует хотя бы несколько изображений, даже непрофессиональным веб-дизайнерам необходимо получить минимальные знания по использованию графических редакторов.

Я опять же подчеркиваю, что все эти обязанности могут выпасть на долю одного дизайнера, создающего и внешний вид, и функциональные элементы сайта. Однако, работая с крупными веб-сайтами, имеющими большой бюджет, есть шанс найти собственную нишу в сфере дизайна.

Разработка

Изрядный объем времени процесса веб-дизайна занимает создание и поиск проблем в документах, таблицах стилей, сценариях и изображениях, которые составляют сайт. В больших компаниях, занимающихся веб-дизайном, группа, которая занимается созданием файлов, составляющих веб-сайт, может быть названа отделом *разработки* или *производства*.

Веб-разработчики могут не разрабатывать внешний вид и структуру сайта сами, но они должны тесно общаться с дизайнерами и понимать намеченные цели сайта, чтобы они могли предложить решения, отвечающие этим целям.

Понятие «разработка» включает в себя такие широкие дисциплины, как авторская разработка, создание стилей и написание сценариев/программирование.

Верстка/разметка

Верстка — это термин, обозначающий процесс подготовки контента для публикации во Всемирной паутине, или, точнее, разметки контента с помощью HTML-тегов, описывающих содержимое и его функции. Если вы хотите стать веб-разработчиком, вам потребуются знания *тонкостей* языка HTML и того, как он функционирует в различных браузерах и устройствах. Спецификация HTML постоянно развивается, и это означает, что вы должны идти в ногу с последними достижениями и возможностями, а также знать об ошибках и ограничениях.

К счастью, начинать не сложно, а потом вы сможете постепенно совершенствовать свои умения. Мы займемся языком HTML в главе 2, а затем обсудим его подробно во второй части этой книги.

Стилизация

В веб-дизайне, внешний вид страницы в браузере определяется правилами стилей, прописанными в CSS (каскадных таблицах стилей). Мы углубимся в изучение CSS в третьей части этой книги (а также разберемся, что означает слово «каскадные»!), а пока просто уясните, что в современном веб-дизайне, внешний вид страницы обрабатывается отдельно от HTML-разметки страницы. Опять же, если вас интересует веб-разработка, знание стилей CSS и того, как они поддерживаются (или не поддерживаются) браузерами, гарантированно будет частью ваших должностных обязанностей.

Создание сценариев и программирование

Так как Всемирная паутина превратилась в платформу приложений для выполнения различных задач, программирование еще никогда не играло такой важной роли. JavaScript — это язык, заставляющий элементы страницы выполнять определенные действия. Он добавляет поведение и функциональность этим элементами даже самому окну браузера.

Front-end- и back-end-разработка

Вы можете услышать, как веб-дизайнеры и разработчики говорят, что они специализируются на *front-end-* и *back-end*-разработке веб-сайта.

Front-end-разработка
Front-end относится к любому аспекту процесса разработки, который проявляется при загрузке страницы в браузере или имеет к этому непосредственное отношение. Эта книга сосредоточена, прежде всего, на front-end-разработке.

Следующие задачи обычно относят к front-end-разработке:

- Графический дизайн и подготовка изображений
- Дизайн интерфейса
- Информационный дизайн, в той мере, в какой он имеет отношение к опыту взаимодействия пользователя с сайтом
- Верстка HTML-документов и таблиц стилей
- JavaScript

Back-end-разработка
Back-end-разработка относится к программам и сценариям, которые скрытно выполняются на стороне сервера, чтобы сделать веб-страницы динамичными и интерактивными. В целом, back-end-разработка обычно выполняется опытными программистами, но всем веб-дизайнерам пригодятся знания функциональных возможностей back-end-разработки.

Следующие задачи относятся к back-end-разработке:

- Информационный дизайн, в той мере, в какой он имеет отношение к организации данных на сервере
- Обработка форм
- Программирование баз данных
- Системы управления контентом
- Другие веб-приложения, работающие на стороне сервера, использующие PHP, JSP, Ruby, ASP.NET, Java и другие языки программирования.

ПРИМЕЧАНИЕ

Многие графические дизайнеры сами переводят свои проекты в документы HTML и CSS. На самом деле, бытует распространенное мнение, что для того, чтобы называться «веб-дизайнером», нужно уметь создавать дизайн самостоятельно, и почти все согласны, что ваши шансы при поиске работы возрастают, если вы можете не только создавать дизайны, но и писать код.

Существуют и другие языки программирования для Всемирной паутины, в том числе PHP, Ruby, Python и ASP.NET, которые запускаются на стороне сервера и обрабатывают данные и информацию перед отправкой в браузер пользователя. Подробнее о том, что происходит на стороне сервера, см. во врезке «Front-end и back-end разработка».

Создание веб-сценариев и программирование определенно требует некоторых навыков программирования. В то время как многие веб-программисты — дипломированные специалисты, нередко они оказываются и самоучками. Разработчики, которых я знаю, обычно начинали с копирования и адаптации существующих сценариев, а затем постепенно развивали свои навыки программирования. И все же, если у вас нет опыта программирования, вначале вы можете испытывать некоторые сложности.

Обучение веб-программированию выходит за рамки этой книги. С языком JavaScript вы познакомитесь в главе 21 (обучению работе с JavaScript посвящены отдельные книги). Можно создавать отвечающие всем требования, богатые контентом, отлично разработанные сайты, не имея навыков программирования, поэтому веб-дизайн не должен отпугивать непрофессионалов. Однако, как только понадобится собрать сведения посредством форм или вывести информацию по запросу пользователя, обычно в команде требуется программист. Вы также можете спросить сотрудников вашей хостинговой компании, предлагают ли они нужные вам элементы функциональности в форме простых в использовании готовых сервисов.

Контент-стратегии и наполнение

Третьим в нашем списке, хотя в идеале первым в самом процессе создания сайта, стоит важный вопрос о контенте самого сайта. Каждый, кто называет себя «веб-дизайнером», должен быть в курсе, что все наши действия направлены на поддержание процесса передачи контента или функциональности нашим пользователям. Кроме того, хорошо написанный текст может помочь повысить эффективность создаваемых нами пользовательских интерфейсов.

Конечно, кто-то должен создавать и поддерживать контент — не стоит недооценивать, сколько сил требуется на то, чтобы делать это успешно. Кроме того, я хочу обратить ваше внимание, что в современной команде веб-разработчиков контентом занимаются два специалиста: контент-стратег и информационный архитектор.

Если контент написан неверно, сайт не может быть эффективным в полной мере. *Контент-стратег* гарантирует, что каждый фрагмент текста на сайте, от длинных описательных текстов до меток или кнопок, способствует развитию фирменного стиля и маркетинговых целей компании. Контент-стратегия может также распространяться на моделирование данных и управление контентом в больших и постоянных масштабах, например планирование многократного использования контента и режима обновлений.

Информационный архитектор (также называемый *информационным дизайнером*) организует контент логически для удобства нахождения информации. Он может быть ответственным за функционирование поиска, схемы сайта, а также способы организации контента и данных на веб-сервере. Информационная архитектура неизбежно переплетается с проектированием опыта взаимодействия и пользовательских интерфейсов, и нередко всем этим занимается один человек.

Мультимедиа

Одной из эффектных сторон Всемирной паутины является то, что вы можете добавить на сайт мультимедийные элементы, такие как звуки, видеофайлы, анимация и даже интерактивные игры. Возможно, вы захотите освоить навыки работы с мультимедийным контентом — например редактирование аудиофайлов, видеомонтаж или Flash-программирование (см. врезку «Несколько слов о Flash») вдобавок к уже имеющимся дизайнерским умениям — или заняться только этим и стать специалистом в данной сфере.

Если не привлекает возможность превратиться в разработчика мультимедийных элементов, вы всегда можете нанять соответствующего специалиста.

Компании по разработке веб-продуктов обычно ищут людей, которые освоили стандартные мультимедийные инструменты, а также имеют отличное визуальное чутье и инстинкт для интуитивного и креативного дизайна мультимедийных проектов.

Несколько слов о Flash

Adobe Flash (ранее Macromedia Flash, а еще ранее — FutureSplash) — мультимедийный формат, разработанный специально для Всемирной паутины. Эта технология предоставляет возможность создавать полноэкранную анимацию, интерактивную графику, интегрированные аудио- и видеоклипы, даже основанные на сценариях игры и приложения, и представлять все это в файлах небольшого размера. Однако в последнее время происходит снижение интереса к Flash-технологии вследствие ряда причин:

- Решения корпорации Apple не обеспечивать поддержку Flash на устройствах iPhone и iPad, отдав предпочтение бесплатным методам языка HTML5.
- Решения корпорации Adobe прекратить поддержку Flash (своего собственного продукта) в мобильных браузерах.
- Нового программируемого элемента **canvas** в языке HTML5, который обеспечивает некоторую функциональность, аналогичную платформе Flash.
- Критики того, что Flash-элементы иногда мешают пользователям. Например, кому захочется просматривать целиком ролик или прослушивать звуковое сопровождение на сайте ресторана, если всего лишь нужно узнать, открыт ли он в воскресенье?
- Некоторые разработчики с неприязнью относятся к элементам Flash, так как для их воспроизведения требуется плагин.

Нередко веб-профессионалы заявляют, что «технология Flash мертва», но несмотря на такое резкое превращение в аутсайдера, платформа Flash все равно имеет ряд преимуществ, при правильном использовании.

- Поскольку используется векторная графика, файлы имеют небольшой размер и изображения масштабируются без потери детализации.
- Это потоковый формат, поэтому видеоролики начинают воспроизводиться сразу после начала загрузки параллельно со скачиванием данных.
- Вы можете использовать язык ActionScript, чтобы назначить объектам поведение и добавить расширенную интерактивность, применяя объекты Flash в качестве элементов интерфейса для динамически генерируемого контента или функций электронной коммерции.
- Плагин для поддержки технологии Flash широко распространен и поддерживается большинством браузеров настольных компьютеров.
- Несмотря на то что HTML5 — многообещающий и быстроразвивающийся язык, на момент написания книги HTML5 отстает по свойствам и функциональности от платформы Flash.

Технология Flash не исчезнет мгновенно, хотя даже корпорация Adobe прилагает усилия по разработке альтернативных вариантов обеспечения должной функциональности средствами языка HTML5.

Требуемые знания

НА ЗАМЕТКУ

Веб-технологии:

- Язык гипертекстовой разметки (HTML)
- Каскадные таблицы стилей (CSS)
- Сценарии JavaScript/DOM
- Сценарии на стороне сервера и управление базами данных

Если вы — графический дизайнер, работающий в программах Photoshop и Illustrator, вас может выбить из колеи необходимость научиться создавать свои проекты с помощью текста, но я уверяю, что начать довольно просто. Кроме того, существуют средства разработки, ускоряющие процесс верстки, которые мы обсудим далее в этой главе.

Ниже приводится список технологий, связанных с веб-разработкой. Какие языки и технологии вы изучите, зависит от того, в какой роли вы видите себя в процессе веб-дизайна. Однако я советую *всем*, кто занимается созданием веб-сайтов и желающим превратить front-end разработку в источник заработка, освоить язык HTML и каскадные таблицы стилей. Веб-профессионалы, подкованные с технической стороны, могут дополнительно изучить конфигурирование сервера, базы данных и вопросы производительности сайта, но, как правило, перед front-end-разработчиками такие задачи не ставятся (хотя базовое знакомство с работой сервера никогда не помешает).

Язык гипертекстовой разметки (HTML)

HTML (HyperText Markup Language, Язык гипертекстовой разметки) — язык, используемый для создания документов веб-страниц. В настоящее время используется несколько версий HTML: прочно утвердился HTML 4.01, а более новая и мощная, черновая спецификация HTML5 обретает популярность и получает все большую поддержку в браузерах. У этих двух версий есть более узкая реализация, называемая XHTML (eXtensible HTML, расширяемый HTML). Это, по существу, тот же самый язык HTML с более строгими правилами синтаксиса. Мы поговорим об отличиях этих версий в главе 10.

Обычно отсылки на HTML и XHTML имеют вид (X)HTML.

HTML —язык не программирования, а разметки, он создает систему для идентификации и описания различных компонентов документа, таких как заголовки, абзацы и списки. Разметка обозначает скрытую *структуру* документа (можно сказать, что это подробная машиносчитываемая схема). Чтобы написать HTML-код, вам не нужны навыки программирования — только терпение.

Лучший способ изучить язык HTML — написать код нескольких страниц вручную, что мы и сделаем в упражнениях этой книги.

Если вы в конечном итоге займетесь веб-дизайном, язык HTML будет вашим воздухом. Даже людям, увлеченным версткой веб-страниц лишь как хобби, принесут пользу знания о том, как все это функционирует. Хорошая новость заключается в том, что основы изучить легко.

Консорциум Всемирной паутины

Консорциум Всемирной паутины (World Wide Web) (именуемый сокращенно W3C) — это организация, которая наблюдает за развитием веб-технологий. Группа была основана в 1994 году Тимом Бернерс-Ли, изобретателем Всемирной паутины, в Массачусетском технологическом институте.

В начале, консорциум Всемирной паутины интересовался главным образом протоколом HTTP и развитием языка HTML. Теперь же он закладывает основы для будущего Всемирной паутины, развивая множество технологий и протоколов, которые должны взаимодействовать в надежной инфраструктуре.

Для получения ответов на любые вопросы по веб-технологиям перейдите на сайт консорциума Всемирной паутины по адресу **www.w3.org**.

Для получения дополнительной информации о консорциуме Всемирной паутины и о том, чем занимаются его сотрудники, см. **www.w3.org/Consortium/**.

Каскадные таблицы стилей (CSS)

В то время как HTML используется, чтобы описать содержимое веб-страницы, именно каскадные таблицы стилей (Cascading Style Sheets, CSS) влияют на то, как *выглядит* контент. Говоря о веб-дизайне, способ, которым представлена страница, известен как ее *представление*. Это означает, что шрифтами, цветами, фоновыми изображениями, интервалами между строками, макетом страницы и прочим... управляют CSS. С помощью новейшей версии (CSS3) вы можете добавлять на страницу даже специальные эффекты и простую анимацию.

Каскадные таблицы стилей также управляют представлением документов не только в браузерах, но и в таких контекстах, как печать и устройства с экранами с малой диагональю. Кроме того, в таблицах стилей существуют правила для определения невизуального представления документов, например как будет звучать текст при считывании его программой экранного доступа (однако они не очень хорошо поддерживаются).

Таблицы стилей —отличный инструмент, позволяющий автоматизировать процесс разработки, потому что вы можете производить изменения, относящиеся ко всем страницам сайта, редактируя один-единственный документ таблицы стилей. Они в некоторой степени поддерживаются всеми современными браузерами.

ПРИМЕЧАНИЕ

Когда в этой книге говорится о «таблицах стилей», это всегда относится к каскадным таблицам стилей (CSS), стандартному языку таблиц стилей Всемирной паутины.

Хотя вполне возможно создавать веб-страницы, используя только HTML-теги, вы, вероятно, захотите применить таблицы стилей, чтобы не ограничивать себя стилями браузеров, заданными по умолчанию. Если вы занимаетесь проектированием веб-сайтов профессионально, мастерство использования таблиц стилей просто необходимо.

Таблицы стилей обсуждаются далее, в части III данной книги.

Сценарии JavaScript/DOM

JavaScript — язык сценариев, который наделяет веб-страницы интерактивностью и вариантами поведения, включая следующие (перечислены далеко не все):

- Проверка правильности значений, введенных в элементы формы
- Замена стилей для элемента или всего сайта
- Требование от браузера запомнить информацию о пользователе для следующего сеанса его посещения
- Создание интерфейсных виджетов, таких как раскрывающиеся меню.

Язык JavaScript используется для управления элементами на веб-странице, примененными к ним стилями, или даже самим браузером. Существуют и другие языки веб-сценариев, но JavaScript (также называемый ECMAScript) стандартизирован и наиболее широко распространен.

Вы можете также услышать термин *сценарии объектной модели документа* или *сценарии DOM*, используемый в отношении JavaScript. Технология DOM является сокращением термина *объектная модель документа* (Document Object Model) и обращается к стандартизированному списку элементов веб-страницы, к которым можно получить доступ и управлять ими, используя JavaScript (или другой язык сценариев). Сценарии DOM используются взамен подхода DHTML (Dynamic HTML, динамический HTML), который теперь считается устаревшим.

Написание кода на языке JavaScript — по сути, программирование, обучение этому языку может занять много времени, если у вас нет опыта в данной сфере. Многие люди самостоятельно изучают язык JavaScript, читая книги, а также используя уже созданные сценарии и изменяя их. Большинство инструментов для верстки веб-страниц поставляется со стандартными сценариями, которые вы можете использовать наряду с прочими функциями этого программного обеспечения.

Профессиональные веб-разработчики обязаны знать язык JavaScript, однако многие дизайнеры полагаются на программистов в добавлении поведений в свои проекты. Таким образом, хотя код на языке JavaScript и полезен, изучение его не является обязательным для *всех*

Три «слоя» веб-дизайна

Современный веб-дизайн можно представить как состоящий из трех отдельных «слоев».

Содержимое документа с разметкой (X)HTML составляет *структурный слой*. Он формирует основу, на которой могут применяться другие слои.

Как только структура документа создана, вы можете добавить информацию из таблицы стилей, чтобы управлять тем, как должно выглядеть содержимое. Это называют *слоем представления*.

Наконец, *слой поведения* включает сценарии, которые наделяют страницу интерактивностью.

веб-дизайнеров. Эта книга не научит вас языку JavaScript, поэтому стоит обратить внимание, например, на книгу Дэвида Макфарланда. «JavaScript. Подробное руководство» (Эксмо, 2009), если вы хотите узнать больше.

Программирование на стороне сервера

Некоторые простые веб-сайты — коллекции статических HTML-документов и файлов изображений, но большинство коммерческих сайтов имеет более продвинутые функциональные возможности, такие как поддержка форм, динамически генерируемые страницы, корзины покупателей, системы управления контентом, базы данных и т. д. Этот функционал поддерживается специальными приложениями, запущенными на стороне сервера. Существует множество языков сценариев и программирования, которые используются для создания веб-приложений, включая следующие:

- PHP (CakePHP, CodeIngniter, Drupal)
- Python (Django, TurboGears)
- Ruby (Ruby on Rails, Sinatra)
- JavaScript (Node.js, Rhino, SpiderMonkey)
- Java (Grails, Google Web Toolkit, JavaServer Faces)
- ASP.Net (DotNetNuke, ASP.Net MVC)

Разработка веб-приложений — задача программистов и не требуется от всех веб-дизайнеров. Однако это не означает, что вы не можете предложить такие функциональные возможности вашим клиентам. Покупательские корзины, системы управления контентом, списки рассылок и гостевые книги легко можно приобрести как готовые решения без необходимости программировать их с нуля.

Необходимое обеспечение

Неудивительно, что профессиональные веб-дизайнеры нуждаются в изрядном количестве, как аппаратных средств, так и программного обеспечения. Один из самых общих вопросов, который мне задают мои студенты: «Что я должен приобрести?» Я не могу сказать вам определенно, что купить, но предоставлю краткий обзор типичных инструментов, которые вам понадобятся.

Примите во внимание, что в то время как я перечислю самые популярные платные программы, у многих из них существуют бесплатные или условно-бесплатные эквиваленты, которые вы можете загрузить, если ограничены в средствах (посетите ресурс **www.softodrom.ru**). Немного усилий, и вы сможете создать полноценный веб-сайт, настроенный и прекрасно функционирующий без особых затрат

Краткое знакомство с XML

Если вы знакомы с миром веб-дизайна, то наверняка слышали аббревиатуру *XML* (*расширяемый язык разметки*). Вообще-то XML не совсем язык, он, скорее, представляет собой свод правил для того, чтобы создавать другие языки разметки.

В качестве упрощенного примера: если бы вы публиковали на странице рецепты, то могли бы использовать XML, чтобы создать пользовательский язык разметки, который включает элементы (теги) **<компонент>**, **<инструкции>** и **<порции>**, которые точно описывают типы информации в ваших документах рецептов. Один раз правильно помеченная, эта информация может рассматриваться как данные. Фактически, XML — мощный инструмент для совместного использования данных разными приложениями. Несмотря на то, что он разрабатывался принципиально для Всемирной паутины, фактически XML оказал большее влияние вне веб-среды из-за своих возможностей обработки данных. Существуют файлы XML, работающие «за кадром» в растущем числе программных приложений, таких как Microsoft Office, Adobe Flash и Apple iTunes.

Однако существуют многие языки XML, которые используются и во Всемирной паутине. Наиболее распространен XHTML, который является языком HTML, переработанным в соответствии с более строгими правилами XML (Мы поговорим подробнее об XHTML в главе 10). Существуют также: RSS (Really Simple Syndication, очень простой сбор сводной информации), который позволяет представлять ваш контент в виде кратких публикаций канала RSS, SVG (Scalable Vector Graphics, масштабируемая векторная графика), который использует теги, чтобы описать геометрические формы, и MathML, описывающий математическое обозначения.

Ваш непосредственный опыт работы с XML как веб-дизайнера, вероятно, ограничится созданием документов с помощью языка XHTML или, возможно, добавлением к веб-сайту канала RSS или изображений в формате SVG. Развитие новых языков XML должно быть обязанностью программистов или специалистов по XML.

Аппаратное обеспечение

Для удобства при разработке веб-сайта я рекомендую следующее оборудование:

Надежный, современный компьютер. Компьютер под управлением Windows, Linux или OS X подойдет. Креативщики в профессиональных компаниях по веб-разработке имеют тенденцию работать на компьютерах Mac. Хотя и приятно работать на сверхпроизводительном компьютере, файлы веб-сайтов очень малы и при работе с ними требуют не так много ресурсов. Если только вы не занялись обработкой звуковых файлов и видеомонтажом, не беспокойтесь о том, что ваш компьютер не самый новый и производительный.

Дополнительная оперативная память. Поскольку вам придется переключаться между многими, приложениями, нужно установить достаточный объем оперативной памяти, чтобы запускать одновременно несколько программ, задействующих значительные ее объемы.

Монитор с большой диагональю. Хотя это и не обязательное требование, монитор с большой диагональю или высоким разрешением облегчает жизнь, особенно графическому дизайнеру (хотя я видела, как верстальщики кода прекрасно управлялись на ноутбуке MacBook Air с диагональю 11 дюймов). Чем больше размер экрана вашего монитора, тем больше окон приложений и панелей управления вы сможете от-

крыть одновременно. Чтобы принимать важные решения по дизайну страниц, вы должны видеть бо́льшую их часть одновременно.

Если вы используете монитор с высоким разрешением, при работе над дизайном страниц помните о пользователях, пользующихся мониторами с меньшим разрешением.

Сканер и/или цифровая камера. Если собираетесь подготавливать собственную графику, вам понадобятся некоторые инструменты для создания изображений или текстур.

Дополнительный компьютер. Многие веб-дизайнеры используют второй тестовый компьютер, управляемый другой платформой, в отличие от компьютера, который они используют для разработки (иначе говоря, если вы верстаете страницы в операционной системе OS X, тестируйте их в операционной системе Windows, и наоборот). Поскольку браузеры работают по-разному на компьютерах с операционными системами OS X и Windows, важно проверить работу страницы в максимально возможном количестве сред, и особенно в текущей версии операционной системе Windows. Если вы только увлекаетесь веб-дизайном и работаете дома, можно проверить страницы на компьютере друга. Пользователям компьютеров Mac рекомендуется прочитать врезку «Запуск операционной системы Windows на компьютере Mac»

Мобильные устройства. Всемирная паутина стала мобильной! Это означает, что крайне важно протестировать внешний вид и производительность вашего сайта в мобильном браузере на смартфоне или планшете. Возможно, вы сами уже обзавелись смартфоном. Если у вас нет средств на приобретение устройств с различными платформами, попросите друзей потратить несколько минут и протестировать ваш сайт на своих устройствах.

Программное обеспечение

Сейчас нет недостатка в программном обеспечении, пригодном для создания веб-страниц. Раньше мы могли пользоваться только инструментами, изначально разработанными для предпечатной подготовки. Сегодня есть замечательные программы, специально созданные для веб-дизайна, которые повышают эффективность процесса разработки сайтов. Я не могу перечислить все существующее программное обеспечение для веб-дизайна, поэтому хочу познакомить вас с самыми распространенными и проверенными инструментами. Вы можете загрузить демонстрационные версии многих из этих программ с веб-сайтов компаний, как это перечислено во врезке «Популярные инструменты веб-дизайна» далее в этой главе.

Программы для верстки веб-страниц

Инструменты для верстки веб-страниц схожи с программами предпечатной подготовки, различие в конечном продукте — веб-странице

Запуск операционной системы Windows на компьютере Mac

Если у вас есть компьютер Mac с процессором Intel, на котором установлена операционная система OS X версии Leopard или выше, вам не нужен отдельный компьютер для тестирования проектов в среде Windows. Теперь вы можете запускать Windows прямо на компьютере Mac с помощью бесплатного приложения Boot Camp, которое позволяет переключаться на систему Windows при загрузке компьютера.

Существует несколько других продуктов виртуализации для компьютеров Mac, которые позволяет легко переключаться между операционными системами OS X и Windows, в том числе:

- VMFusion (**www.vmware.com/ru/**) — коммерческая программа с бесплатным пробным периодом, доступная для загрузки.
- Parallels Desktop для OS X (**www.parallels.com/ru/**) — коммерческая программа с бесплатным пробным периодом.
- Oracle VirtualBox (**virtualbox.org**) — бесплатная программа, позволяющая запускать несколько «гостевых» операционных систем, включая Windows и некоторые модификации Unix.

Для запуска виртуальной машины требуется приобрести копию операционной системы Microsoft Windows, но все же это дешевле, чем покупка второго компьютера.

ПРИМЕЧАНИЕ

Для выполнения упражнений из этой книги вам будет достаточно текстового редактора, установленного вместе с вашей операционной системой. Специальное программное обеспечение не требуется.

(файл HTML и дополнительные файлы). Эти инструменты обеспечивают визуальный интерфейс «WYSIWYG» (What You See Is What You Get — Что видишь, то и получишь) и использование подсказок, которые освобождают от ввода повторяющегося кода HTML и CSS. Однако эти инструменты не освобождают вас от изучения языка HTML. Даже самые сложные из них не сгенерируют такой чистый и продуманный HTML-код, как при профессиональной верстке вручную, хотя они могут ускорить процесс, если вы уверены в своих знаниях.

Ниже приведены некоторые популярные программы для верстки веб-страниц:

Adobe Dreamweaver (Windows и OS X). Наиболее серьезная и профессиональная программа, способствует получению чистого кода и предлагает расширенные возможности.

Microsoft Expression Web (только Windows). Часть пакета профессиональных средств проектирования корпорации Microsoft, может похвастаться созданием кода, совместимым со стандартами и макетами, основанными на таблицах стилей.

Nvu (Linux, Windows и OS X). Не хотите платить за редактор WYSIWYG? Nvu (произносится «эн-вью») — инструмент с открытым исходным кодом, который соответствует многим возможностям программы Dreamweaver и может быть загружен с веб-сайта **nvu.mozilla-russia.org**.

Редакторы HTML-кода

Редакторы HTML-кода (в противоположность инструментам для верстки) разработаны, чтобы ускорить процесс написания HTML-кода вручную. Они не позволяют редактировать страницу визуально, поэтому необходимо тестировать ее в браузере. Многие профессиональные веб-дизайнеры на самом деле предпочитают создавать HTML-документы вручную и рекомендуют следующие пять инструментов:

TextPad (только Windows). TextPad — простой и недорогой редактор кода для операционной системы Windows.

Sublime Text (Window, OS X и Linux). Этот недорогой и многообещающий редактор кажется урезанным, но предлагает множество функций (например, подсветка синтаксиса и обзор кода целиком), которые очень нравятся разработчикам.

Coda компании Panic (только OS X). Пользователям программы Coda нравится визуальная последовательность действий, инструменты управления файлами и встроенный доступ к терминалу.

TextMate компании MacroMates (только OS X). Этот расширенный текстовый редактор включает инструменты управления проектами и интерфейс, встраиваемый в операционную систему OS X. Его популярность растет, потому что он прост в использовании, многофункционален и недорог.

BBEdit компании Bare Bones Software (только OS X). Множество замечательных возможностей подсветки синтаксиса кода сделали эту программу ведущим редактором для веб-разработчиков, работающих на компьютерах под управлением операционной системы OS X.

ПРИМЕЧАНИЕ

Дистрибутивы или ссылки на все перечисленные программы вы найдете на диске, прилагающемся к книге.

Графические редакторы

Вероятно, вы захотите добавить на ваши страницы изображения, поэтому вам понадобится графический редактор. Подробно я рассмотрю некоторые наиболее популярные графические программы в части IV. Тем временем вы можете изучить следующие инструменты создания графики для веб-страниц:

Adobe Photoshop (Windows и OS X). Photoshop — бесспорный промышленный стандарт для создания изображений, как для печати, так и веб-страниц.

Adobe Photoshop Elements (Windows и OS X). Эта упрощенная версия программы Photoshop разработана для любительского редактирования и организации фотографий, но вы также обнаружите, что она содержит все инструменты, необходимые для создания изображений для веб-страниц.

Adobe Illustrator (Windows и OS X). Так как дизайнерам нужно создавать логотипы, значки и иллюстрации различных размеров и разрешений, многие начинают работу с векторными изображениями в программе Illustrator, чтобы добиться максимальной гибкости. Вы можете сохранять изображения для Всемирной паутины непосредственно из программы Illustrator или переносить их в Photoshop для дополнительной обработки.

Adobe Fireworks (Windows и OS X). Эта программа для создания веб-графики сочетает редактор изображений с инструментами для создания векторных иллюстраций, а также предоставляет дополнительные инструменты для создания графических изображений для Всемирной паутины.

Corel PaintShop Pro (только Windows). Полнофункциональный редактор изображений, популярный среди пользователей Windows, прежде всего из-за относительно низкой цены.

GIMP (Unix Windows и OS X). Этот бесплатный графический редактор функциональностью напоминает Photoshop.

Интернет-инструменты

Так как вы будете иметь дело с Интернетом, вам понадобятся некоторые инструменты, предназначенные для просмотра и перемещения файлов по сети.

Различные браузеры. Поскольку браузеры отображают страницы поразному, вы захотите проверить страницы на максимально возможном количестве браузеров. На рынке представлены сотни браузеров, но указанные ниже наиболее распространены на компьютерах под управлением операционных систем Windows и OS X:

Windows:

- Internet Explorer (текущая версия и по крайней мере две предшествующих версии)
- Firefox
- Chrome
- Opera

OS X:

- Safari
- Chrome
- Firefox
- Opera

И не забывайте о мобильных браузерах. Приведенный ниже список представляет собой обзор наиболее распространенных мобильных веб-браузеров на момент написания книги (хотя кто знает, какие мобильные браузеры будут популярны сейчас, когда вы читаете эту книгу).

- Mobile Safari (iOS)
- Android Browser (Android)
- BlackBerry Browser (RIM)
- Nokia Series 40 и Nokia Browser для Symbian
- Opera Mobile и Mini (устанавливается на любое устройство)
- Internet Explorer Mobile (Windows Phone)
- Silk (Kindle Fire).

Программы для передачи файлов (по протоколу FTP). Такая программа позволит вам передавать файлы между вашим компьютером и сервером, на котором будут размещены ваши страницы во Всемирной паутине. Все программы верстки веб-страниц, перечисленные ранее, имеют встроенные инструменты для работы с протоколом FTP. Существуют также отдельные программы для передачи файлов по FTP, которые приведены ниже.

Windows:

- WS_FTP
- CuteFTP
- AceFTP
- Filezilla

OS X:

- Transmit
- Cyberduck
- Fetch

Терминальные приложения. Если вы знакомы с операционной системой Unix, вам может пригодиться терминальное приложение (интерпретатор командной строки), позволяющее вводить команды Unix на сервере. Оно может быть полезно для изменения разрешений доступа

НА ЗАМЕТКУ

Популярные инструменты веб-дизайна

Верстка веб-страниц

- Adobe Dreamweaver, **www.adobe.com/ru/products/dreamweaver**
- Microsoft Expression Web, **www.microsoft.com/expression**
- Nvu (редактор веб-страниц с открытым исходным кодом), **nvu.mozilla-russia.org**

Редактирование HTML-кода

- TextMate компании MacroMates, **www.macromates.com**
- Sublime Text, **www.sublimetext.com**
- TextPad для Windows, **www.textpad.com/products/textpad/index.html**
- Coda компании Panic Software, **www.panic.com/coda/**
- BBEdit компании Bare Bones Software, **www.barebones.com/products/bbedit**

Обработка графики

- Adobe Photoshop, **www.adobe.com/ru/products/photoshop**
- Adobe Photoshop Elements, **www.adobe.com/ru/products/photoshopel**
- Adobe Illustrator, **www.adobe.com/ru/products/illustrator**
- Adobe Fireworks, **www.adobe.com/ru/products/fireworks**
- Corel PaintShop Pro, **corel.ru/product/pspx5/**
- GIMP, **gimp.ru**

Браузеры

- Internet Explorer (только Windows), **www.microsoft.com/rus/windows/internet-explorer**
- Firefox, **www.mozilla.org/ru/firefox/fx/**
- Chrome, **www.google.com/chrome?hl=ru**
- Opera, **ru.opera.com**
- Safari (только OS X), **www.apple.com/ru/safari**

Сетевые инструменты

- WS_FTP, CuteFTP, AceFTP и другие инструменты для Windows доступны на ресурсе: **www.softodrom.ru**
- Transmit (OS X), **www.panic.com/transmit**
- Cyberduck (OS X), **cyberduck.ch**
- Fetch (OS X), **fetchsoftworks.com**
- Cygwin (эмулятор Linux для Windows), **cygwin.com**
- PuTTY (эмулятор терминала telnet/SSH), **www.chiark.greenend.org.uk/~sgtatham/putty**

к файлу, перемещения или копирования файлов и каталогов или управления программным обеспечением сервера.

Пользователи операционной системы Windows могут установить эмулятор Linux — программу под названием Cygwin — для доступа к командной строке. Существует также PuTTY, бесплатный клиент Telnet/SSH. Система OS X содержит встроенное приложение под названием Terminal, которое является полноценной терминальной программой, предоставляющей доступ к системе Unix, и позволяющей использовать протокол SSH для доступа посредством командной строки к другим системам в Интернете.

УПРАЖНЕНИЕ 1.1. РЕЗЮМЕ

Теперь, когда вы делаете первый шаг в изучении веб-дизайна, наступает момент, чтобы подытожить ваши средства и цели. Используя материалы в этой главе как общее руководство, попытайтесь лаконично ответить на следующие вопросы:

- Какова ваша цель? Стать профессиональным веб-дизайнером? Делать только персональные веб-сайты?
- Какие аспекты веб-дизайна интересуют вас больше всего?
- Какие из имеющихся у вас навыков будут полезны при создании веб-страниц?
- Какие навыки вам нужно освежить в памяти?
- Какие инструменты из числа аппаратных средств и программного обеспечения для веб-дизайна у вас уже есть?
- Какие инструменты вы должны приобрести? Какие инструменты вы хотели бы приобрести со временем?

Резюме

Мораль этой главы: вам не нужно изучать все. И даже если вы хотите узнать в конечном счете все, вы не должны изучать все сразу. Поэтому расслабьтесь и не беспокойтесь. Другая хорошая новость — в то время как существует множество профессиональных и дорогих инструментов, можно создать, разместить и открыть для доступа простой веб-сайт без особых затрат, используя бесплатные или недорогие программы и уже имеющийся компьютер.

Скоро вы убедитесь, насколько легко приступить к созданию веб-страниц — вы научитесь этому, когда прочтете книгу, которую держите в руках. И с того момента вы можете продолжить обучение и накопление опыта и найти свою специфическую нишу в сфере веб-дизайна.

ГЛАВА 2

КАК РАБОТАЕТ ВСЕМИРНАЯ ПАУТИНА

Я начала заниматься веб-дизайном в начале 1993 года — практически одновременно с рождением самой Всемирной паутины. По ее меркам я «старикашка», но все же прошло не так много времени, чтобы забыть момент, когда я впервые взглянула на веб-страницу. Тогда было трудно сказать, откуда появлялась информация и как все это работало.

Данная глава содержит исчерпывающую информацию о функционировании Всемирной паутины и познакомит вас с некоторыми основными терминами. Мы начнем с общей картины и доберемся до специфических особенностей.

В этой главе

- Отношение Всемирной паутины к Интернету
- Роль сервера
- Роль браузера
- Понятие URL-адреса и его компонентов
- Анатомия веб-страницы

Интернет против Всемирной паутины

Нет, это не сражение, это только возможность указать на различие между этими двумя словами, которые все чаще и чаще используются как синонимы.

Всемирная паутина — это подмножество Интернета. Это просто один из многих способов передачи информации по объединенным в сеть компьютерам.

Интернет — это сеть связанных компьютеров. Он не принадлежит какой-либо компании; это совместно работающие компьютеры, связь между которыми управляется системой стандартов и правил. Цель связи компьютеров состоит в совместном использовании информации. Существует много вариантов передачи данных между компьютерами, включая электронную почту, протокол FTP и множество других специализированных способов, на основе которых функционирует Интернет. Эти стандартизированные методы переноса данных по сети известны как *протоколы*.

Всемирная паутина (кратко называемая *веб*) является только одним из путей, каким можно получить доступ к информации через Интернет. Этот способ уникален тем, что позволяет документам быть связанными друг с другом посредством *гипертекстовых* ссылок. Таким образом формируется огромная информационная «сеть*». Всемирная паутина использует протокол, названный *HTTP* (*Протокол передачи гипертек-*

* То есть веб.

ста). Если вы уже пытались разобраться в принципах функционирования Всемирной паутины, данная аббревиатура должна выглядеть для вас знакомой, потому что это первые четыре буквы почти всех адресов веб-сайтов, что мы обсудим в следующем разделе.

Обслуживание вашей информации

Давайте поговорим о компьютерах, которые составляют Интернет. Поскольку они «выдают» документы по запросу, эти компьютеры известны как *серверы*. Точнее, сервер — это не сам компьютер, а программное обеспечение, которое позволяет ему взаимодействовать с другими подобными устройствами. Однако слово «сервер» также используется, когда говорят о компьютере. Роль серверного программного обеспечения в том, чтобы ожидать запрос информации, а при получении такового разыскать и отослать эту информацию обратно как можно быстрее.

Для Всемирной паутины нет особых различий, с кем иметь дело — с мощным сервером под управлением системы Unix или скромным персональным компьютером. Именно серверное программное обеспечение играет главную роль. Чтобы компьютер стал частью Всемирной паутины, на нем должно выполняться специальное ПО, позволяющее обрабатывать транзакции протокола передачи гипертекста (Hypertext Transfer Protocol). Веб-серверы также называют «HTTP-серверами».

Серверное программное обеспечение различно, но чаще всего используется Apache HTTP-сервер (программное обеспечение *с открытым исходным кодом*) и Internet Information Services (IIS) корпорации Microsoft. Программное обеспечение Apache свободно распространяется и доступно для ориентированных на Unix компьютеров, а также предустановлено на компьютерах Mac, работающих под управлением системы OS X. Кроме того, существует версия для операционной системы Windows. IIS — часть семейства решений для серверов корпорации Microsoft.

Каждому компьютеру и устройству (модему, маршрутизатору, смартфону, автомобилю и т. д.) в Интернете назначается уникальный числовой *IP-адрес* (Internet Protocol — *межсетевой протокол*). Например компьютер, на котором размещается сайт *yandex.ru*, имеет IP-адрес 93.158.134.3. Все эти числа могут вызвать головокружение. К счастью, существует *система доменных имен* (*DNS*, Domain Name System), которая позволяет нам обращаться к этому серверу также по его *доменному имени* yandex.ru. Числовой IP-адрес используется компьютерами, в то время как доменное имя более понятно людям. Сопоставление текстовых доменных имен с их соответствующими числовыми IP-адресами — задача отдельного *DNS-сервера*.

Сконфигурировать ваш веб-сервер можно так, чтобы с единственным IP-адресом сопоставлялось более одного доменного имени, позволяя нескольким сайтам совместно использовать один и тот же сервер.

ТЕРМИНОЛОГИЯ

Открытый исходный код

Программное обеспечение с открытым исходным кодом разрабатывается совместными усилиями таким образом, чтобы другие программисты могли использовать и модифицировать этот код. Такие программы обычно бесплатны.

Больше никаких IP-адресов

Администрация адресного пространства Интернет (Internet Assigned Numbers Authority, *IANA*) — организация, которая присваивает IP-номера — передала последний набор IP-адресов 3 февраля 2011 года. Все верно, больше нет свободных IP-адресов в формате `###.###.###.###`. Этот формат IP-адреса (так называемый *IPv4*) способен производить 4,3 миллиарда уникальных адресов, что казалось огромным количеством в 1977 году, когда задумывался «эксперимент» с Интернетом. Его создатели никак не могли предвидеть, что однажды каждому телефону, телевизору и предмету на полке в магазине потребуется собственный адрес.

Решением представляется новый формат IP-адресов (*IPv6*, пока находится в разработке), позволяющий создать триллионы и триллионы уникальных IP-номеров, но есть небольшая загвоздка — он несовместим с текущей сетью, работающей на основе адресов в формате IPv4. Так что IPv6 будет действовать параллельно тому Интернету, который есть сегодня. В конце концов использование адресов IPv4 прекратится, но некоторые говорят, что на это уйдут десятилетия.

Несколько слов о браузерах

Теперь вы знаете, что сервер выполняет обслуживание, но как насчет другой стороны? Программное обеспечение, которое формирует и отправляет запросы, называют *клиентом*. Люди используют браузеры настольных компьютеров, мобильных устройств и другие вспомогательные технологии (например, программы экранного доступа) в качестве клиентов для получения доступа к информации во Всемирной паутине. Сервер возвращает документы браузеру (который в технических кругах называют *пользовательским агентом*) для отображения.

Запросы и ответы обрабатываются протоколом HTTP, упомянутым ранее. Хотя мы говорили о «документах», протокол HTTP может использоваться для передачи изображений, видеороликов, звуковых файлов, данных, сценариев и всех других ресурсов, из которых обычно состоят веб-сайты и приложения.

Когда мы думаем о браузере, то обычно представляем окно программы на компьютерном мониторе, с веб-страницей, показанной в этом окне. Такие программы известны как графические, или настольные, браузеры, и в течение долгого времени они были единственными средствами просмотра ресурсов Всемирной паутины. К числу самых популярных браузеров для настольных компьютеров на момент написания этой книги относились Internet Explorer для операционной системы Windows, Chrome, Firefox, и Safari (OS X), а также Opera. Однако сегодня все больше и больше людей выходят в Интернет в дороге, используя браузеры в смартфонах и планшетах.

Краткая история Всемирной паутины

Всемирная паутина зародилась в Европейской организации по ядерным исследованиям (ЦЕРН) в Женеве (Швейцария) в 1989 году. Программист по имени Тим Бернерс-Ли впервые предложил систему управления информацией, которая использовала «гипертекстовый» процесс для объединения связанных документов по сети. Он и его партнер, Роберт Кайо, создали прототип проекта и выпустили его для презентации. В первые несколько лет веб-страницы были только текстовыми. Трудно поверить, но в 1992 году (не так давно) в мире было всего 50 веб-серверов.

Реальный рост популярности Всемирной паутины произошел в 1993 году, когда был представлен первый графический браузер (NCSA Mosaic). Это позволило Всемирной паутине превратиться из научного исследования в средство массовой информации. Развитие Всемирной паутины продолжается под наблюдением Консорциума Всемирной паутины (W3C).

Если вы хотите глубже окунуться в историю, изучите эти сайты:

Материалы из Википедии:

ru.wikipedia.org/wiki/Интернет

ru.wikipedia.org/wiki/Всемирная_паутина

Исторические архивы консорциума Всемирной паутины:

www.w3.org/History.html

ТЕРМИНОЛОГИЯ

Сторона сервера и сторона клиента

Часто в веб-дизайне вы услышите упоминание о приложениях на *стороне клиента* или на *стороне сервера*. Эти термины используются, чтобы указать, какой компьютер выполняет обработку. Приложения на стороне клиента запускаются на компьютере пользователя, в то время как приложения и операции на стороне сервера используют ресурсы серверного компьютера.

Интранет и экстранет

Думая о веб-сайте, вы предполагаете, что он доступен любому пользователю, занимающемуся веб-серфингом. Однако многим организациям выгоднее использовать совместный доступ к информации и возможности веб-сайтов для обмена внутрикорпоративными данными. Такие специальные веб-ориентированные сети называют *интранетом*. Они функционируют как обычные веб-сайты, только расположены на компьютерах со специальными устройствами безопасности (*брандмауэрами*), которые запрещают доступ компьютерам вне компании. Интранет имеет множество применений, например обеспечение совместного доступа к информации о персонале или предоставление доступа к базе данных учетных записей.

Экстранет похож на интранет, только он разрешает доступ к информации и аутентифицированным пользователям, находящимся за пределами организации. Например фирма-изготовитель может предоставить своим клиентам пароли, которые позволяют им проверять состояние заказов в базе данных компании. Разумеется, пароли определяют уровни доступа к информации.

Также важно иметь в виду, что существуют альтернативные способы просмотра. Пользователи с ограничениями по зрению могут слушать веб-страницу, читаемую программой экранного доступа (или очень сильно масштабировать текст). Люди с нарушениями опорно-двигательной системы могут пользоваться вспомогательными устройствами для перехода по ссылкам и ввода текста. Создаваемые сайты должны быть доступны и удобочитаемыми для всех этих пользователей, независимо от выбранного ими способа просмотра.

Даже в современных графических браузерах, которые впервые показали нам огромный мир Всемирной паутины, веб-страницы могут выглядеть и функционировать по-разному. Это происходит из-за различий в поддержке веб-технологий и возможности пользователей изменять настройки просмотра.

Адреса веб-страниц (URL)

У каждого документа во Всемирной паутине есть свой собственный специальный адрес, именуемый *URL* (Uniform Resource Locator — Унифицированный локатор ресурса). Практически каждый день мы видим URL-адреса (произносится как «ю-эр-эл») на рекламных уличных щитах, визитных карточках или в телевизионной рекламе. Веб-адреса полностью интегрированы в современный язык.

Некоторые URL-адреса короткие и благозвучные. Другие могут быть похожи на не подвластные памяти последовательности символов, разделенные точками и слешами, но каждая часть имеет определенную цель. Сейчас мы рассмотрим их.

Состав URL-адреса

Полный URL-адрес состоит из трех частей: протокола, имени сайта и абсолютного пути к файлу или ресурсу, как показано на рис. 2.1.

Рис. 2.1. Части URL-адреса

❶ `http://`

Первая задача URL-адреса — определить протокол для текущей специфической транзакции. Буквы HTTP сообщают серверу, что нужно использовать протокол передачи гипертекста, или перейти в «веб-режим».

❷ `www.example.com`

Следующая часть URL-адреса идентифицирует веб-сайт по его доменному имени. В этом примере доменное имя — example.com. Часть «www.» в начале — особое имя хоста в этом домене. Имя хоста «www» стало соглашением, но не является правилом. Фактически, иногда имя хоста может быть опущено. В домене допустимо присутствие нескольких веб-сайтов (иногда называемых поддоменами или субдоменами), например, ***development.example.com***, ***clients.example.com*** и так далее.

❸ `/2013/example/webpage.html`

Это абсолютный путь через каталоги на сервере к запрашиваемому документу HTML, *webpage.html*. Слова, разделенные слешами (косыми чертами) — это имена каталогов, начиная с корневого каталога хоста (обозначенного первым слешем — /). Поскольку Интернет первоначально содержал компьютеры, управляемые операционной системой Unix, наш текущий порядок действий все еще следует многим ее правилам и соглашениям. Отсюда и взят символ /, разделяющий имена каталогов.

Подведем итоги: приведенный в качестве примера URL-адрес сообщает, что требуется использовать протокол HTTP, чтобы соединиться в Интернете с веб-сервером с именем ***www.example.com***, и запросить документ *webpage.html* (расположенный в каталоге *example*, который в свою очередь находится в каталоге *2013*).

Почему в этом URL-адресе нет http://?

Поскольку почти все веб-страницы используют протокол передачи гипертекста, часть `http://` часто только подразумевается. Таким образом, когда адреса сайтов рекламируются в печати или по телевидению, это способ сохранить URL-адрес коротким и благозвучным. Кроме того, браузеры запрограммированы добавлять часть `http://` автоматически, избавляя вас от лишних нажатий клавиш. Может показаться, что мы опускаем `http://`, но на самом деле он отсылается на сервер незаметно для нас.

Когда мы начнем использовать URL-адреса для создания гиперссылок в документах HTML в главе 6, вы узнаете, что необходимо указывать протокол при создании ссылки на веб-страницу на другом сервере.

ПРИМЕЧАНИЕ

Иногда вам будут встречаться URL-адреса, которые начинаются с части `https://`. Это признак зашифрованной транзакции с сервером. У защищенных серверов есть специальные устройства шифрования, которые скрывают определенный контент, такой как номера кредитных карт, в процессе передачи данных серверу и обратно. Обратите на это внимание, когда в следующий раз будете совершать онлайн-покупки.

Файлы по умолчанию

Очевидно, не каждый URL-адрес настолько длинный. Многие адреса не включают имени файла, а просто указывают на каталог, например:

`http://www.eksmo.ru`

`http://www.jendesign.com/resume/`

Когда сервер получает запрос имени каталога, а не определенного файла, он ищет в нем документ, заданный по умолчанию, обычно с именем *index.html*. Поэтому, если ввести вышеупомянутые URL в адресную строку браузера, в итоге вы увидите следующие документы:

```
http://www.eksmo.ru/index.html
```

```
http://www.jendesign.com/resume/index.html
```

Имя файла, заданного по умолчанию (также называемого *индексным файлом*), может меняться и зависит от того, как сконфигурирован сервер. В этих примерах он носит имя *index.html*, но некоторые серверы предпочитают имя файла *default.htm*. Если ваш сайт использует на стороне сервера программное обеспечение для генерации страниц, индексному файлу можно присвоить имя *index.php* или *index.asp*. Но предварительно согласуйте это решение с администратором сервера, чтобы убедиться, что вы присваиваете индексному файлу надлежащее имя.

Также следует отметить, что в первом примере оригинальный URL-адрес не имеет заключительного слеша, указывающего, что это был каталог. Когда слеш опущен, сервер самостоятельно добавляет его, если находит каталог с таким именем.

Индексный файл также полезен в целях безопасности. Некоторые серверы (в зависимости от их конфигурации) возвращают содержимое каталога для отображения в браузере, если файл по умолчанию не найден. Рис. 2.2 демонстрирует, как в таком случае файлы каталога *housepics* отображены в браузере. Один из способов воспрепятствовать людям, шпионящим за вашими файлами, — убедиться, что в каждом каталоге есть индексный файл. Системный администратор может добавить и другие формы защиты, чтобы запретить просмотр ваших каталогов в браузере.

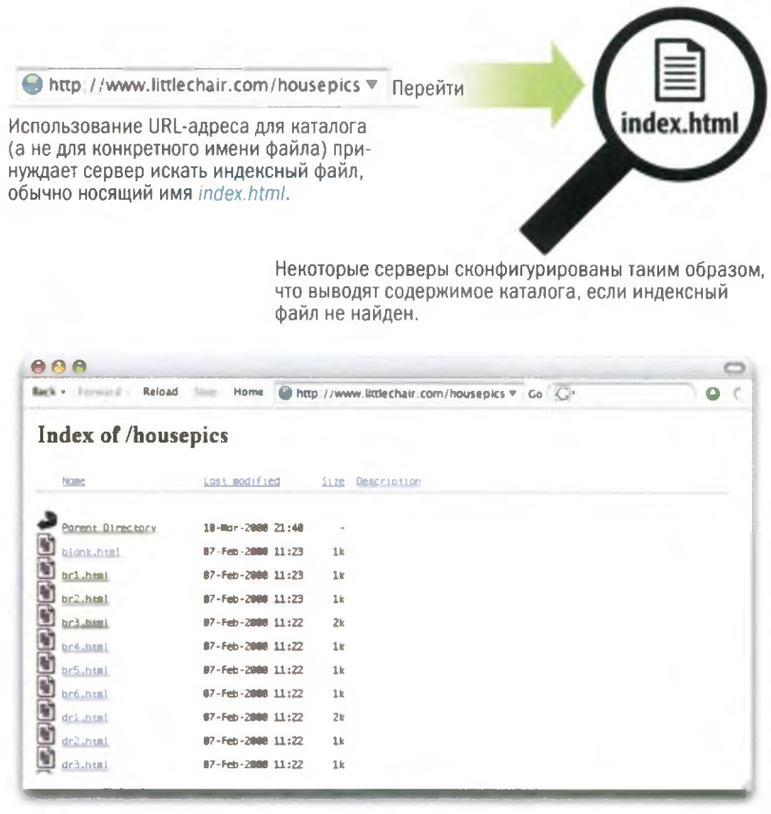


Рис. 2.2. *Некоторые серверы отображают содержимое каталога, если индексный файл не найден*

Анатомия веб-страницы

Мы все знакомы с тем, как веб-страницы выглядят в окне браузера, но что происходит «за кадром»?

В верхней части рис. 2.3 вы видите, как простая веб-страница отображается в браузере. Хотя ее можно рассматривать как единый документ, фактически она состоит из четырех отдельных файлов: HTML-документа (*index.html*), таблицы стилей (*kitchen.css*) и двух изображений (*foods.gif* и *spoon.gif*). Всеми компонентами управляет HTML-документ.

HTML-документы

Вы удивитесь, узнав, что графически насыщенные и интерактивные страницы, которые мы видим во Всемирной паутине, сгенерированы простыми текстовыми документами. Такой файл называют *исходным документом*.

Взгляните на *index.html*, исходный документ для веб-страницы «Кухня Кристины». Вы увидите, что он содержит текстовое содержимое страницы плюс специальные *теги* (обозначенные угловыми скобками, < и >), которые описывают каждый текстовый элемент на странице.

Добавление описывающих тегов к текстовому документу известно как *разметка* документа. Веб-страницы используют язык, названный *языком гипертекстовой разметки* (HyperText Markup Language), или коротко *HTML*, который был создан специально для документов с гипертекстовыми ссылками. Язык HTML определяет множество текстовых элементов, которые составляют документы, таких как заголовки, абзацы, подчеркнутый текст и, конечно, ссылки.

Есть также HTML-элементы, которые добавляют информацию о документе (например его название), а также изображения, видео, Flash-ролики и виджеты элементов формы, например для ввода имени пользователя.

Стоит вкратце отметить, что на самом деле на сегодняшний день существует несколько версий языка HTML. Наиболее устоявшимися являются версия HTML 4.01 и ее более строгий родственник XHTML 1.0. Возможно, вы слышали, какая шумиха поднялась во Всемирной паутине с появлением новой версии HTML5, предназначенной для лучшей обработки веб-приложений и постепенно обретающей поддержку браузеров. Я подробно расскажу обо всех версиях и их различиях в главе 10. А пока нам нужно рассмотреть основные элементы, которые применимы вне зависимости от выбранной вами версии языка HTML.

УПРАЖНЕНИЕ 2.1. ПРОСМОТР ИСХОДНОГО КОДА

Вы можете просмотреть код любого HTML-файла веб-страницы, выбрав команду **Вид** ⇒ **Исходный код страницы** (View ⇒ Page Source) или **Вид** ⇒ **Источник** (View ⇒ Source) в меню вашего браузера. Тот откроет исходный код документа в отдельном окне. Давайте заглянем во внутреннее устройство веб-страницы.

1. На диске, прилагающемся к книге, найдите в папке *Примеры\глава-02* файл *index.html* и откройте его в браузере. Вы должны увидеть веб-страницу, показанную на рис. 2.3.
2. Выберите команду **Вид** ⇒ **Исходный код страницы** (View ⇒ Page Source) или **Вид** ⇒ **Источник** (View ⇒ Source) в меню вашего браузера. В браузерах Chrome и Opera команда **Исходный код** (View Source) находится в меню **Средства разработки** (Developer). В браузере Internet Explorer команда выглядит следующим образом: **Вид** ⇒ **Просмотреть HTML-код** (View ⇒ View HTML). (Если строка меню в вашем браузере не отображается, нажмите клавишу **Alt**.) Откроется окно, демонстрирующее исходный код, показанный на рис. 2.3.
3. Исходный код для большинства сайтов значительно более сложный. Рассмотрите исходный код сайта **rambler.ru** или любого другого. Не волнуйтесь, если вы не разберетесь в коде. Большая его часть станет понятна к тому времени, когда вы закончите читать эту книгу.

Предупреждение. Имейте в виду, что, хотя изучение кода чужих страниц полезно, кража кода для собственных проектов — дурной тон (и даже незаконна!). Если вы хотите использовать код в том виде, в котором вы его видите на чужой странице, спросите разрешения у владельца и всегда ссылайтесь на него.

Быстрое знакомство с HTML-разметкой

Мы рассмотрим язык HTML более подробно в части II, так как я не хочу нагружать вас большим количеством деталей прямо сейчас. Но есть несколько моментов, на которые следует внимание. Они касаются того, как функционирует HTML-код и как с ним обращаются браузеры.

Просмотрите HTML-код на рис. 2.3 и сравните его с результатом в браузере. Легко заметить, как элементы, помеченные HTML-тегами в исходном коде, соответствуют тому, что отображается в окне браузера.

Во-первых, вы заметите, что текст внутри скобок (например, **<body>**) не отображается на странице в браузере. Показано только содержимое элемента, а разметка скрыта. *Теги* указывают на имя HTML-элемента — обычно сокращенное, такое как *h1* для *heading level 1* (заголовок 1 уровня) или *em* для *emphasized text* (акцентированный текст).

Во-вторых, вы увидите, что большинство HTML-тегов идут парами, окружая содержимое элемента. В нашем HTML-документе **<h1>** говорит о том, что следующий текст должен быть отформатирован как заголовок уровня 1; **</h1>** обозначает на конец заголовка. У некоторых элементов, именуемых *пустыми*, нет содержимого. В нашем примере тег **<hr>** указывает на пустой элемент, который предписывает браузеру «создать в этой позиции горизонтальную линию».

Поскольку я не была знакома с программированием, когда впервые начала писать HTML-код, мне помогала мысль о тегах и тексте, как об «узелках на веревочке», которые браузер обрабатывает последовательно, один за другим.

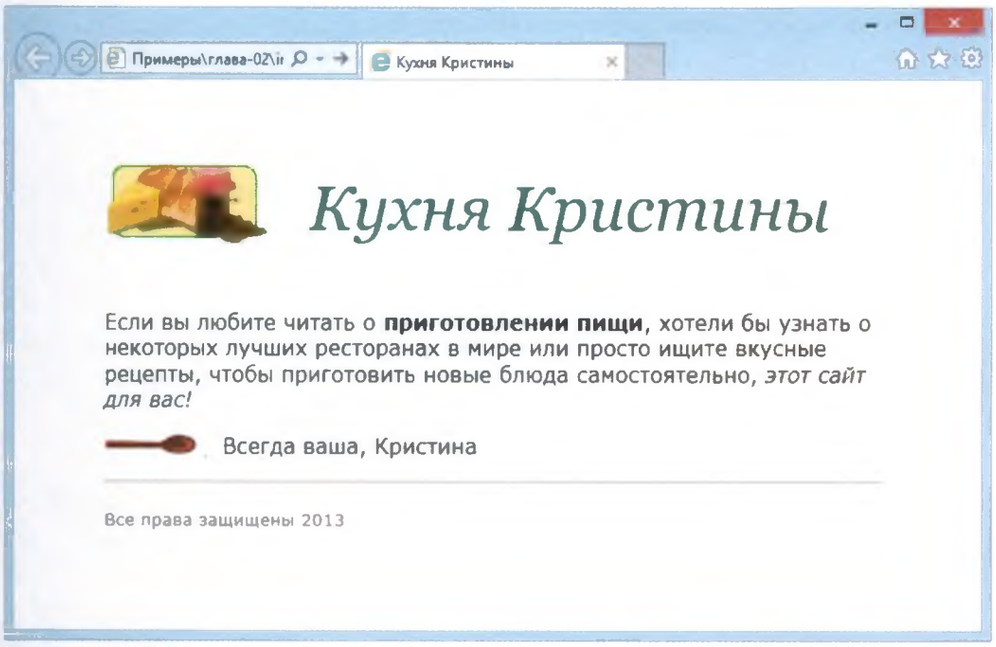
Например, когда браузер сталкивается с открытой скобкой (**<**), он считает все следующие символы частью разметки, пока не находит закрывающую скобку (**>**). Точно так же он считает весь контент после открытия тега **<h1>** заголовком, пока не сталкивается с закрывающим тегом **</h1>**. Это способ, которым браузер *анализирует* HTML-документ. Понимание принципов работы браузера с HTML-кодом может быть полезно при анализе неправильного HTML-документа.

Но где изображения?

Как вы уже поняли, изображения не встраиваются в HTML-файл, но как они там оказываются, когда вы просматриваете веб-страницу?

На рис. 2.3 продемонстрировано, что каждое изображение — это отдельный графический файл.

Графика помещается в текст с помощью HTML-элемента изображения (**img**), который сообщает браузеру, где искать графический файл (согласно URL-адресу). Когда браузер находит элемент **img**, он запраши-



Эта веб-страница на самом деле состоит из четырех отдельных файлов: текстового HTML-документа, таблицы стилей и двух рисунков. Теги в HTML-документе задают браузеру инструкции, как должен быть обработан текст и где следует разместить изображения.

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Кухня Кристины</title>
<link rel="stylesheet" href="kitchen.css" type="text/css" >
</head>

<body>
<h1><img src="foods.gif" alt="логотип с продуктами">Кухня Кристины</h1>

<p>Если вы любите читать о <strong>приготовлении пищи</strong>, хотели бы узнать о неко-
торых лучших ресторанах в мире или просто ищете вкусные рецепты, чтобы приготовить новые
блюда самостоятельно, <em>этот сайт для вас!</em></p>

<p><img src="spoon.gif" alt="изображение ложки">Всегда ваша, Кристина</p>
<hr>
<p><small>Все права защищены 2013</small></p>
</body>
</html>
```

kitchen.css

```
body { font: normal 1em Verdana; margin: 1em 10%;}
h1 { font: italic 3em Georgia; color: rgb(23, 109, 109); margin: 1em 0 1em;}
img { margin: 0 20px 0 0; }
h1 img { margin-bottom: -20px; }
small { color: #666666; }
```

foods.gif ***spoon.gif***

Рис. 2.3. *Исходный файл и изображения, которые составляют простую веб-страницу*

Добавление поведений с помощью JavaScript

В дополнение к *структуре* документа и *представлению*, существует также компонент *поведения*, который определяет, как все *работает*. Во Всемирной паутине поведение определяется языком сценариев, который называется *JavaScript*. Я затрону его в части V этой книги, однако для изучения языка JavaScript требуется больше информации, чем я могу предоставить в рамках данного издания.

Многие дизайнеры (включая меня) обращаются к людям, имеющим опыт написания сценариев, чтобы добавить сайтам функциональность. Однако знание языка JavaScript становится все более необходимым для профессии веб-дизайнера.

вает у сервера файл изображения и затем помещает его в поток контента. Программное обеспечение браузера собирает отдельные фрагменты в финальную страницу. Видеоконтент и прочие вложенные файлы добавляются тем же способом.

Сборка страницы происходит мгновенно, поэтому она появляется так, как будто вся страница загружается сразу. При медленных соединениях, или на более медленных компьютерах, или если страница содержит большое количество графики, процесс сборки более заметен, поскольку изображения появляются позже текста. Иногда странице даже требуется перезагрузка по мере появления новых изображений (хотя вы можете сконструировать свои страницы так, чтобы этому воспрепятствовать).

Добавление стилей

Я хочу обратить ваше внимание на последний ключевой компонент нашей простой страницы. В верхней части HTML-документа есть элемент `link`, который указывает на файл таблицы стилей *kitchen.css*, которая содержит информацию о том, как должна выглядеть страница в браузере. Это инструкции стилей, написанные в соответствии с правилами *каскадных таблиц стилей* (*CSS*). CSS позволяет дизайнерам добавлять визуальные инструкции стилей (известные как *представления* документа) к размеченному тексту (*структура* документа в терминологии веб-дизайнеров). В третьей части книги вы по-настоящему почувствуете мощь каскадных таблиц стилей.

На рис. 2.4 показана веб-страница сайта «Кухня Кристины» с примененными стилями и без них. Браузеры оснащены стилями, используемыми по умолчанию для каждого поддерживаемого ими HTML-элемента, так что если для HTML-документа нет собственных инструкций стилей, браузер применит свои (то, что вы видите на скриншоте слева). Всего несколько правил стилей способны значительно улучшить внешний вид страницы.

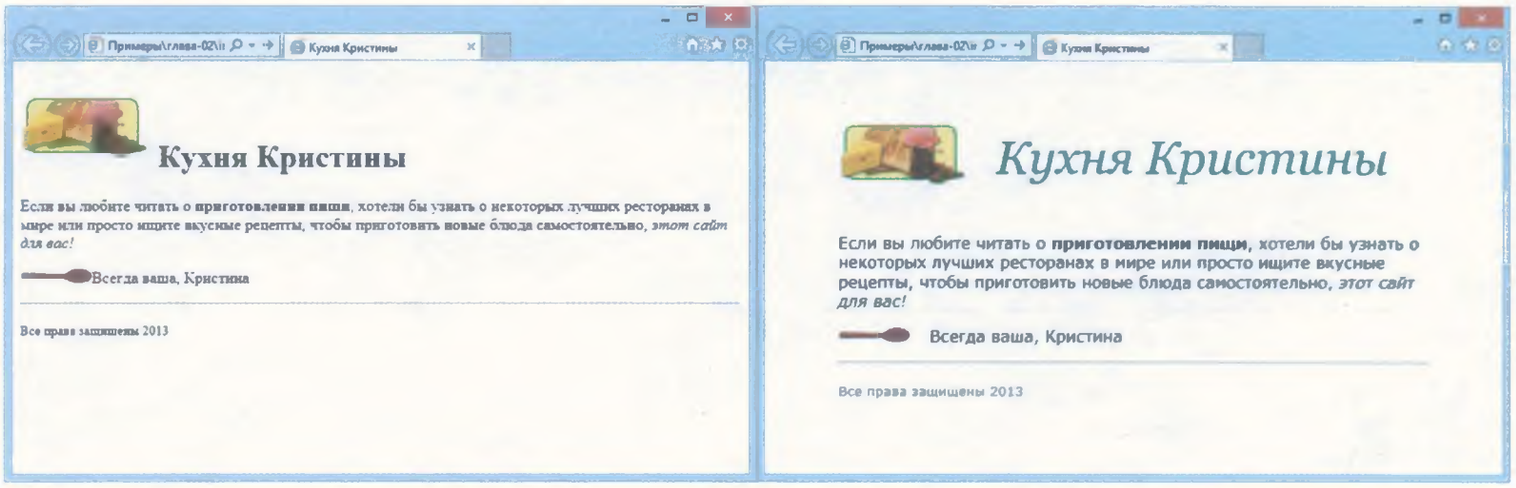


Рис. 2.4. Веб-страница сайта «Кухня Кристины» с примененными стилями (справа) и без них (слева)

Резюме

В завершение этой главы давайте рассмотрим события, происходящие с каждой веб-страницей, которая появляется на экране вашего компьютера (рис. 2.5).

1. Вы запрашиваете веб-страницу, указав ее URL-адрес (например, *http://jenskitchensite.com*) непосредственно в браузере или переходя

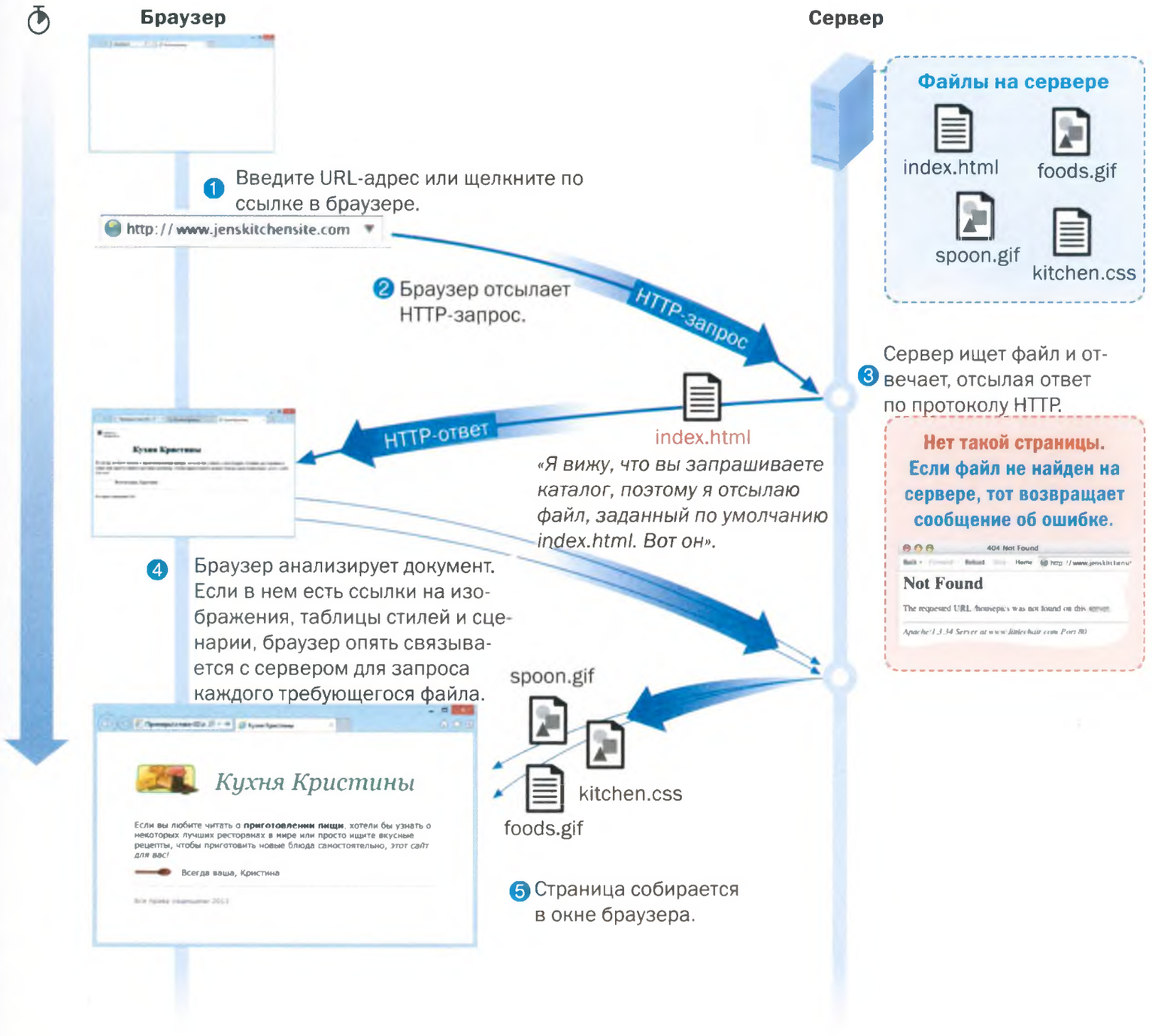


Рис. 2.5. Как браузеры отображают веб-страницы

по ссылке на странице. URL-адрес содержит всю информацию, необходимую, чтобы найти в Интернете определенный документ.

2. Ваш браузер посылает HTTP-запрос на сервер, указанный в URL-адресе, и запрашивает нужный файл. Если URL-адрес указывает не на файл, а на каталог, браузер запрашивает находящийся там индексный файл.

3. Сервер ищет требуемый документ и выдает HTTP-ответ.

   а. Если страница не может быть найдена, сервер возвращает сообщение об ошибке. Сообщение обычно следующее — «404. Нет такой страницы», хотя могут быть выведены и более гостеприимные сообщения об ошибках.

   б. Если документ найден, сервер извлекает требуемый файл и возвращает его браузеру.

4. Браузер разбирает HTML-документ. Если страница содержит изображения (обозначенные элементом `img`), браузер связывается с сервером снова, чтобы запросить каждый графический файл, указанный в разметке.

5. Браузер вставляет каждое изображение в позицию документа, обозначенную элементом `img`. И *все*! Собранная веб-страница отображается на вашем экране.

ГЛАВА 3

ВАЖНЫЕ КОНЦЕПЦИИ ДЛЯ ВЕБ-ДИЗАЙНЕРА

По мере того как Всемирная паутина развивается и количество устройств, с которых мы можем посещать ее ресурсы, экспоненциально увеличивается, работа веб-дизайнеров и разработчиков значительно усложняется. Честно говоря, мне не хватит книги, чтобы изложить все происходящее. В последующих главах я остановлюсь на краеугольных камнях веб-дизайна — HTML-элементах, стилях CSS, обзоре JavaScript и подготовке веб-графики — знание которых обеспечит прочную основу для дальнейшего развития ваших навыков.

Но прежде чем начать с азов, я хочу ввести несколько важных понятий, которые, по-моему, должен знать каждый веб-дизайнер. Мы рассмотрим идеи и соображения, которые вдохновляют наши решения и вносят вклад в современную среду веб-дизайна. Я часто буду употреблять термины, поясняемые ниже.

Суть в том, что веб-дизайнер никогда не знает точно, как будут просматриваться создаваемые им страницы. Неизвестно, какие из сотен браузеров будут использованы, запустят их на рабочем компьютере или на каком-нибудь портативном устройстве, насколько велико будет окно браузера, какие шрифты установлены, включены ли дополнительные функции, такие как JavaScript, какова скорость соединения с Интернетом, будут ли страницы считываться с помощью программ экранного доступа и так далее. Важные понятия этой главы в первую очередь обозначают методы борьбы с неизбежным элементом неизвестности в нашей среде. К ним относятся:

- Множество устройств
- Веб-стандарты
- Прогрессивное улучшение
- Адаптивный веб-дизайн
- Доступность
- Производительность сайта

Поскольку мы только начали, я буду приводить краткие и нетехнические описания. Моя цель — дать вам базовое понимание того, что я подразумеваю под такими терминами, как «прогрессивное улучше-

В этой главе

Wi-Fi соединением. А на новых планшетах iPad с экранами с высоким разрешением можно путешествовать по Всемирной паутине с помощью низкоскоростного 3G-соединения. Другими словами, все сложно!

Скоро люди будут чаще посещать веб-сайты с мобильных и альтернативных устройств, чем с настольного компьютера. Уже сейчас значительная часть пользователей использует смартфоны как единственное средство доступа в Интернет. Значит, важно оценить ситуацию правильно! Но, честно говоря, на момент написания книги я не совсем понимала, как уместить весь контент, который мы привыкли видеть на своем рабочем столе, в портативные устройства так, чтобы использовать его было по-прежнему приятно. Веб-дизайнеры достигли больших успехов в понимании проблемы и выработали замечательный дух сотрудничества, но дело в том, что наши инструменты и технологии не совсем подходят для этой задачи, и понадобится некоторое время, чтобы наверстать упущенное.

Мобильная Всемирная паутина?

Вы можете услышать, как дизайнеры используют термин «мобильный Интернет» или «мобильная Всемирная паутина», но на самом деле (как написал Стивен Хэй в своем Twitter-аккаунте в 2011 году, см. рис. 3.2) мобильной Всемирная паутина является не более чем настольной или планшетной и т. д. Существует только Всемирная паутина, доступ к которой можно получить со всевозможных устройств. На момент написания книги термин «мобильная Всемирная паутина» использовался как некое всеобъемлющее понятие для описания усилий по адаптации наших навыков дизайна, чтобы охватить большее количество всевозможных случаев. И как выясняется, расколоть этот крепкий орешек можно разными способами.

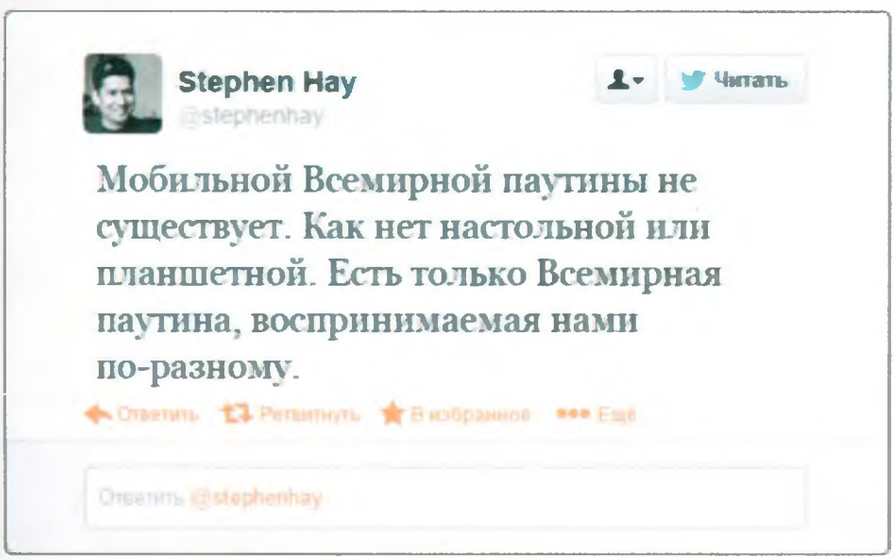


Рис. 3.2. Перевод Twitter-сообщения Стивена Хэя по поводу мобильной Всемирной паутины

Я хочу, чтобы вы уяснили: дизайн, каким вы его видите, работая над ним на своем настольном компьютере, не всеми будет восприниматься одинаково. Об этом факте должен помнить каждый специалист по веб-дизайну.

Дополнительные источники информации

- Статья «Тренды мобильного интернета» позволит вам быть в курсе основных тенденций, характерных для развивающейся мобильной Всемирной паутины (**seo-box.ru/news-30.html**).
- Книга «Сначала мобильные» Люка Вроблевски (Манн, Иванов и Фербер). Люк раньше других начал настаивать, что веб-сайты должны прекрасно функционировать на мобильных устройствах, и он делится своей точкой зрения в этой небольшой книге, набитой идеями.
- Статья **www.cisco.com/web/RU/news/releases/txt/2011/020211b.html** рассматривает тенденции развития мобильной Всемирной паутины до 2015 года.

Соблюдение стандартов

Следование веб-стандартам — основной инструмент для обеспечения максимальной согласованности вашего сайта.

Так как же справиться с этим разнообразием? Для начала хороший вариант — соблюдать стандарты HTML, CSS, JavaScript, задокументированные консорциумом Всемирной паутины. Соблюдение веб-стандартов поможет обеспечить единообразное представление вашего сайта во всех браузерах, которые их придерживаются (около 99% браузеров используемых в настоящее время). Также не помешает сделать контент заранее совместимым со стандартами, поскольку веб-технологии и возможности браузеров развиваются. Еще одно преимущество состоит в том, что клиенты придерживаются более высокого мнения о дизайнерах, создающих сайты в соответствии со стандартами.

Понятие соответствия стандартам может показаться очевидным, но в прошлом все, в том числе производители браузеров, очень свободно обращались с HTML-разметкой и сценариями. Мы заплатили за это необходимостью создавать сайты дважды, чтобы заставить их работать во всех браузерах. Я еще расскажу о веб-стандартах далее, поэтому сейчас не буду вдаваться в подробности. Достаточно сказать, что веб-стандарты — ваши друзья. Все, что вы узнаете в этой книге, поможет вам двигаться в правильном направлении.

Дополнительные источники информации

Авторитетное руководство по соблюдению стандартов, объясняющее, насколько это логично для бизнеса, — книга «Веб-дизайн по стандартам» Джеффри Зельдмана («НТ Пресс»).

Прогрессивное улучшение

Прогрессивное улучшение — это стратегия, позволяющая справиться с неизвестными настройками браузера.

Со множеством браузеров появляется множество уровней поддержки веб-стандартов. На самом деле, ни один браузер не соблюдает все стандарты на 100%, и всегда есть новые технологии, медленно набирающие обороты.

Кроме того, пользователи могут задавать собственные настройки, например отключать поддержку JavaScript. Суть в том, что мы имеем дело с широким спектром возможностей браузера, от базовой поддержки только HTML-разметки до всех дополнительных функций.

*Прогрессивное улучшение** — это одна из стратегий, помогающая справиться с неизвестными настройками браузера. При создании дизайна с прогрессивным улучшением вы начинаете с базового варианта, при котором контент и функциональность доступны даже для загрузки в самых урезанных версиях браузеров и вспомогательных устройств. Далее вы наслаиваете более современные функции для браузеров, которые могут их воспроизвести. В конце можно добавить некоторые красивые, но необязательные эффекты, такие как анимация или скругленные углы рамок, чтобы улучшить опыт взаимодействия с сайтом для пользователей с самыми передовыми браузерами.

Прогрессивное улучшение — это подход, который оживляет все аспекты дизайна и проектирования страницы, в том числе HTML, CSS и JavaScript.

Стратегия верстки

Когда HTML-документ записан в логическом порядке и все элементы имеют значимую разметку, его можно использовать в самых разнообразных средах просмотра, в том числе очень старых и еще только создаваемых браузерах, на мобильных и вспомогательных устройствах. Возможно, внешний вид документа будет отличаться, но важно то, что контент останется доступен. Это также гарантирует, что поисковые системы, такие как Google, правильно каталогизируют контент. Четкий HTML-документ и его точно и подробно описанные элементы являются основой доступности.

ПРИМЕЧАНИЕ

Прогрессивное улучшение является оборотной стороной более старого подхода к работе с разнообразными браузерами, называемого *отказоустойчивостью***, при котором вы сначала проектируете максимально улучшенный опыт взаимодействия, а потом создаете серию урезанных вариантов для браузеров, не поддерживающих полную версию.

Стратегия стилизации

Различного опыта взаимодействия с сайтом можно добиться, просто воспользовавшись тем, как браузеры анализируют правила таблиц стилей. Не вдаваясь в излишние технические подробности, можно написать правило, которое сделает элемент фона красным, но также будет содержать стиль, добавляющий к нему красивый градиент (переход от одного цвета к другому) для браузеров, которые знают, как создавать градиенты.

Или вы можете использовать селекторы CSS для отображения определенных стилей только в последних версиях браузеров. Зная, что браузеры просто игнорируют те свойства и правила, которые не понимают,

* Англ. термин — progressive enhancement (примеч. ред.)
** Англ. термин — graceful degradation (примеч. ред.)

вы можете смело использовать инновации, не ухудшая при этом работу старых браузеров. Просто необходимо помнить, что сначала следует прописывать базовые правила стилей, а затем добавлять улучшения, как только будут выполнены минимальные требования.

Стратегия сценариев

JavaScript — это язык сценариев, делающий веб-страницы интерактивными и динамическими (обновление контента «на лету» или в ответ на ввод пользователя). Без него Всемирная паутина во многом походила бы на статичные буклеты. Как и в случаях с другими веб-технологиями, браузеры по-разному обрабатывают сценарии JavaScript (особенно на устройствах, отличных от настольного компьютера), и некоторые пользователи предпочитают совсем отключать поддержку JavaScript. Первое правило прогрессивного улучшения — убедиться, что основные функции, такие как ссылки со страницы на страницу или выполнение важных задач (например, предоставление данных с помощью форм) работают, даже если JavaScript выключен. Так вы обеспечиваете базовый опыт взаимодействия и улучшаете его, когда язык JavaScript доступен.

Дополнительные источники информации

- Статья «Концепция прогрессивного улучшения» (**designformasters.info/posts/understanding-progressive-enhancement/**).
- Статья «Progressive Enhancement» (**serenity.su/blog/post/33636043528/progressive-enhancement**).
- Статья «Graceful Degradation» (**serenity.su/blog/post/3026847774**).

Адаптивный веб-дизайн

По умолчанию большинство браузеров на небольших устройствах, таких как смартфоны и планшеты, сжимают веб-страницу до размеров экрана и предоставляют механизмы для масштабирования и перемещения по ней. Хотя технически все работает, это не очень удобно. Текст слишком мелкий, чтобы его прочесть, ссылки настолько малы, что по ним не попасть пальцем, а увеличение масштаба и панорамирование отвлекают.

Адаптивный веб-дизайн — это стратегия по предоставлению пользовательских макетов для устройств в зависимости от размера области просмотра (окна браузера). Хитрость адаптивного дизайна в том, что для всех устройств предоставляется один HTML-документ, но применяются разные таблицы стилей в зависимости от размера экрана, чтобы обеспечить наиболее оптимизированный макет для конкретного устройства. Например при просмотре страницы на смартфоне текст отображается в одной колонке с крупными ссылками для легкого нажатия.

Но когда та же самая страница просматривается в браузере на большом настольном компьютере, контент перегруппируется в несколько коло-

нок с традиционными элементами навигации. Как по волшебству! (Но на самом деле это просто каскадные таблицы стилей.)

Адаптивный веб-дизайн — стратегия для работы при неизвестных размерах экрана.

Сообщество веб-дизайнеров активно обсуждает адаптивный веб-дизайн с тех пор, как Итан Маркотт впервые написал ввел это понятие в статье «Responsive Web Design» (**www.alistapart.com/articles/responsive-web-design/**). Данный вид дизайна стал одним из основных инструментов, используемых при работе с неизвестным размером окна.

На рис. 3.3 показано несколько примеров адаптации сайта под размеры экрана монитора настольного компьютера, планшета и смартфона. Другие вдохновляющие примеры можно найти на сайте **mediaqueri.es**. Попробуйте открыть один из вариантов дизайна в своем браузере, а затем изменить размер окна на очень узкое или очень широкое и проанализировать, как при этом меняется макет.

Открытая библиотека изучения медицинской аппаратуры **www.omdrl.org**

Электронный журнал «Smashing Magazine» **smashingmagazine.org**

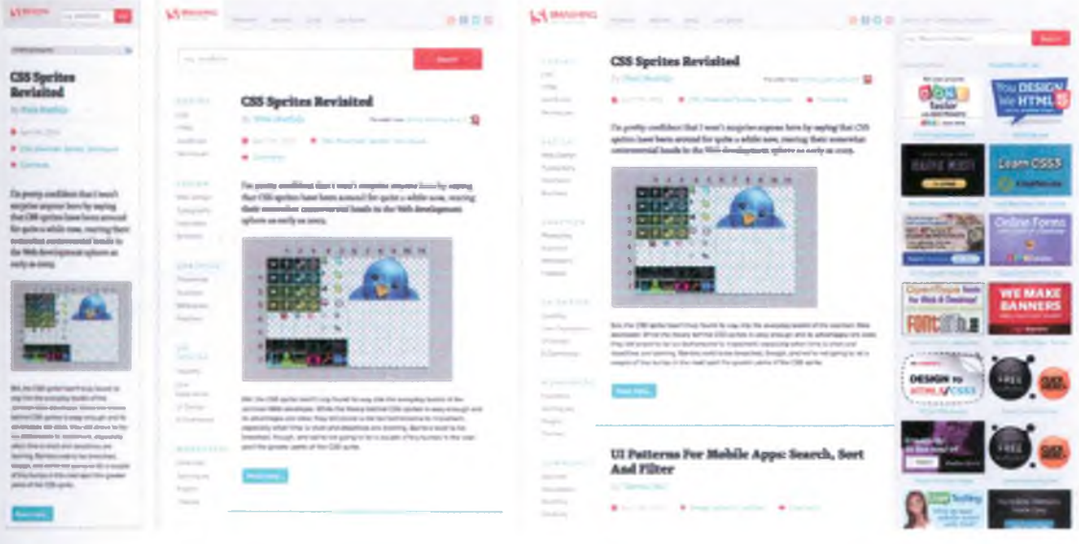


Рис. 3.3. *Адаптивные макеты сайтов меняются в зависимости от размера окна браузера*

Специализированные мобильные сайты

Альтернативой одному адаптивному сайту станет проектирование дополнительного (вдобавок к основному) сайта с уникальным URL-адресом, который будет запускаться с сервера в ответ на запрос, приходящий с мобильного устройства. В URL-адресах мобильных сайтов, как правило, указывается префикс `m` или `mobile`. Для некоторых типов сайтов отдельная мобильная версия — лучшее решение, особенно если вы знаете, что посетители, заходящие с мобильных устройств, придерживаются иных моделей просмотра сайта, чем пользователи настольных компьютеров. На специализированных мобильных сайтах наиболее часто запрашиваемые функции выделяются на первом экране, а многое из «лишней» информации с сайта для настольного браузера (например, котировки акций) просто отбрасывается.

На рис. 3.4 сравниваются основной и мобильный сайты компании Walgreens в том виде, в каком они появились в середине 2012 года. Обратите внимание, что пользователям мобильных телефонов предлагается гораздо более узкий спектр возможностей.

Специализированный мобильный сайт может стать лучшим способом облегчить выполнение сложных задач пользователям смартфонов. Дело в том, что адаптивный веб-дизайн не является универсальным решением. Для сайтов, контент которых главным образом текстовый, небольшой настройки макета может быть достаточно для удобного чтения на всех устройствах. Для других сайтов и веб-приложений предпочтительным может оказаться совершенно иной вид.

Недостатком специализированного мобильного сайта является то, что объем работы увеличится более чем в два раза. Потребуется дополнительное планирование контента, проектирование шаблонов, затраты времени на производство и текущее обслуживание. Но если это означает предоставление посетителям функций, которые им действительно нужны, сайт стоит вложений.

Рис. 3.4. Сравнение основного и специализированного мобильного сайтов

Адаптивный веб-дизайн помогает решить проблемы, связанные с макетом, но он не является панацеей от всех сложностей мобильного веб-дизайна. Чтобы обеспечить пользователям лучший опыт взаимодействия на выбранном ими устройстве, вам может потребоваться оптимизация, выходящая за пределы настройки внешнего вида сайта. Некоторые проблемы лучше решать, прибегая к помощи сервера для определения устройства и его возможностей, а затем принимать решения, что отправить обратно. Применяя прогрессивное улучшение, вы можете отобразить базовый вариант сайта во всех основных браузерах и на всех устройствах, и при этом создать расширенные параметры для тех устройств, которые смогут их использовать.

Для некоторых сайтов и сервисов, возможно, предпочтительнее создать отдельный мобильный сайт (см. врезку «Специализированные мобильные сайты») с настраиваемым интерфейсом и набором функций, использующим возможности телефона, например геолокацию.

При этом адаптивный дизайн хоть и не избавляет от всех проблем, все же является важной частью решения по формированию удовлетворительного опыта взаимодействия в широком круге браузеров.

Дополнительные источники информации

Я расскажу об адаптивном дизайне подробнее в главе 18, когда вы станете немного опытнее. Для дальнейшего изучения адаптивного дизайна я рекомендую следующие книги:

- Начинающим дизайнерам необходимо прочитать книгу Итана Маркотта «Отзывчивый веб-дизайн» (Манн, Иванов и Фербер). Это короткая книга, идеально подходящая для начала изучения принципов работы адаптивного веб-дизайна и того, как применить его самостоятельно.
- Книга «Сначала мобильные» Люка Вроблевски (Манн, Иванов и Фербер).

ПРИМЕЧАНИЕ

Даже специализированные мобильные сайты могут и должны выгодно применять адаптивные техники для настройки удобства работы с сайтом в зависимости от устройства. Не обязательно выбирать одно из двух решений.

Обеспечение доступности

Говоря об огромном количестве браузеров, используемых сегодня, до сих пор мы касались только визуальных, управляемых с помощью указателя мыши или касаниями (жестами) пальцев. Важно, однако, иметь в виду, что люди получают доступ ко Всемирной паутине различными путями — с помощью программ экранного доступа, ввода шрифтом Брайля, экранных луп, джойстиков, ножных педалей и так далее. Веб-дизайнеры должны так проектировать страницы, чтобы создавать как можно меньше барьеров на пути получения информации, независимо от возможности пользователей и устройств. Другими словами, вы должны создавать дизайн с учетом *доступности*.

Хотя описанные методы и стратегии предназначены для пользователей с ограниченными возможностями, такими как ухудшения зрения или нарушения опорно-двигательного аппарата, они могут быть также полезны другим пользователям с неудобными средствами веб-серфинга. Например получающим доступ с портативных устройств, или имеющим медленное коммутируемое подключение к Интернету, или отключившим изображения и JavaScript. Доступные сайты также более эффективно индексируются поисковыми системами, например Google. Дополнительные усилия, затраченные на создание доступного веб-сайта, будут вознаграждены.

Существуют четыре основные категории недостатков, которые влияют на то, как люди взаимодействуют с компьютерами и информацией в них.

Нарушения зрения. Люди с нарушениями зрения могут пользоваться вспомогательными устройствами, такими, как программы экранного доступа, дисплей Брайля или экранная лупа, чтобы прочитать текст на

Национальный стандарт Российской Федерации по доступности: ГОСТ Р 52872-2007

Следующие 10 кратких рекомендаций обобщают концепцию создания документов с расширенными возможностями доступа.

1. Графические файлы обязаны сопровождаться текстом, поясняющим изображение.
2. Таблицы не должны иметь большое количество вложений.
3. Веб-страницы не должны иметь фреймовую структуру.
4. При наличии гиперссылки необходимо текстовое описание объекта, на который она указывает.
5. Элементы форм веб-страниц обязаны сопровождаться текстовым описанием.
6. Часто посещаемые страницы по своему объему не должны превышать 2–3 экранов текста.
7. Число ссылок на странице должно быть не более 15.
8. Каждый графический файл следует снабжать поясняющим текстом в атрибуте `alt`.
9. При размещении Flash-элементов необходимо предусмотреть возможность перехода на страницу с аналогичной текстовой информацией.
10. Информация, представленная в файлах формата PDF в виде текста, должна корректно озвучиваться с помощью программ экранного доступа.

Полностью прочитать текст стандарта вы сможете, открыв документ GOST-52872-2007.pdf с диска, прилагающегося к книге.

экране. Они также могут использовать функцию масштабирования текста в браузере, чтобы сделать шрифт достаточно крупным для чтения.

Нарушения опорно-двигательного аппарата. Пользователи с нарушениями опорно-двигательного аппарата могут использовать для навигации во Всемирной паутине и ввода информации специальные устройства, такие как модифицированные мыши и клавиатуры, педали и джойстики.

Нарушения слуха. Пользователи с нарушениями слуха не будут слышать звуковое сопровождение мультимедийных файлов, поэтому необходимо предоставить им альтернативы, например транскрипции аудиофайлов или субтитры для видеороликов.

Когнитивные нарушения. Пользователям с нарушениями памяти, затруднениями в понимании прочитанного, сложностями при решении проблем и концентрации внимания удобно, когда сайты разработаны просто и понятно. Эти качества полезны любым пользователям вашего сайта.

В связи с необходимостью сделать Всемирную паутину удобной для всех, ее консорциум запустил программу «Инициатива по обеспечению веб-доступности» (WAI, Web Accessibility Initiative). Сайт WAI (**www.w3.org/WAI**) — отличная отправная точка для получения дополнительной информации о доступности веб-сайтов. Один из документов, подготовленных WAI, направлен на то, чтобы помочь разработчикам создавать доступные сайты — Руководство по обеспечению доступности веб-контента (Web Content Accessibility Web Content Accessibility Guidelines, WCAG). С ним можно ознакомиться по адресу **w3c.org.ru/wp-content/uploads/2011/10/русский-авторизованный-перевод-WCAG.pdf**. Правительство РФ выпустило национальный стандарт по

доступности (см. врезку «Национальный стандарт Российской Федерации по доступности: ГОСТ Р 52872-2007»). Всем сайтам выгодно придерживаться этих руководящих принципов, но если вы разрабатываете правительственный сайт, их соблюдение является обязательным.

Еще одна разработка консорциума Всемирной паутины — Инициатива по обеспечению веб-доступности многофункциональных веб-приложений (WAI-ARIA, Accessible Rich Internet Applications) — спецификация, в которой рассматривается доступность веб-приложений, содержащих динамически создаваемый контент, сценарии и расширенные элементы интерфейса, которые особенно сложны при использовании вспомогательных устройств.

Рекомендация ARIA описывает ряд ролей для контента и виджетов, которые авторы могут явно к ним применить с помощью атрибута `role`. Роли включают в себя меню, индикаторы процесса, ползунковые регуляторы, таймеры, подсказки, и т. п., а также добавляют расширенный слой семантики, если это необходимо. Чтобы ознакомиться с полным списком ролей, перейдите по адресу www.w3.org/TR/wai-aria/roles#role_definitions.

Дополнительные источники информации

Следующие ресурсы пригодятся для дальнейшего изучения доступности веб-сайтов:

- Ресурс Web Accessibility Initiative (WAI), www.w3.org/WAI
- Ресурс WebAIM, www.webaim.org
- Книга «Универсальные принципы дизайна» Уильяма Лидвелла, Кристины Холден и Джилл Батлер (Питер, 2012)

Производительность сайта

Количество пользователей, выходящих в Интернет через медленные коммутируемые соединения, сокращается (менее 10% в России на момент написания книги), а процент людей, использующих для выхода в Интернет мобильные устройства, быстро растет, и ожидается, что в конечном итоге их число превысит количество пользователей с настольными компьютерами и ноутбуками.

Более 70% пользователей в России получают доступ в Интернет посредством широкополосного подключения*, и эта цифра постоянно увеличивается. Доля пользователей Интернета с мобильными устройствами или по беспроводным каналам связи доходит до 46%**.

* По данным сервиса Руметрика, опубликованным на странице hostinfo.ru/articles/events/1561/

** net.compulenta.ru/727306/

Если у вас есть смартфон, то вы знаете, как раздражает ожидание полной загрузки страницы через мобильное соединение.

Однако производительность сайта важна независимо от того, каким способом пользователи получают к нему доступ. Исследование, проведенное корпорацией Google в 2009* году, показало, что увеличение времени загрузки страницы результатов поиска всего на 100–400 мс, привело к снижению числа поисков на (0,2–0,6%). Компания Amazon.com показала, что сокращение времени загрузки страницы всего на 100 мс привело к росту доходов на 1%**. Другие исследования показывают: пользователи ожидают, что загрузка сайта займет менее двух секунд, и почти треть аудитории уйдет с вашего сайта на другой, если загрузка длится дольше. Кроме того, эти люди вряд ли вернутся. Корпорация Google учитывает скорость загрузки сайта при ранжировании результатов поиска — если ваш сайт грузится медленно, вряд ли он окажется на желанной первой странице. Вывод: производительность сайта (вплоть до миллисекунды!) очень важна.

Существует множество приемов для улучшения производительности вашего сайта, и их можно разделить на две категории: ограничение размеров файлов и сокращение числа запросов, посылаемых на сервер. Приведенный ниже список — это лишь поверхностные меры по оптимизации сайта, но он даст вам общее представление о том, что можно сделать.

- Оптимизация изображений для получения минимального возможного размера файла без потери качества. О ней вы узнаете в главе 20.
- Минимизация размера документов HTML и CSS путем удаления лишних пробелов между символами и переводов строк.
- Сведение к минимуму использования сценариев JavaScript.
- Добавление сценариев таким образом, чтобы они загружались параллельно с другими активами страницы и не блокировали ее визуализацию.
- Запрет загрузки неиспользуемых активов (например, изображений, сценариев или библиотек JavaScript).
- Сокращение количества запросов, посылаемых браузером на сервер (известных как HTTP-запросы).

Каждое обращение к серверу в виде HTTP-запроса занимает несколько миллисекунд, и эти миллисекунды действительно способны повлиять на производительность. Все маленькие виджеты Twitter, кнопки «лайков» Facebook и рекламные объявления могут отсылать десятки запросов на сервер. Вы удивитесь, узнав, сколько запросов на сервер отправляет самый простой веб-сайт.

---

* **googleresearch.blogspot.com/2009/06/speed-matters.html**

** Статистические данные взяты из презентации «Make Data Matter» Грэга Линдэна (Стэнфордский Университет, 2006)

Если вы хотите посмотреть подробную информацию о своем сайте, используйте средства разработчика в браузере Chrome, чтобы увидеть каждый запрос на сервер, и сколько миллисекунд на него тратится. Ниже показано, как это делается:

1. Запустите браузер Chrome и перейдите на любую веб-страницу.
2. Перейдите в меню **Настройка и управление Google Chrome** (View) и выберите **Инструменты ⇒ Инструменты разработчика** (Developer ⇒ Developer Tools). В нижней части окна браузера откроется панель.
3. Выберите вкладку **Network** (Сеть) на панели инструментов разработчика и перезагрузите страницу. На диаграмме (обычно называемой *диаграммой водопада*) отображаются все сделанные запросы и загруженные активы. Столбцы справа показывают количество времени в миллисекундах, потраченного на каждый запрос. В нижней части графика можно увидеть сводную информацию о количестве поданных запросов и общий объем переданных данных.

На рис. 3.5 показана часть диаграммы водопада для определения производительности сайта. Вы можете полюбопытствовать относительно любого сайта во Всемирной паутине.

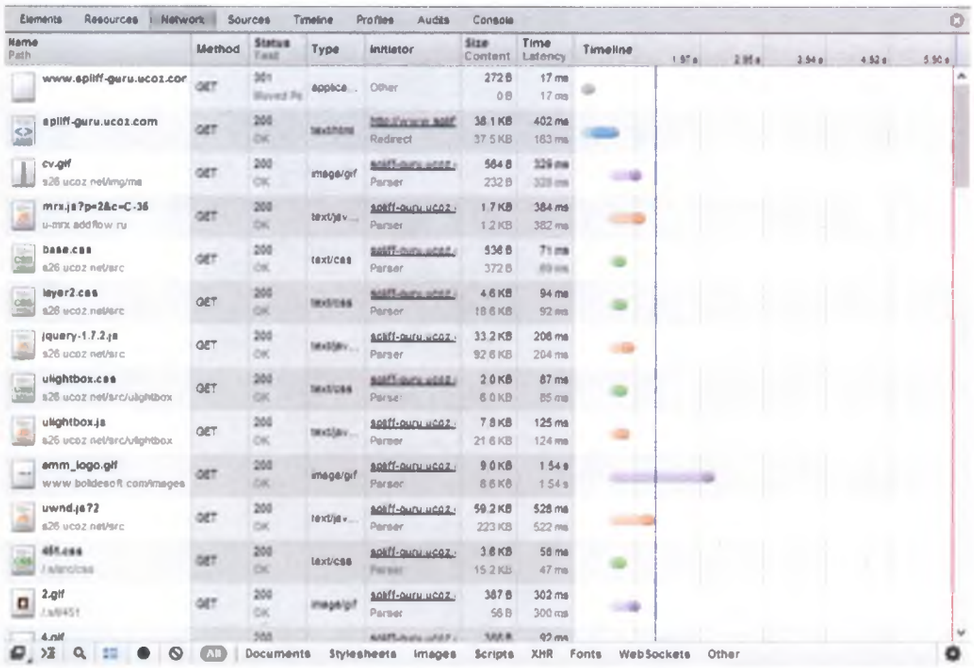


Рис. 3.5. *Подобные диаграммы водопада, создаваемые на вкладке* ***Network*** *браузера Chrome, показывают вам индивидуальные запросы, посылаемые на сервер веб-страницей, и количество времени, которое занимает каждый запрос*

Я не буду уделять много внимания производительности сайта в этой книге, но хочу, чтобы вы, занимаясь веб-дизайном, помнили о том, насколько важно придерживаться как можно меньших размеров файлов и избавляться от ненужных запросов на сервер.

Дополнительные источники информации

Существуют и другие методы, слишком технические для этой книги (и, честно говоря, для меня). Думаю, что если вы читаете эту книгу, то, вероятно, не совсем готовы стать мастером по повышению производительности сайта. Но когда вы захотите этим заняться, можете посетить сайт корпорации Google (developers.google.com/ speed/?hl=ru-RU), который отлично подходит для начала изучения проблемы оптимизации сайтов. На нем собраны превосходные учебники и статьи, а также инструменты для измерения скорости сайта.

Другие инструменты увеличения производительности сайта

Попробуйте применить некоторые из этих инструментов для тестирования производительности веб-сайта:

- WebPagetest (**webpagetest.org**) — инструмент, доступный для общего пользования бесплатно и с открытым исходным кодом. Сайт WebPagetest отображает диаграмму водопада, снимок и другие статистические данные вашего сайта.
- Бесплатный инструмент YSlow (**yslow.org**) компании Yahoo! анализирует сайт в соответствии с 23 правилами веб-производительности, затем присваивает сайту класс и выдает предложения по улучшению сайта.
- Для мобильных сайтов попробуйте программу MobiTest компании Blaze (**www.blaze.io/mobile/**) — бесплатный инструмент для тестирования производительности веб-сайта на различных мобильных устройствах.
- Существует также ряд эмуляторов медленного подключения к Интернету, позволяющих вам почувствовать опыт взаимодействия пользователей при далеко не идеальной скоростью работы сети. Sloppy (**www.dallaway.com/sloppy**) — веб-инструмент, на странице которого вы вводите адрес сайта и выбираете скорость работы модема (а потом ждете, ждете...). Пользователи OS X могут также попробовать программу Slowy (**slowyapp.com**).

РАЗМЕТКА HTML ДЛЯ СТРУКТУРИЗАЦИИ

ЧАСТЬ II

В этой части

ГЛАВА 4

СОЗДАНИЕ ПРОСТОЙ СТРАНИЦЫ

В первой части этой книги были представлены общие принципы разработки веб-страниц. Теперь, когда мы рассмотрели основные концепции, пришло время приступить к созданию реальной веб-страницы. Она будет простой, но даже самые сложные страницы построены на основе описанных здесь принципов.

В этой главе мы пошагово создадим простую веб-страницу, чтобы вы получили представление о разметке документа с помощью HTML-тегов. Упражнения позволят вам одновременно практиковаться.

Я хочу, чтобы вы смогли:

- Получить представление о том, как работает HTML-разметка, а также познакомиться с элементами и атрибутами языка.
- Увидеть, как браузеры интерпретируют HTML-документы.
- Изучить основную структуру HTML-документа.
- Получить первое представление о том, как работает таблица стилей.

Не пугайтесь, что в этой главе придется изучать конкретные элементы текста или правила таблицы стилей, мы обратимся к ним позднее. А сейчас просто сосредоточьтесь на процессе, общей структуре документа, а также новых терминах.

В этой главе

- Знакомство с элементами и атрибутами
- Пошаговая демонстрация разметки простой веб-страницы
- Элементы, составляющие структуру документа
- Основные элементы текста и изображений
- Простая таблица стилей
- Исправление ошибок веб-страниц

Веб-страница шаг за шагом

Вы рассмотрели HTML-документ в главе 2, а теперь получите возможность создать свою собственную страницу и поупражняться с нею в браузере. В этой главе представлены пять основных шагов создания веб-страницы.

Шаг 1: Начальное наполнение страницы контентом. В качестве отправной точки мы добавим текстовый контент и посмотрим, что с ним делают браузеры.

Трудный путь верстки HTML-кода

Я придерживаюсь метода обучения разметке HTML-документов — по старинке, вручную. Нет лучшего способа по-настоящему понять, как работает разметка, чем ввести ее собственными руками, по одному тегу за раз, а затем открыть страницу в браузере. Чтобы развить умение правильной разметки документа, не требуется много времени.

И хотя потом вы, возможно, предпочтете использовать инструменты верстки веб-страниц, понимание синтаксиса языка HTML позволит использовать эти средства более эффективно. Кроме того, вы будете довольны тем, что сможете взглянуть на исходный код файла и понять то, что вы видите. Также это имеет решающее значение для устранения ошибок неработающей страницы или тонкой настройки форматирования по умолчанию, которое выполняют веб-инструменты.

Профессиональные веб-разработчики в основном вручную размечают контент, так как при этом они могут лучше контролировать код и принимать взвешенные решения о том, какие элементы использовать.

Шаг 2: Задание структуры документа. Вы познакомитесь с синтаксисом HTML-элементов и элементами, которые задают структуру документа.

Шаг 3: Определение элементов текста. Вы опишете содержимое, используя соответствующие элементы текста, и узнаете, как правильно применять HTML-элементы.

Шаг 4: Добавление изображений. Добавляя изображение на страницу, вы узнаете об атрибутах и пустых элементах.

Шаг 5: Изменение внешнего вида страницы с помощью таблицы стилей. Это упражнение сформирует представление о том, как, используя каскадные таблицы стилей, оформить содержимое страницы.

К концу этой главы у вас будет создан исходный документ для страницы, показанной на рис. 4.1.

В ходе данной демонстрации мы будем часто проверять нашу работу в браузере — вероятно, чаще, чем вы бы это делали в реальной жизни. Это необходимо потому, что на начальном этапе изучения языка HTML полезно видеть причины и следствия каждого небольшого изменения в исходном файле.

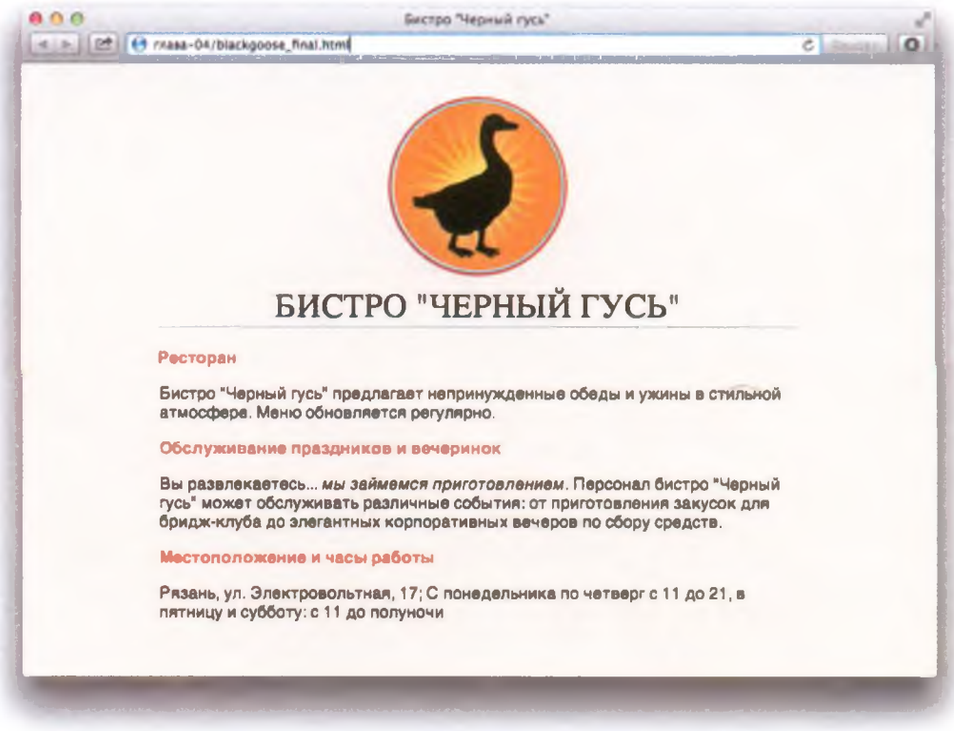


Рис. 4.1. В этой главе мы шаг за шагом запишем исходный код этой веб-страницы

Запуск текстового редактора

В этой главе и на протяжении всей книги, мы будем записывать HTML-документы вручную, поэтому прежде всего необходимо запустить текстовый редактор.

Для этих целей годится текстовый редактор, доступный в составе операционной системы, например Блокнот (Notepad) (Windows) или TextEdit (OS X). Другие текстовые редакторы хороши до тех пор, пока вы можете сохранять обыкновенный текстовый документ (ASCII) в виде файла с расширением *.html*. Если у вас есть средства верстки веб-страниц WYSIWYG, такие как Dreamweaver или SharePoint Designer, на данный момент забудьте о них. Я хочу, чтобы вы прочувствовали процесс разметки документа вручную (см. врезку «Трудный путь верстки HTML-кода»).

В данном разделе показано, как открыть новый документ в текстовых редакторах Блокнот (Notepad) и TextEdit. Даже если вы использовали эти программы прежде, бегло просмотрите некоторые специальные настройки, которые позволят легче выполнить упражнение. Мы начнем с редактора Блокнот (Notepad); пользователи компьютеров Mac могут пропустить этот раздел.

Создание нового документа в редакторе Блокнот (для пользователей Windows)

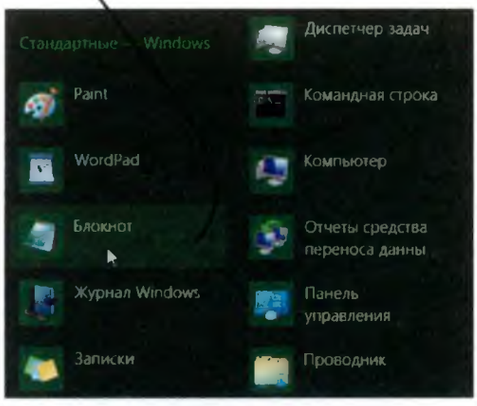
Для создания нового документа в редакторе Блокнот (Notepad) в операционной системе Windows 8 (рис. 4.2) выполняются следующие шаги:

1. Откройте начальный экран (установив указатель мыши в левом нижнем углу экрана и щелкнув мышью) и найдите на нем программу Блокнот (Notepad). Если там этот ярлык отсутствует, откройте список всех установленных программ, щелкнув правой кнопкой мыши по экрану и нажав кнопку **Все приложения** (All apps) в правом нижнем углу. Ярлык программы Блокнот (Notepad) находится в папке **Стандартные** (Accessories).
2. Щелкнув по нему, откройте новый документ. Вы готовы к вводу кода.
3. Далее мы отобразим расширения файлов. Этот шаг не требуется для создания HTML-документов, но поможет с первого взгляда определить тип файла. В любом окне программы Проводник (Explorer) перейдите на вкладку **Вид** (View) и в раскрывающемся списке **Параметры** (Options) выберите пункт **Изменить параметры папок и поиска** (Change folder and search options).
4. В открывшемся диалоговом окне **Параметры папок** (Folder and search options) перейдите на вкладку **Вид** (View).
5. Сбросьте флажок **Скрывать расширения для зарегистрированных типов файлов** (Hide extension for known file types). Щелкните по кнопке **OK**, чтобы сохранить изменения. Расширения файлов теперь будут отображаться.

ПРИМЕЧАНИЕ

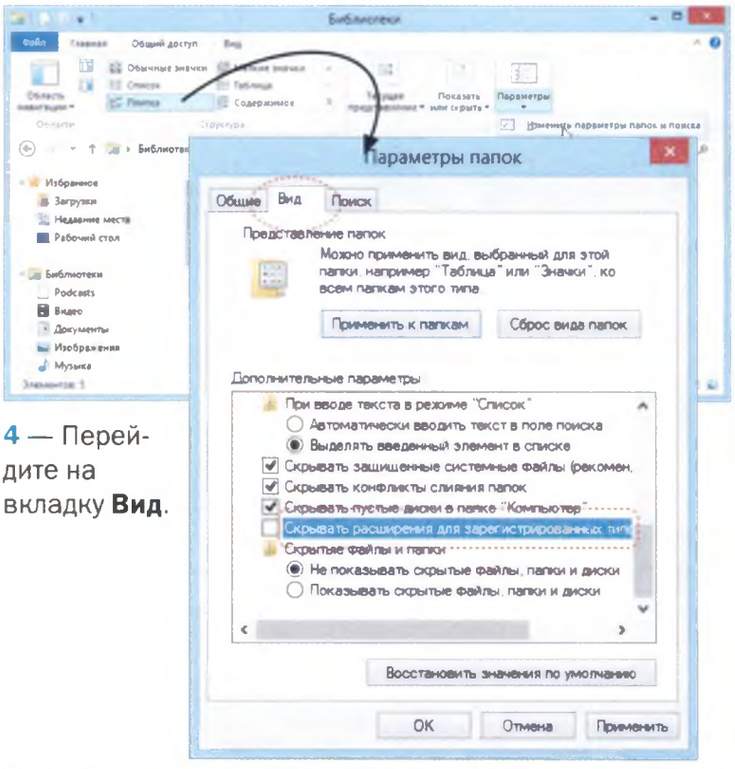
В операционной системе Windows 7 для открытия диалогового окна **Параметры папок** (Folder and search options) нажмите клавишу **Alt**, чтобы отобразить меню, а затем выберите команду меню **Параметры** ⇒ **Параметры папок** (Options ⇒ Folder options). В операционной системе Windows Vista эта команда называется **Параметры папок и поиска** (Folder and Search Options).

2 — Щелкнув по ярлыку программы Блокнот, создайте новый документ.



1 — На начальном экране найдите текстовый редактор Блокнот.

3 — Откройте диалоговое окно **Параметры папок**, выбрав в окне программы Проводник команду меню **Параметры ⇒ Изменить параметры папок и поиска**.



4 — Перейдите на вкладку **Вид**.

5 — Сбросьте флажок **Скрывать расширения для зарегистрированных типов файлов**. Щелкните по кнопке **OK**, чтобы сохранить настройки.

Рис. 4.2. Создание нового документа в редакторе Блокнот и отображение расширений файлов

Создание нового документа в редакторе TextEdit (для пользователей OS X)

По умолчанию текстовый редактор TextEdit создает документы в формате форматированного текста, то есть такие документы имеют скрытые инструкции для форматирования текста — начертания полужирным шрифтом, настройки размера шрифта и так далее. HTML-файлы должны быть простыми текстовыми документами, поэтому нужно изменить формат, как показано в следующем примере (рис. 4.3).

1. Чтобы найти редактор TextEdit в папке **Приложения** (Applications), воспользуйтесь приложением Finder. Для запуска найденного щелкните два раза либо по имени приложения, либо по его значку.
2. В редакторе TextEdit откроется новый документ. Если в верхней части окна документа отображается панель форматирования, включен режим форматированного текста. Вы можете изменить этот режим следующим образом.
3. С помощью команды меню **TextEdit ⇒ Настройки** (TextEdit ⇒ Preferences) откройте одноименное окно.

4. Далее нужно настроить следующие параметры:
   - Установите переключатель в положение **Простой текст** (Plain text).
   - На вкладке **Открытие и сохранение** (Open and Save) установите флажок **Отображать файлы HTML в виде кода HTML, а не форматированного текста** (Display HTML files as HTML code instead of formatted text) и сбросьте флажок **Добавлять расширение .txt к именам файлов простого текста** (Add '.txt' extension to plain text files).
5. После выполнения всех этих действий закройте окно **Настройки** (Preferences).
6. При создании нового документа в программе TextEdit уже не будет отображаться панель форматирования. Документ можно конвертировать обратно в форматированный текст в любой момент, выбрав команду меню **Формат ⇒ Конвертировать в форматированный текст** (Format ⇒ Make Rich Text), если вы не используете программу TextEdit для верстки HTML-файлов.

Панель в верхней части окна документа указывает, что документ открыт в режиме форматированного текста

У документа в формате простого текста нет панели

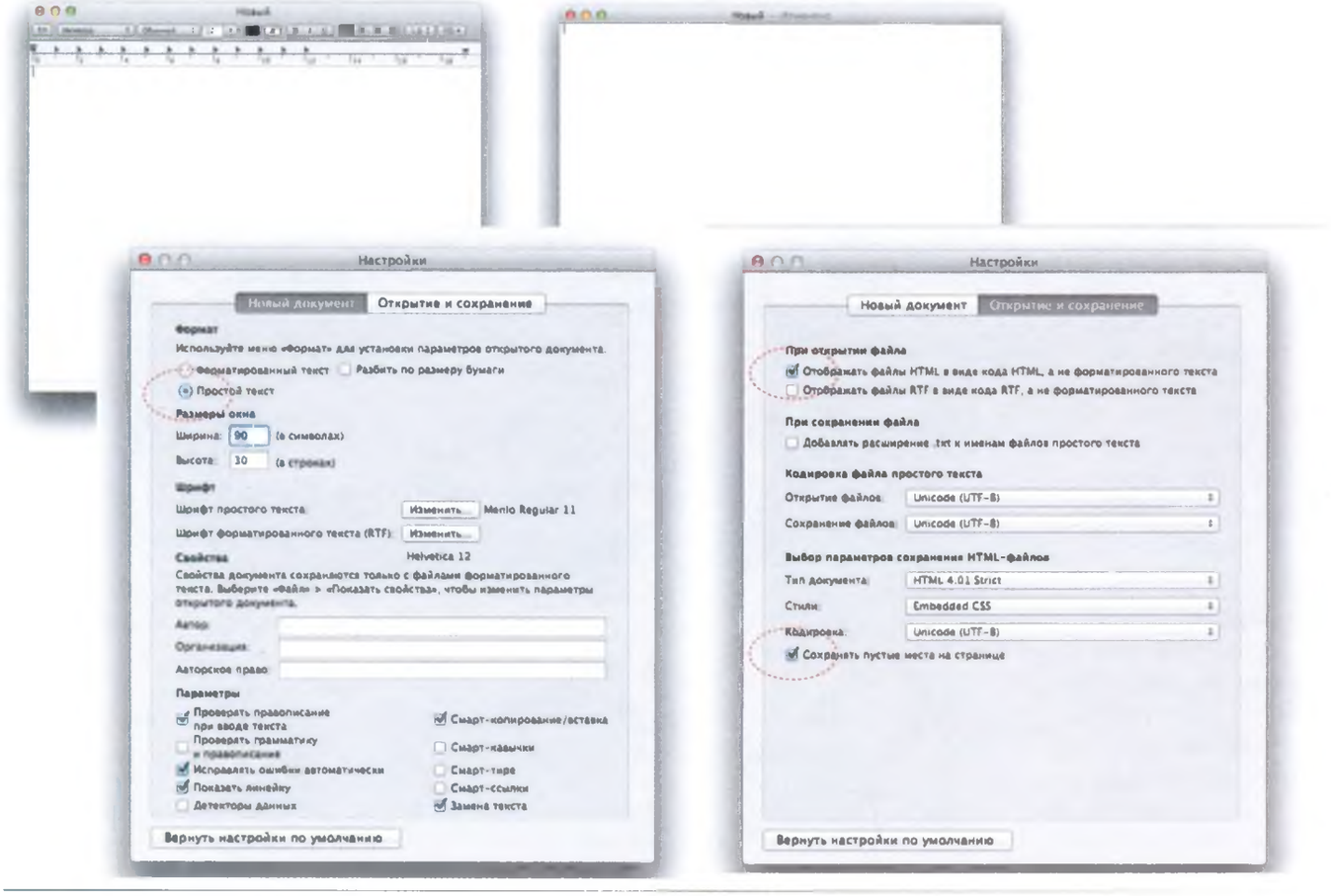


Рис. 4.3. Запуск редактора TextEdit и выбор режима простого текста в окне ***Настройки***

Допустимые имена файлов

Присваивая имена файлам, придерживайтесь следующих правил и соглашений:

Используйте соответствующие суффиксы. Файлы HTML должны заканчиваться расширением *.html* (некоторые серверы допускают использование расширения *.htm*). Графические файлы для веб-страниц должны быть помечены в соответствии со своим форматом: *.png*, *.gif* или *.jpg* (формат *.jpeg* также приемлем).

Никогда не используйте пробелы в именах файлов. Чтобы визуально отделить слова, используйте символ подчеркивания или тире. Например, *vladimir_biografija.html* или *vladimir-biografija.html*.

Избегайте специальных символов и кириллицы, таких как ?, %, #, /, :, ;, * и т. д. Имена файлов должны содержать только латинские буквы, цифры, знаки подчеркивания, тире и точки.

В зависимости от конфигурации сервера имена файлов могут быть чувствительны к регистру. Последовательное использование только строчных букв, даже когда в этом нет необходимости, позволит легко управлять именами ваших файлов.

Придерживайтесь коротких имен. Короткие имена позволяют контролировать количество символов и размер HTML-файла. Если вы действительно должны дать файлу длинное, многословное имя, для улучшения восприятия вы можете отделить слова заглавными буквами, например *DlinnoeNazvanieDocumenta.html*, или символами подчеркивания, например, *dlinnoe_nazvanie_documenta.html*.

Собственные соглашения. Полезно разработать последовательную схему именования для огромных сайтов. Например всегда использовать строчные символы и подчеркивания между словами. В дальнейшем при переходе по ссылке на этот файл вы сможете вспомнить, почему вы назвали его именно так.

Шаг 1: Добавление контента

Теперь, когда вы создали новый документ, пришло время ввода контента. Веб-страницы всегда начинаются с этого.

Упражнение 4.1 поможет вам справиться со вводом исходного текста и сохранением документа в новой папке.

УПРАЖНЕНИЕ 4.1. ВВОД КОНТЕНТА

1. Введите текст главной страницы в новый документ в текстовом редакторе в том виде, в котором вы его здесь видите. Для повторения примера сохраните такие же пустые строки. Неотформатированный вариант текста для этой страницы доступен на диске, прилагающемся к книге, в файле *Примеры\глава-04\Текст.txt*.

```
Бистро "Черный гусь"

Ресторан

Бистро "Черный гусь" предлагает непринужденные обеды
и ужины в стильной атмосфере. Меню обновляется регу-
лярно.

Обслуживание праздников и вечеринок

Вы развлекаетесь — мы займемся приготовлением. Пер-
сонал бистро "Черный гусь" может обслуживать раз-
личные события: от приготовления закусок для бридж-
клуба до элегантных корпоративных вечеров по сбору
средств.

Местоположение и часы работы

Рязань, ул. Электровольтная, 17;

С понедельника по четверг с 11 до 21, в пятницу
и субботу с 11 до полуночи
```

2. Выберите команду меню **Файл** ⇒ **Сохранить** (File ⇒ Save) или **Файл** ⇒ **Сохранить как** (File ⇒ Save as), чтобы перейти в диалоговое окно **Сохранить как** (Save as) (рис. 4.4). Первое, что вы должны сделать — это создать новую папку, которая будет содержать все файлы для этого сайта, другими словами *локальную корневую папку*.
   - Windows: Чтобы создать новую папку, щелкните по кнопке с изображением папки в верхней части диалогового окна.
   - OS X: Щелкните по кнопке **Новая папка** (New Folder).

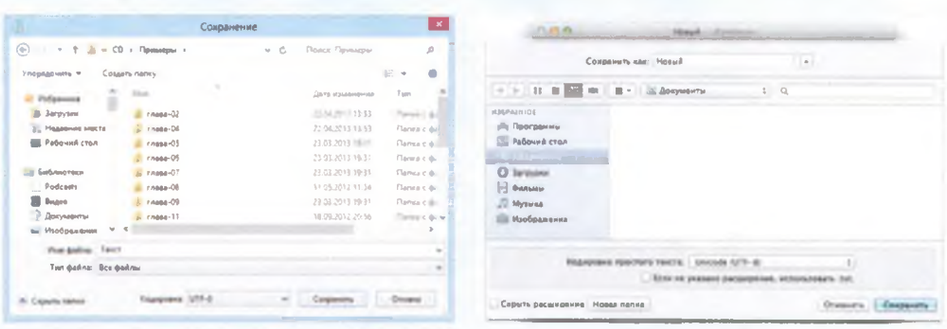


Рис. 4.4. Сохранение файла index.html в новой папке с именем bistro

Назовите новую папку ***bistro*** и сохраните в ней текстовый файл с именем ***index.html***. Пользователи операционной системы Windows должны выбрать пункт **Все файлы** (All Files) из списка **Тип файла** (Save as type), чтобы избавиться от расширения *.txt*, которое добавляет к имени файла редактор Блокнот (Notepad). Чтобы файл был признан браузером в качестве веб-документа, его имя должно заканчиваться на *.html*. Дополнительные советы по присвоению файлам имен можно найти во врезке «Допустимые имена файлов».

Предупреждение. Для правильного отображения кириллических символов (то есть русского текста) необходимо в раскрывающемся списке **Кодировка** (Encoding) выбрать пункт **UTF-8**.

3. Взглянем на файл *index.html* в браузере. Запустите любой браузер (я использую Firefox) и выберите команду меню **Файл** ⇒ **Открыть** (File ⇒ Open) или **Файл** ⇒ **Открыть файл** (File ⇒ Open file). Перейдите к файлу *index.html* и выберите его, чтобы открыть в браузере. Вы должны увидеть что-то похожее на ту страницу, что показана на рис. 4.5. Мы поговорим о результатах в следующем разделе.

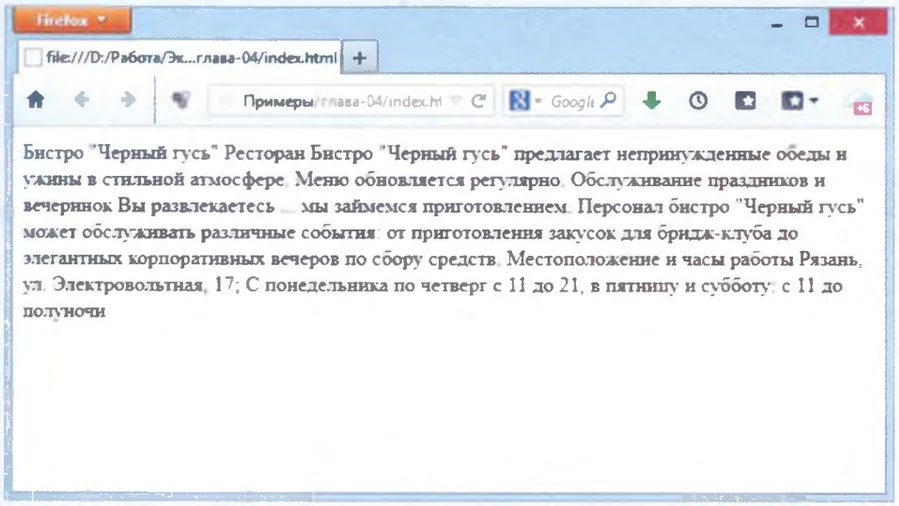


Рис. 4.5. Первый вариант главной страницы, показанный в браузере

Резюме

Наш контент выглядит не слишком хорошо (рис. 4.5). Все слова в тексте оказались сдвинутыми вместе, а это отличается от того, как выглядел текст в исходном документе. Из этого можно сделать некоторые выводы. Первое, что очевидно, — браузер игнорирует строки в исходном документе. Во врезке «Что игнорируют браузеры» перечислены и другие виды данных, которые не отображаются в окне браузера.

Во-вторых, мы видим, что простого ввода некоторого контента и присвоения документу имени с расширением *.html* недостаточно. Хотя браузер отображает текст из файла, *структура* контента осталась неопределенной. И вот здесь наступает очередь языка HTML. Чтобы доба-

Что игнорируют браузеры

При просмотре в браузере часть информации в исходном документе игнорируется, в том числе:

Знаки табуляции и несколько пробелов. Когда браузер встречает знак табуляции или последовательность из нескольких пробелов, он отображает один пробел. Так что, если документ содержит такой, например, текст:

```
много,      много лет      назад
```

в браузере он отобразится так:

```
много, много лет назад
```

Переводы строк (разрывы или переносы строк). Браузеры воспринимают переводы строк как пробелы, поэтому, исходя из вышеописанного пункта, переводы строк не влияют на форматирование страницы. Текст и элементы будут следовать непрерывно, пока в тексте документа не встретится новый блочный элемент, например, заголовок (`h1`) или абзац (`p`), или элемент перевода строки (`br`).

Табуляция. Табуляция также воспринимается как пробелы между символами, и это значит, что она бесполезна.

Неопознанные элементы разметки. Браузеры игнорируют любые теги, которые они не понимают, или те, что были введены с ошибками. В зависимости от элементов и браузера, это приводит к различным результатам. Браузер может вообще ничего не отобразить или покажет содержимое тега так, как если бы это был обычный текст.

Текст в комментариях. Браузеры не будут отображать текст между специальными тегами `<!--` и `-->`, используемыми для обозначения комментариев. См. врезку «Добавление скрытых комментариев» далее в этой главе.

вить структуру сначала к самому HTML-документу (рассматривается в шаге 2), затем к содержимому страницы (шаг 3), мы будем использовать *разметку*. Если браузер «знает» структуру контента, он сможет сделать отображаемые страницы более понятными.

Шаг 2: Структурирование документа

У нас есть контент, который мы сохранили в HTML-документе, теперь мы готовы приступить к его разметке.

HTML-элементы

Элемент состоит из содержимого (контента) и его разметки.

Вернувшись к главе 2, вы увидите примеры HTML-элементов с открывающим тегом, например ** ** для абзаца, и закрывающим (** **). Прежде чем начать добавлять теги к нашему документу, рассмотрим структуру HTML-элемента и изучим некоторые важные термины. Общий вид HTML-элемента представлен на рис. 4.6.

Элементы определяются тегами в исходном тексте. *Тег* состоит из имени элемента (как правило, это аббревиатура длинного имени элемента) заключенного в угловые скобки (**< >**). Браузер «знает», что любой текст внутри угловых скобок является скрытым, и не отображает его в своем окне.

Имя элемента появляется в *открывающем теге* (также называемом *начальным тегом*), а затем в *закрывающем* (или *конечном*) *теге*, после слеша (**/**). Закрывающий тег работает чем-то вроде выключателя для элемента. Будьте внимательны и не используйте подобный символ — обратный слеш в закрывающем теге (смотрите совет «Слеш против обратного слеша»).

Теги, добавленные к контенту, называются *разметкой*. Важно отметить, что *элемент* состоит *как из* контента, *так и* из разметки (открывающий и закрывающий теги). Однако не все элементы содержат контент.

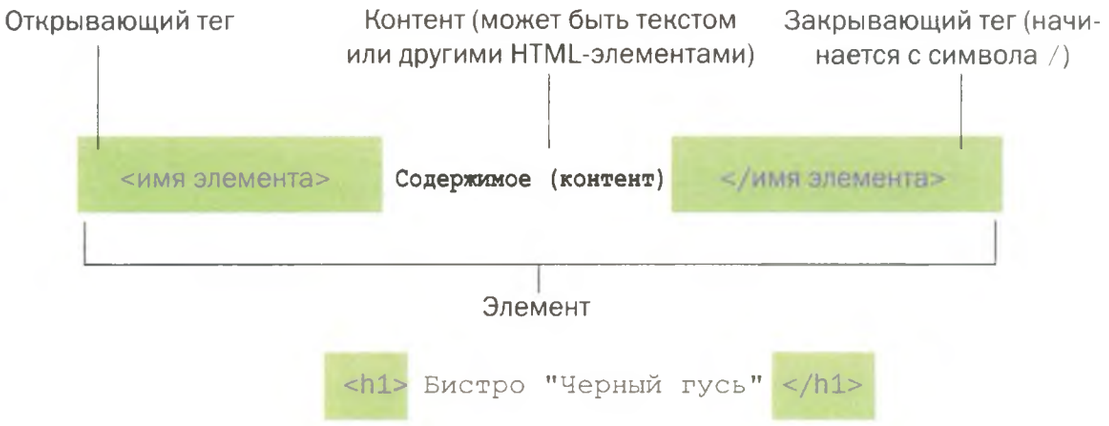


Рис. 4.6. Структура HTML-элемента

Некоторые из них *пустые* по определению, такие, например, как элемент **img**, используемый для добавления изображения на страницу. О пустых элементах мы поговорим в этой главе немного позже.

И еще один момент — использование прописных букв. В языке HTML использование прописных букв в именах элементов не играет большой роли. Так, для браузера теги **<img>**, **<Img>** и **<IMG>** означают одно и то же. Однако в языке XHTML (более строгой версии HTML) все имена элементов должны быть указаны строчными буквами, чтобы считаться допустимыми. Многим веб-разработчикам нравится упорядоченность более строгой разметки XHTML, и они придерживаются написания строчными буквами, как и я в данной книге.

Базовая структура документа

На рис. 4.7 показан рекомендуемый упрощенный каркас документа на языке HTML5. Я говорю «рекомендуемый» потому, что единственным *обязательным* элементом в HTML является тег **title**. Но я рекомендую, особенно новичкам, четко организовать документы с помощью надлежащей структурной разметки. И если вы верстаете страницы, используя XHTML, все перечисленные ниже элементы, кроме тега **meta**, обязательно должны присутствовать, чтобы код был верным. Давайте посмотрим, что происходит на рис. 4.7.

❶ Я не хочу сбивать вас с толку, но первая строка в примере — вовсе не элемент. Это объявление типа документа (также называемое *объявление DOCTYPE*), определяющее этот файл как документ на языке HTML5. Я расскажу подробнее об объявлениях типа документа в главе 10, а в рамках этого обсуждения достаточно сказать, что его добавление сообщает браузерам: документ необходимо интерпретировать в соответствии со спецификацией HTML5.

❷ Весь документ заключен в элемент **html**. Элемент **html** называется *корневым элементом*, поскольку он содержит все элементы в документе и не может содержаться ни в каком другом элементе. Он используется как для HTML, так и для XHTML-документов.

❸ Содержимое внутри элемента **html** состоит из *головы* (*раздел заголовка*) и *тела*. Элемент **head** (раздел заголовка) содержит описательную информацию о документе, такую как заголовок, используемые таблицы стилей, сценарии и прочие метаданные.

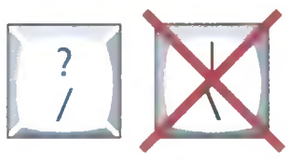
❹ Элементы **meta**, расположенные внутри элемента **head**, предоставляют данные о самом документе. Элемент **meta** можно использовать для предоставления самой разнообразной информации, но в данном случае он определяет *кодировку набора символов* (стандартизированный набор букв, цифр и символов), применяемую в документе. Я не хочу сейчас слишком углубляться в детали, но имейте в виду, что существует множество веских причин для указания атрибута **charset** в каждом документе, поэтому я включила его в минимальную структуру документа.

СОВЕТ

Слеш против обратного слеша

В HTML-тегах и URL-адресах используется наклонная черта (слеш) (/). Слеш находится под знаком вопроса (?) на стандартной QWERTY-клавиатуре в английской раскладке.

Этот символ легко спутать с обратной наклонной чертой (обратным слешем) (\), которая находится под символом вертикальной черты (|). Обратный слеш не допустим в именах тегов или URL-адресах, так что будьте внимательны и не используйте его.



ПРИМЕЧАНИЕ

До появления спецификации HTML5 синтаксис, используемый для указания набора символов в элементе **meta**, был значительно сложнее. Если бы вы писали документ на языке HTML 4.01 или XHTML, элемент **meta** выглядел бы так:

```
<meta http-equiv="content-type"
content="text/html;
charset=UTF-8">
```

Польза элемента title

Для каждого документа, элемент **title** не только необходим, но и очень полезен. Заголовок документа — это то, что отображается в списке Закладок или в Избранном пользователя, а также на вкладках в браузере. Кроме того, описательные заголовки документа — ключевой инструмент повышения доступности, так как они — первое, что человек видит при использовании программ экранного доступа. Поисковые системы также в значительной степени зависят от заголовков документов. Поэтому важно обеспечить *содержательные и описательные* заголовки для всех ваших документов и избегать расплывчатых заголовков, таких как «Добро пожаловать» или «Моя страница». Также можно ограничить длину заголовков документа, чтобы они могли уместиться в области заголовка браузера. Другим удачным приемом будет поместить в начале часть заголовка, содержащую более определенную информацию (например, расположить описание страницы перед названием компании), чтобы заголовок документа был виден, когда в окне браузера открыто сразу несколько вкладок.

5. Также в разделе заголовка обязателен элемент **title**. В соответствии со спецификацией HTML, каждый документ должен содержать описательное название.
6. Наконец, элемент **body** (тело) содержит весь контент, который мы хотим отобразить в окне браузера.

Готовы ли вы добавить некоторую структуру к главной странице бистро «Черный гусь»?

Откройте документ *index.html* и перейдите к упражнению 4.2.

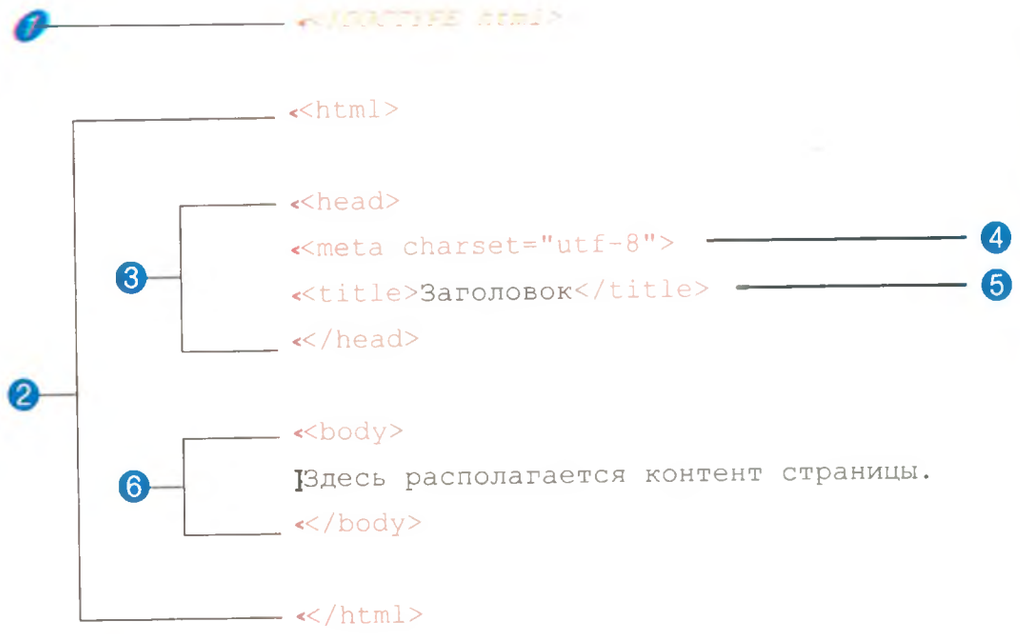


Рис. 4.7. Минимальная структура HTML-документа

После структурирования не так много изменилось, разве что на вкладке браузера теперь отображается название документа. Оно также будет добавлено в Закладки или Избранное, если кто-нибудь захочет сохранить там эту страницу (см. врезку «Польза элемента **title**»). Однако контент все еще представляет собой единое целое, поскольку мы не дали браузеру указаний, как его разделить. Мы позаботимся об этом в дальнейшем.

Шаг 3: Определение текстовых элементов

С небольшим опытом верстки веб-документов можно, не слишком задумываясь, добавить разметку, которая определяет заголовки и подзаголовки (**h1** и **h2**), абзацы (**p**) и выделенный текст (**em**) к нашему контенту, как это мы сделали в упражнении 4.3. Однако прежде чем начать, я хочу поговорить о том, что мы *делаем* и что *не делаем* при разметке контента с помощью языка HTML.

ПРИМЕЧАНИЕ

До появления спецификации HTML5 синтаксис, используемый для указания набора символов в элементе **meta**, был значительно сложнее. Если бы вы писали документ на языке HTML 4.01 или XHTML, элемент **meta** выглядел бы так:

```
<meta http-
equiv="content-type"
content="text/html;
charset=UTF-8">
```

Польза элемента title

Для каждого документа, элемент **title** не только необходим, но и очень полезен. Заголовок документа — это то, что отображается в списке Закладок или в Избранном пользователя, а также на вкладках в браузере. Кроме того, описательные заголовки документа — ключевой инструмент повышения доступности, так как они — первое, что человек видит при использовании программ экранного доступа. Поисковые системы также в значительной степени зависят от заголовков документов. Поэтому важно обеспечить содержательные и описательные заголовки для всех ваших документов и избегать расплывчатых заголовков, таких как «Добро пожаловать» или «Моя страница». Также можно ограничить длину заголовков документа, чтобы они могли уместиться в области заголовка браузера. Другим удачным приемом будет поместить в начале часть заголовка, содержащую более определенную информацию (например, расположить описание страницы перед названием компании), чтобы заголовок документа был виден, когда в окне браузера открыто сразу несколько вкладок.

5. Также в разделе заголовка обязателен элемент **title**. В соответствии со спецификацией HTML, каждый документ должен содержать описательное название.
6. Наконец, элемент **body** (тело) содержит весь контент, который мы хотим отобразить в окне браузера.

Готовы ли вы добавить некоторую структуру к главной странице бистро «Черный гусь»?

Откройте документ *index.html* и перейдите к упражнению 4.2.

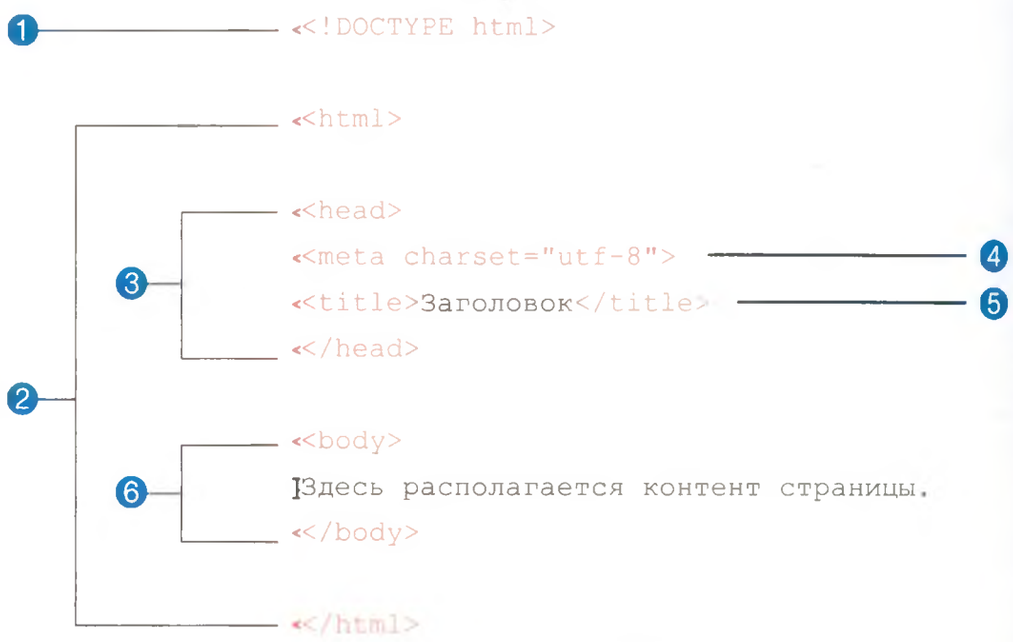


Рис. 4.7. Минимальная структура HTML-документа

После структурирования не так много изменилось, разве что на вкладке браузера теперь отображается название документа. Оно также будет добавлено в Закладки или Избранное, если кто-нибудь захочет сохранить там эту страницу (см. врезку «Польза элемента **title**»). Однако контент все еще представляет собой единое целое, поскольку мы не дали браузеру указаний, как его разделить. Мы позаботимся об этом в дальнейшем.

Шаг 3: Определение текстовых элементов

С небольшим опытом верстки веб-документов можно, не слишком задумываясь, добавить разметку, которая определяет заголовки и подзаголовки (**h1** и **h2**), абзацы (**p**) и выделенный текст (**em**) к нашему контенту, как это мы сделали в упражнении 4.3. Однако прежде чем начать, я хочу поговорить о том, что мы *делаем* и что *не делаем* при разметке контента с помощью языка HTML.

УПРАЖНЕНИЕ 4.2. ДОБАВЛЕНИЕ ОСНОВНОЙ СТРУКТУРЫ

1. Откройте недавно созданный документ *index.html*, если он еще не открыт.
2. Сначала добавьте объявление **DOCTYPE** на языке HTML5:

```
<!DOCTYPE html>
```

3. Поместите весь документ в корневой элемент HTML, добавив открывающий тег **<html>** в самом начале и закрывающий тег **</html>** в конце текста.
4. Далее создадим заголовок документа, который содержит название страницы. Вставьте теги **<head>** и **</head>** до начала контента. Внутри элемента **head** добавьте информацию о кодировке набора символов (**<meta charset="utf-8">**) и название **Бистро "Черный гусь"**, окруженное открывающим и закрывающим тегами **<title>**.

   *Исходя из корректной терминологии, элемент **title** должен быть вложен внутрь элемента **head**. Я расскажу о вложенности в следующих главах.*
5. Определим, наконец, тело документа, заключив содержимое в теги **<body>** и **</body>**. По окончании ваших действий исходный документ должен выглядеть следующим образом (разметка выделена):

```
<!DOCTYPE html>
<html>
<head>
<meta charset ="utf-8">
<title>Бистро "Черный гусь"</title>
</head>
<body>
Бистро "Черный гусь"
Ресторан
Бистро "Черный гусь" предлагает непринужденные обеды и ужины в стильной атмосфере. Меню обновляется регулярно.
Обслуживание вечеринок и праздников
Вы развлекаетесь — мы займемся приготовлением. Персонал бистро "Черный гусь" может обслуживать различные события: от приготовления закусок для бридж-клуба до элегантных корпоративных вечеров по сбору средств.
Местоположение и часы работы
Рязань, ул. Электровольтная, 17;
С понедельника по четверг с 11 до 21, в пятницу и субботу с 11 до полуночи
</body>
</html>
```

6. Сохраните документ в папке *bistro* так, чтобы переписать старый файл. Откройте файл в браузере или обновите страницу, если она уже открыта. На рис. 4.8 показано, как должна выглядеть страница.

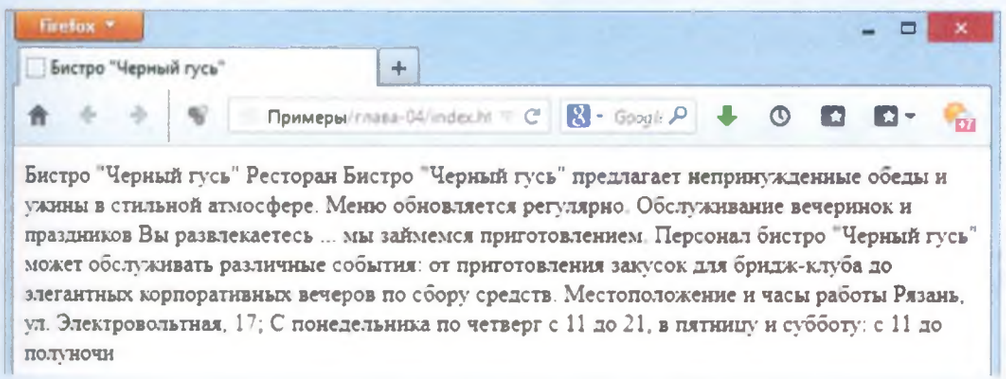


***Рис. 4.8.** Главная страница в браузере после определения элементов структуры*

Семантическая разметка

Цель языка HTML заключается в обеспечении смысла и структуры контента. Этот язык *не* предназначен для создания инструкции о том, как должен выглядеть контент документа (его представления).

Ваша работа при разметке контента состоит в выборе HTML-элемента, который обеспечивает наиболее значимое описание. Это называется *семантической разметкой*. Например, первый уровень заголовка на странице должен быть отмечен как `h1`, поскольку это наиболее важный заголовок на странице. Не заботьтесь о том, как он выглядит в браузере, его вид можно легко изменить с помощью таблицы стилей. Важно то, что вы выбираете элементы наиболее подходящие для контента.

Разметка не только наделяет контент смыслом, но и структурирует документ. Между определяющими элементами, которые следуют друг за другом или вложены друг в друга, создаются отношения, которые можно представить как структуру (или, если использовать профессиональную терминологию, как объектную модель документа (Document Object Model, *DOM*)). Скрытая иерархия документа важна, так как в ней содержатся инструкции для браузеров о том, как отображать документ. Кроме того, это основа, к которой мы можем добавить инструкции для представления документа с помощью таблиц стилей и поведения с помощью языка JavaScript. Более подробно говорить о структуре документа мы будем в части III этой книги при обсуждении каскадных таблиц стилей и части V при обсуждении языка JavaScript.

Хотя с момента своего создания язык HTML был предназначен исключительно для придания смысла и структуры, в первые годы Всемирной паутины его миссия была слегка сорвана. Из-за отсутствия системы таблиц стилей язык HTML был расширен, чтобы дать веб-дизайнерам способы изменять внешний вид шрифтов, цвет и выравнивание. Эти дополнительные возможности все еще существуют, и вы можете столкнуться с ними при просмотре исходного кода страниц старых вебсайтов или страниц, сверстанных с помощью устаревшего программного обеспечения. Однако в этой книге мы сосредоточимся на правильном пути использования языка HTML в соответствии с современным семантическим подходом к веб-разработке на основе стандартов.

Итак, довольно лекций. Настало время приступить к работе над контентом в упражнении 4.3.

Теперь у нас уже кое-что получилось. Если элементы разметки указаны правильно, браузер может отображать текст в максимально понятном виде. В том, что изображено на рис. 4.9, важно отметить несколько моментов.

УПРАЖНЕНИЕ 4.3. ОПРЕДЕЛЕНИЕ ЭЛЕМЕНТОВ ТЕКСТА

1. Откройте документ *index.html* в текстовом редакторе, если он еще не открыт.
2. Первая строка текста в теле документа, «Бистро «Черный гусь»», — основной заголовок на странице, поэтому мы пометим его как элемент заголовка первого уровня (**h1**). Введите открывающий тег **<h1>** в начале строки и закрывающий тег **</h1>** в конце, как показано ниже:

```
<h1>Бистро "Черный гусь"</h1>
```

3. Наша страница имеет три подзаголовка. Подобным образом помечаем их как элементы заголовка 2-го уровня. Первый я укажу в этом упражнении, а вы самостоятельно сделаете то же самое для заголовков «Обслуживание вечеринок и праздников» и «Местоположение и часы работы».

```
<h2>Ресторан</h2>
```

4. Каждый элемент **h2** сопровождается кратким абзацем текста, так что давайте отметим их в качестве элементов абзаца (**p**). Ниже представлен первый абзац, вы делаете остальные.

```
<p>Бистро "Черный гусь" предлагает непринужденные обеды и ужины в стильной атмосфере. Меню регулярно обновляется, чтобы продемонстрировать новые блюда.</p>
```

5. Наконец, в абзаце «Обслуживание вечеринок и праздников» я хочу подчеркнуть, что посетители могут не заботиться о приготовлении блюд. Чтобы акцентировать текст, отметьте его элементом **em**, как показано ниже.

```
<p>Вы развлекаетесь — <em>мы займемся приготовлением</em>. Персонал бистро "Черный гусь" может обслуживать различные события: от приготовления закусок для бридж-клуба до элегантных корпоративных вечеров по сбору средств.</p>
```

6. Теперь, когда мы разметили документ, давайте сохраним его, как делали раньше, и откроем (или обновим) страницу в браузере. Вы должны увидеть страницу, которая выглядит так же, как показано на рис. 4.9. Если этого не происходит, проверьте разметку, чтобы убедиться, что вы не пропустили скобки или слеш в закрывающем теге.

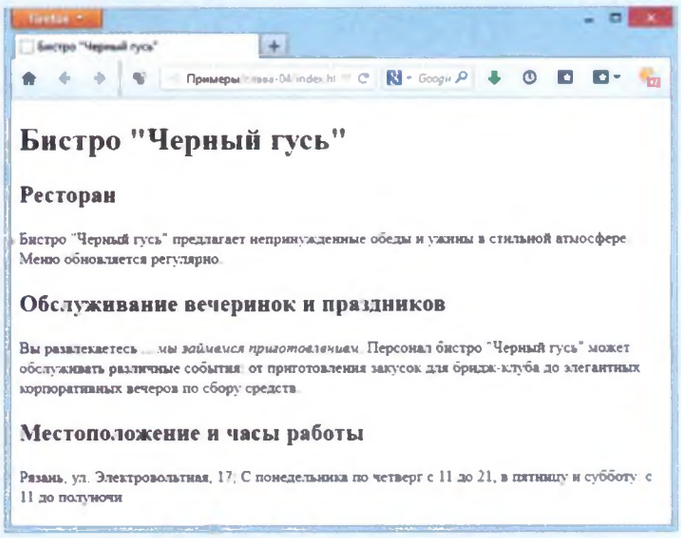


Рис. 4.9. *Главная страница, размеченная с помощью элементов языка HTML*

Блочные и встроенные элементы

Хотя это может показаться очевидным, стоит отметить, что элементы заголовков и абзацев начинаются с новой строки и не собраны все вместе, как было ранее. Это потому, что они являются примерами *блочных элементов* различных уровней. Браузеры рассматривают блочные элементы так, словно они заключены в маленькие прямоугольные области, из которых складываются страницы. По умолчанию каждый блочный элемент начинается с новой строки и, как правило, ограничен сверху и снизу пустыми строками. На рис. 4.10 границы блочных элементов выделены красным цветом.

По-другому выглядит текст, который мы пометили как подчеркнутый (**em**). Он не начинается с новой строки, а остается в составе абзаца. Это потому, что элемент **em** является *встроенным*. Встроенные элементы не начинаются с новой строки; они лишь находятся в составе блоков. На рис. 4.10 встроенный элемент **em** выделен голубым цветом.

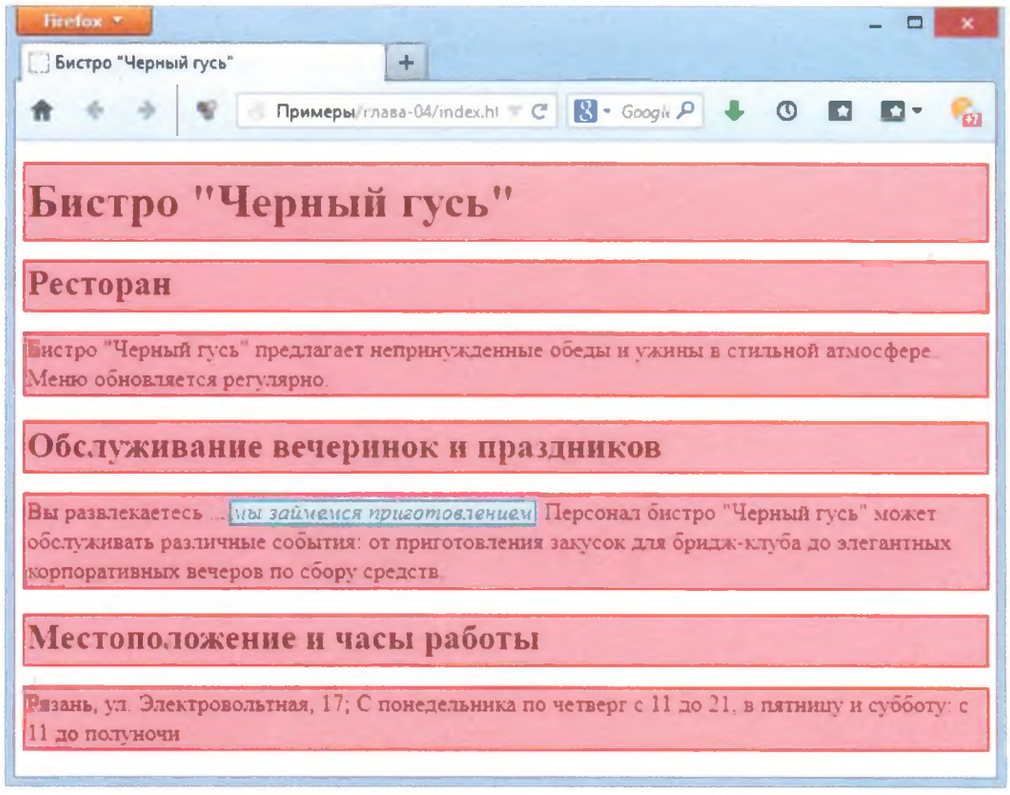


Рис. 4.10. С помощью выделенных областей показана структура элементов веб-страницы

Стили по умолчанию

Рассматривая размеченные страницы на рис. 4.9 и 4.10, вы также заметите, что браузер пытается придать странице некоторую визуальную иерархию, выделив заголовок первого уровня более крупным и жирным шрифтом, заголовок второго уровня немного меньшим шрифтом и так далее.

Как браузер определяет, что элемент **h1** должен выглядеть именно так? Он использует таблицу стилей! У всех браузеров есть собственные встроенные таблицы стилей (называемые в спецификациях *таблицами стилей пользовательского агента*), которые описывают представление элементов по умолчанию. Оно схоже у различных браузеров (например, элементы **h1** всегда выделены крупным и жирным шрифтом), но есть некоторые отличия (так, кавычки могут иметь отступы, а могут и не иметь).

Добавление скрытых комментариев

Вы можете оставлять комментарии в исходном документе для себя и других пользователей, помечая их как комментарии. Все содержимое, которое вы разместите между тегами комментариев (`<!-- -->`), не будет отображаться в браузере, и не будет оказывать никакого влияния на остальную часть контента.

```
<!-- Это комментарий -->
<!-- Это многострочный
комментарий,
который заканчивается
здесь -->
```

Комментарии полезны для маркировки и организации длинных документов — в частности, когда над ними совместно трудится команда разработчиков. В этом примере комментарии используются, чтобы отметить раздел исходного документа, который содержит элементы навигации.

```
<!-- Начало элементов
навигации -->
<ul>
...
</ul>
<!-- Окончание элементов
навигации -->
```

Имейте в виду, что, хотя браузер не будет отображать комментарии на веб-странице, посетители смогут увидеть их, если просмотрят исходный код документа, поэтому убедитесь, что комментарии, которые вы оставляете, допустимы для просмотра. Возможно, стоит удалить заметки, оставленные коллегами-разработчиками перед публикацией файла. К тому же это уменьшит его размер.

Если вам кажется, что браузер делает заголовок **h1** слишком большим и неуклюжим, просто измените правила его таблицы стилей. Не поддавайтесь желанию разметить заголовок с помощью другого элемента только затем, чтобы он выглядел лучше (например, с помощью тега **h3** вместо **h1**, чтобы уменьшить шрифт). До появления вездесущих таблиц стилей элементы подвергались изменению именно таким образом. Теперь, когда для управления процессом разработки веб-страниц появились таблицы стилей, вы должны всегда выбирать элементы на основе того, насколько точно они описывают содержание, и не беспокоиться о представлении браузером элементов по умолчанию.

С помощью таблиц стилей мы исправим представление страницы мгновенно, но сначала давайте добавим туда изображение.

Шаг 4: Добавление изображений

В упражнении 4.4 мы добавим на страницу изображение с помощью элемента **img**. Более подробно это будет рассмотрено в главе 7, но на данный момент у нас есть возможность представить еще два основных понятия разметки: пустые элементы и атрибуты.

Пустые элементы

До настоящего времени все элементы, которые мы уже использовали в главной странице сайта «Бистро "Черный гусь"», следовали синтаксису, показанному на рис. 4.1: немного текстового контента в окружении открывающих и закрывающих тегов.

Несколько элементов, однако, не имеют текстового контента. Они называются *пустыми*. Элемент изображения (**img**) — один из таких — инструктирует браузер получить графический файл с сервера и вставить его в поток текста в данной позиции документа. Пустыми элементами являются также переводы строк (**br**), горизонтальные линии (**hr**) и элементы, которые предоставляют информацию о документе, но не влияют на его отображаемый контент, например элемент **meta**, использованный нами ранее. На рис. 4.11 показан синтаксис пустого элемента (сравните с рис. 4.6). Если вы пишете документ XHTML, синтаксис будет немного отличаться (см. врезку «Пустые элементы в XHTML»)

Пустые элементы в XHTML

Пустые элементы в XHTML *ограничиваются* путем добавления слеша, которому предшествует пробел, перед закрывающей угловой скобкой, например: **<img />**, **
** и **<hr />**. Ниже приведен пример перевода строки с использованием синтаксиса XHTML.

```
<p>390005, Рязань <br />ул. Электровольтная, 17</p>
```

Пример: элемент **
** вставляет перевод строки.

```
<p>390005, Рязань <br>ул. Электровольтная, 17</p>
```

Рис. 4.11. *Структура пустого элемента*

Атрибуты

Вернемся к добавлению изображения с помощью пустого элемента **img**. Очевидно, что тег **<img>** бесполезен сам по себе: нет возможности узнать, какое изображение надо использовать. И здесь на сцену выходят *атрибуты* — инструкции, которые уточняют или корректируют элемент. Для элемента **img** атрибут **src** (сокращенно от английского слова source — «источник») обязателен, он указывает расположение файла изображения посредством его URL-адреса.

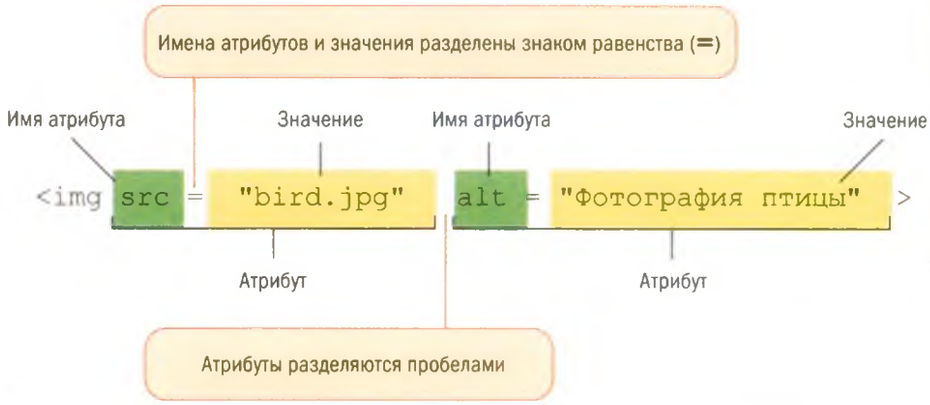


Рис. 4.12. *Элемент* **img** *с атрибутами*

Синтаксис для атрибутов следующий:

```
имя_атрибута="значение"
```

Атрибуты указываются после имени элемента и отделены от него пробелом. В элементах, не являющихся пустыми, атрибуты указываются только в открывающем теге:

```
<элемент имя_атрибута="значение">
<элемент имя_атрибута="значение">контент</элемент>
```

В элементе можно перечислить несколько атрибутов в любом порядке. Разделяйте их пробелами.

```
<элемент атрибут1="значение" атрибут2="значение">
```

Несколько иначе структура элемента **img** с обязательными атрибутами отмечена на рис. 4.12.

Необходимо знать следующее:

- Атрибуты располагаются сразу после имени элемента только в открывающем теге, и никогда в закрывающем.
- К элементу может применяться несколько атрибутов, разделенных пробелами в открывающем теге. Их порядок не важен.
- Атрибуты принимают значения, которые следуют за знаком равенства (=). В языке HTML некоторые значения атрибутов могут быть сокращены до отдельных описательных слов, например атрибут **checked**, устанавливающий флажок элемента формы при ее загрузке. В языке XHTML, однако, у всех атрибутов должны быть явные

значения (**checked="checked"**). Атрибут такого типа называют *логическим*, потому что он описывает свойство, которое либо включено, либо отключено.

- Значение может быть числом, словом, строкой текста, URL-адресом или переменной в зависимости от назначения атрибута. Вам встретятся все эти примеры далее в книге.
- Кавычки не требуются для всех значений в языке HTML, однако, они обязательны в XHTML. Многим веб-разработчикам нравится согласованность и аккуратность при использовании кавычек, даже если они пишут код на языке HTML. Одинарные или двойные кавычки одинаково приемлемы, если они используются согласованно, однако двойные кавычки приняты по соглашению. Обратите внимание, что кавычки в HTML-файлах должны быть прямые ("), а не косые (″) и не елочкой («»).
- Некоторые атрибуты, такие как **src** и **alt**, обязательны для элемента **img**.
- Имена атрибутов для каждого элемента определяются в спецификации HTML. Другими словами, нельзя самостоятельно придумать атрибут для элемента.

Теперь вы должны быть более чем готовы попробовать свои силы в добавлении элемента **img** вместе с атрибутами на страницу «Бистро "Черный гусь"» в упражнении 4.4.

УПРАЖНЕНИЕ 4.4. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЯ

1. Если вы выполняете упражнения по мере прочтения книги, сначала необходимо скопировать файл изображения на жесткий диск, чтобы страница получила к нему доступ, когда вы открываете HTML-файл. Файл изображения находится среди материалов, предоставляемых для этой главы на диске, прилагающемся к книге (*Примеры\глава-04\blackgoose.png*). Обязательно сохраните его в папку *bistro*, где находится файл *index.html*.
2. Как только вы сохранили изображение, вставьте его в начало заголовка первого уровня, добавив элемент **img** и его атрибуты, как показано ниже:

   ```
   <h1><img src="blackgoose.png" alt="Эмблема "Черного гуся"">Бистро "Черный гусь"</h1>
   ```

   Атрибут **src** указывает на имя файла изображения, который должен быть вставлен, **alt** позволяет указать текст, который должен отображаться, если изображение недоступно. Оба эти атрибута обязательны в каждом элементе **img**.
3. Мне нужно, чтобы изображение появилось над заголовком, поэтому давайте добавим перевод строки (**br**) после элемента **img**, тогда текст заголовка начнется с новой строки.

   ```
   <h1><img src="blackgoose.png" alt="Эмблема "Черного
   гуся""><br>Бистро "Черный гусь"</h1>
   ```

4. Для большей ясности разделим последний абзац на три строки. Укажите теги **
** там, где вы хотите установить переводы строк. Попытайтесь сделать так, чтобы ваша страница была похожа на изображенную на рис. 4.13.
5. Теперь сохраните файл *index.html* и откройте или обновите его в окне браузера. Страница должна выглядеть так, как показано на рис. 4.13. Если этого не происходит, убедитесь, что файл изображения *blackgoose.png* находится в том же каталоге, что и *index.html*. Если это так, убедитесь, что в элементе разметки **img** присутствуют все закрывающие кавычки или скобки.

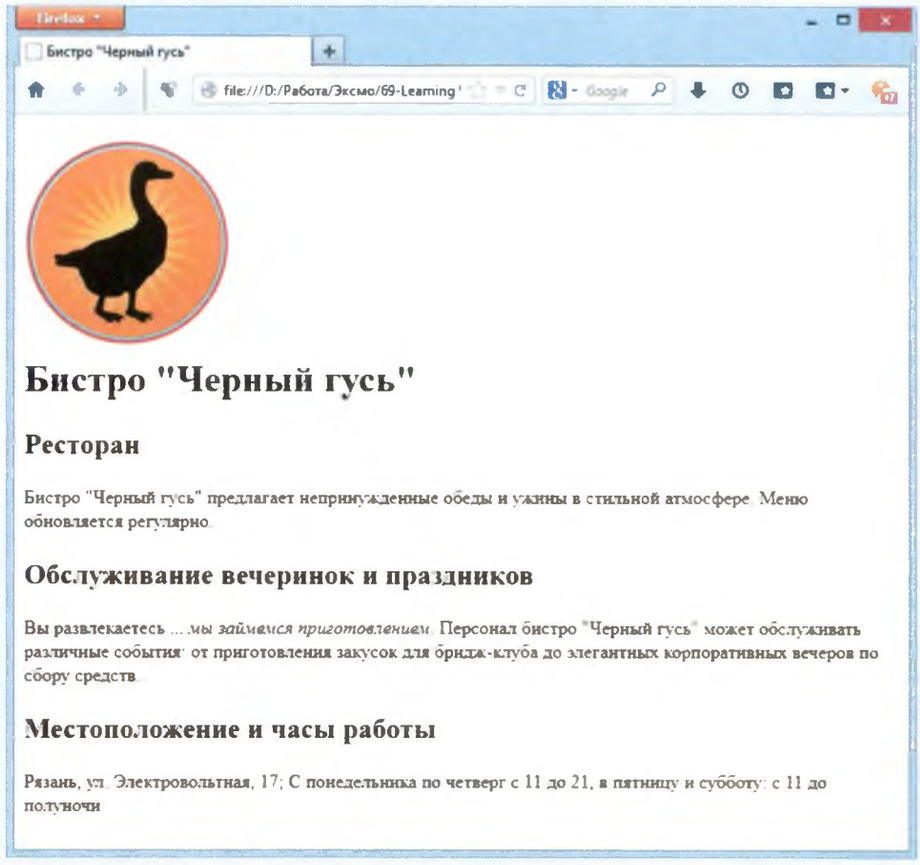


Рис. 4.13. *Главная страница сайта «Бистро "Черный гусь"» со встроенным изображением эмблемы*

Шаг 5: Изменение внешнего вида страницы с помощью CSS

В зависимости от контента и целей веб-сайта, вы можете решить, что документ представлен браузером по умолчанию вполне адекватно. Однако я думаю, что следует немного приукрасить главную страницу бистро «Черный гусь», чтобы произвести хорошее первое впечатление

на потенциальных клиентов. Говоря «приукрасить», я имею в виду, что хотела бы изменить представление страницы с помощью каскадных таблиц стилей (CSS — Cascading Style Sheets).

В упражнении 4.5 мы изменим внешний вид элементов текста и фона страницы, используя некоторые простые правила таблицы стилей. Не стремитесь понять все прямо сейчас — более подробно мы познакомимся с таблицами CSS в части III. Но я хочу по крайней мере дать вам представление о том, что значит добавить «слой» представления к структуре, которую мы создали с помощью разметки.

УПРАЖНЕНИЕ 4.5. ДОБАВЛЕНИЕ ТАБЛИЦЫ СТИЛЕЙ

1. Откройте файл *index.html*, если он еще не открыт.
2. Я собираюсь использовать элемент **style**, чтобы применить к странице очень простую глобальную таблицу стилей. (Это лишь один из способов добавления таблицы стилей; другие рассматриваются в главе 11).

   Элемент **style** помещается внутрь элемента заголовка **head** документа. Он использует обязательный атрибут, сообщающий браузеру тип информации в элементе (в настоящее время единственный вариант — **text/css**). Начнем с добавления элемента стиля документа, как показано ниже:

```
<head>
<meta charset="utf-8">
<title>Бистро "Черный гусь"</title>
<style>
</style>
</head>
```

3. Теперь введите следующие правила стиля в элемент **style** точно так же, как вы их здесь видите. Не беспокойтесь, если не представляете, что происходит (хотя это интуитивно понятно). Все о правилах стилей вы узнаете в части III.

```
<style>
body {
background-color: #faf2e4;
margin: 0 15%;
font-family: sans-serif;
}
h1 {
text-align: center;
font-family: serif;
font-weight: normal;
text-transform: uppercase;
border-bottom: 1px solid #57b1dc;
margin-top: 30px;
}
h2 {
color: #d1633c;
font-size: 1em;
}
</style>
```

4. Настало время сохранить файл и посмотреть на него в браузере. Он должен быть похож на страницу, изображенную на рис. 4.14. Если этого не произошло, убедитесь, что в коде таблицы стилей вы не пропустили точки с запятыми или фигурные скобки.

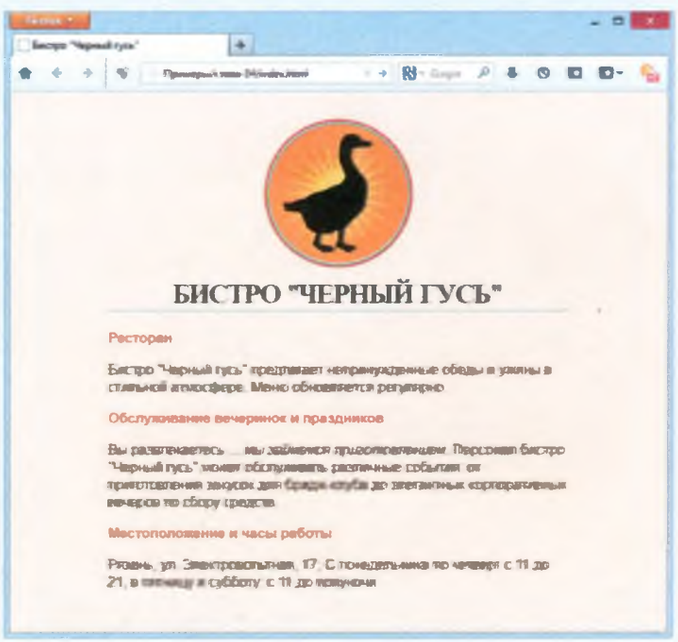


Рис. 4.14. Страница сайта «Бистро "Черный гусь"» после применения правил таблиц CSS

Мы закончили работать над страницей сайта «Бистро "Черный гусь"». Вы не только сверстали свою первую веб-страницу в комплекте с таблицей стилей, но и узнали в ходе работы об элементах, атрибутах, пустых элементах, блочных и встроенных элементах, основной структуре HTML-документа, и правильном использовании разметки. Неплохо для одной главы!

Когда хорошие страницы становятся плохими

ПРИМЕЧАНИЕ

Пропуск слеша в закрывающем теге (по сути, пропуск самого закрывающего тега) для некоторых блочных элементов, таких как заголовки или абзацы, может быть не столь драматичен. Интерпретация браузером начала нового блочного элемента означает, что предыдущий блочный элемент закончен.

Предыдущая демонстрация прошла очень гладко, однако даже для небольших объектов легко ошибиться при вводе HTML-кода вручную. К сожалению, один пропущенный символ может привести к ошибкам на всей странице. Я собираюсь внести ошибку в мою страницу, чтобы мы могли увидеть, что происходит.

Что получится, если я забуду ввести слеш (`/`) в закрывающем теге выделения шрифта (`</ em>`)? Из-за удаления всего лишь одного символа (рис. 4.15) текст остальной части документа отобразится как выделенный курсивом. Так происходит потому, что без этой черты ничто не говорит браузеру отключить форматирование текста, поэтому оно просто продолжается дальше.

Я восстановила слеш, но на этот раз, давайте посмотрим, что произошло бы, если бы я случайно пропустила закрывающую угловую скобку в конце открывающего тега `<h2>` (рис. 4.16).

Видите, что заголовка нет? Это произошло потому, что без закрывающей скобки тега, браузер предполагает, что весь последующий текст — весь, вплоть до следующей закрывающей скобки (`>`), которую он находит — является частью этого тега `<h2>`. Браузеры не отображают текст внутри тега, поэтому мой заголовок исчез. Браузер просто проигнорировал непонятное имя элемента и перешел к следующему.

Ошибки в ваших первых HTML-документах и их устранение — отличный способ обучения. Если вы пишете первые страницы идеально, я рекомендовала бы повозиться с кодом, как я показала, чтобы увидеть, как браузер реагирует на различные изменения. Это может быть очень полезно в дальнейшем при устранении ошибок на странице. Я перечислила некоторые наиболее часто встречающиеся проблемы во врезке «Возникли проблемы?». Замечу, что они характерны не только для начинающих. Что-то подобное иногда встречается даже у профессионалов.

Без угловой скобки все последующие символы будут интерпретироваться как часть длинного, неизвестного имени элемента, и слово «Ресторан» исчезнет со страницы.

```
<h2>Обслуживание вечеринок и праздников</h2>
<p>Вы развлекаетесь ... <em>мы займемся приготовлением<em>. Персонал бистро
"Черный гусь" может обслуживать различные события: от приготовления закусок для
бридж-клуба до элегантных корпоративных вечеров по сбору средств.</p>
```

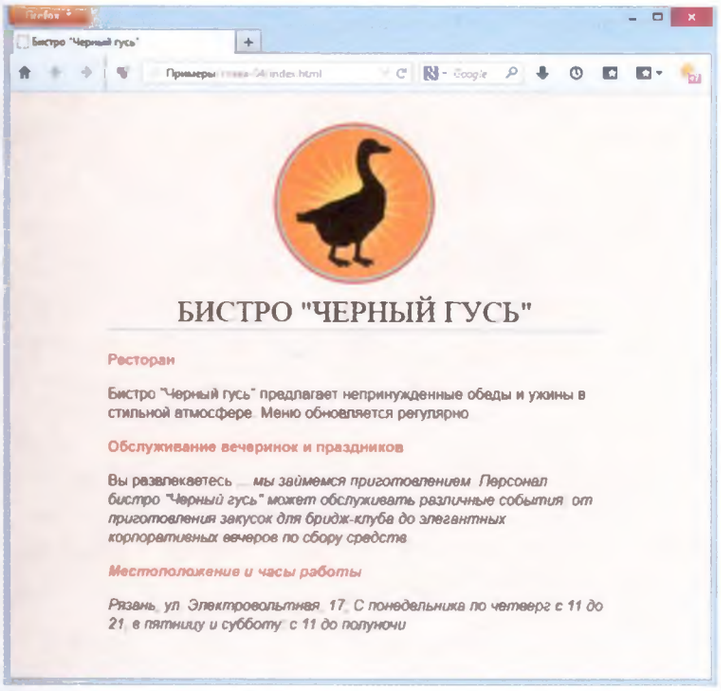


Рис. 4.15. Как видно в данном примере, когда опущен слеш, браузер не знает, где заканчивается элемент

```
<h2Ресторан</h2>
<p>Бистро "Черный гусь" предлагает непринужденные обеды и ужины в стильной
атмосфере. Меню обновляется регулярно.</p>
```

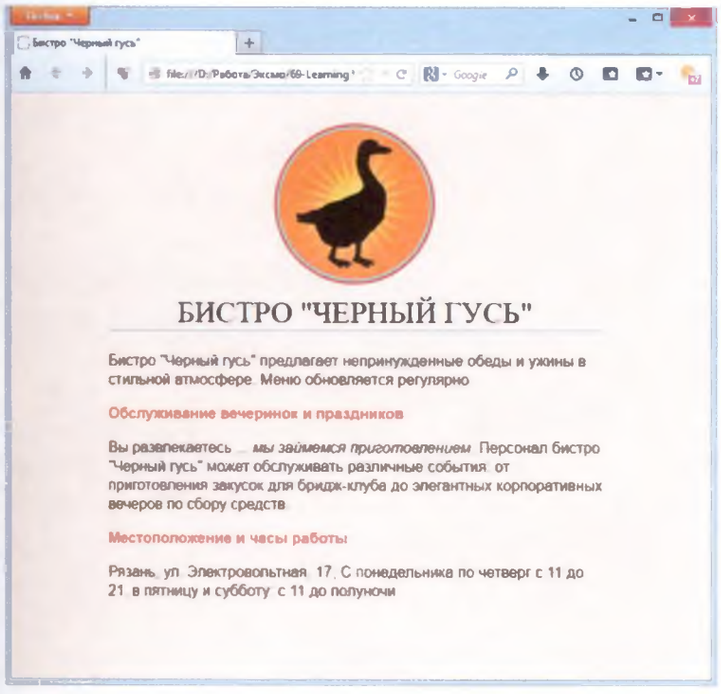


Рис. 4.16. Отсутствие закрывающей угловой скобки превращает все последующее содержимое в часть тега, и следовательно, оно не отображается

Валидация кода

Один из способов, к которому прибегают профессиональные веб-разработчики для поиска ошибок в разметке — валидация документов. Что это значит? Выполнить *валидацию* документа означает проверить разметку, чтобы убедиться, что вы соблюли все правила той версии HTML, которую использовали (существует несколько версий, их мы обсудим в главе 10).

Документы, написанные без ошибок, называют валидными. Настоятельно рекомендуется проводить валидацию документов, особенно для профессиональных сайтов. Валидные документы более единообразно отображаются в различных браузерах, они загружаются быстрее и легче доступны.

Сейчас браузеры не требуют, чтобы документы были валидными (другими словами, они сделают все возможное для их отображения вместе с ошибками), но каждый раз, отступая от стандартов, вы увеличиваете степень непредсказуемости того, как страница отображается или управляется альтернативными устройствами.

Итак, как убедиться, что документ валидный? Вы можете проверить его сами или попросить друга, но люди ошибаются, и на самом деле вы не обязаны помнить все правила спецификации до мельчайших подробностей. Вместо этого используйте *валидатор* — программное обеспечение, которое сравнивает ваш исходный код с указанной версией языка HTML. Ниже представлены некоторые моменты, проверяемые валидаторами:

- Указание объявления DOCTYPE. Без него валидатор не знает, с какой версией языка HTML или XHTML сравнивать.
- Указание кодировки набора символов документа.
- Включение необходимых правил и атрибутов.
- Нестандартные элементы.
- Несоответствие тегов.
- Ошибки вложения.
- Опечатки и другие незначительные ошибки.

Для проверки и исправления ошибок в HTML-документах разработчики используют ряд полезных инструментов. Консорциум Всемирной паутины предлагает бесплатный онлайн валидатор на сайте **validator.w3.org**. Для документов HTML5 используйте онлайн-валидатор, находящийся на сайте **html5.validator.nu**. Кроме того, валидаторы доступны в инструментах для разработчиков, таких как дополнение Firebug для браузера Firefox или встроенные инструменты разработчика в браузерах Safari и Chrome, чтобы вы могли проверить свою работу в процессе. Если для создания сайтов вы используете программу Dreamweaver, в нее также встроен валидатор.

Резюме

В этой главе вы познакомились с элементами, которые определяют структуру документа. Остальные элементы, представленные в упражнениях, более подробно будут рассматриваться в следующих главах.

Элемент	Описание
`body`	Определяет тело документа, содержащее контент
`head`	Определяет заголовок документа, содержащий информацию
`html`	Главный элемент, в котором содержатся все остальные
`meta`	Предоставляет информацию о документе
`title`	Задает название страницы

Возникли проблемы?

Ниже приведены некоторые типичные проблемы, которые возникают при создании веб-страниц и их просмотре в браузере:

Я изменил свой документ, но когда обновил страницу в браузере, она выглядит точно так же.

Это может произойти, если вы не сохранили документ до обновления в браузере или сохранили его в другой папке.

Половина моей страницы исчезла.

Так бывает, если не хватает закрывающей скобки (>) или кавычек внутри тега. Это распространенная ошибка при верстке HTML-кода вручную.

Я добавил изображение с помощью элемента `img`, но появляется только искаженный графический значок.

Неправильное отображение рисунка может быть обусловлено разными факторами. Во-первых, это может означать, что браузер не находит графический файл. Убедитесь, что URL-адрес файла написан правильно. (Мы обсудим URL-адреса далее, в главе 6.) Проверьте, находится ли файл изображения в том каталоге, который вы указали. Если файл на месте, убедитесь, что его формат — один из тех, которые могут отображать веб-браузеры (GIF, JPEG или PNG), и что он имеет соответствующее расширение (соответственно *.gif*, *.jpeg* или *.jpg*, или *.png*).

ГЛАВА 5

РАЗМЕТКА ТЕКСТА

Как только контент готов, и добавлена разметка для структуризации документа (**html**, **head**, **title** и **body**), самое время определить элементы контента. В данной главе представлены элементы, которые применяются для разметки текстового контента. Их не так много, как можно подумать, к тому же вы будете регулярно использовать далеко не все. Несмотря на это данная глава объемна и рассматривает множество вопросов.

Прежде чем мы перейдем к перечислению, не помешает лишний раз напомнить, как важно выбирать элементы семантически, то есть таким образом, чтобы наиболее точно описать значение контента. Если вам не нравится, как выглядит контент, измените его с помощью таблиц стилей. Семантически размеченный документ обеспечивает доступность вашего контента в широком диапазоне сред просмотра, от настольных компьютеров и ноутбуков до смартфонов и программ экранного доступа. Такая разметка также позволяет машинам, например программным системам поиска и индексации, правильно анализировать контент и принимать решения об относительной важности элементов на странице.

Помня все эти принципы, приступим к работе с текстовыми HTML-элементами, начав с самого простого из них — элемента абзаца.

В этой главе

- Выбор оптимального элемента для контента
- Абзацы и заголовки
- Три типа списков
- Организация контента по разделам
- Элементы текстового уровня (встроенные)
- Универсальные элементы **div** и **span**
- Специальные символы

ВАЖНОЕ ПРИМЕЧАНИЕ

Мы будем изучать разметку в соответствии со стандартом HTML5, поддерживаемым консорциумом Всемирной паутины (**www.w3.org/TR/html5/**), каким он был на момент написания книги. Существует еще одна «живая» (и не зарегистрированная) версия языка HTML без номера, поддерживаемая сообществом WHATWG (**whatwg.org**), и представляющая собой почти то же самое, но обычно с небольшими отличиями. Я обязательно укажу элементы и атрибуты, которые принадлежат только одной из спецификаций. Обе они регулярно меняются, поэтому я призываю вас уточнить на перечисленных сайтах, были ли добавлены или исключены какие-либо элементы.

Возможно, вы слышали, что не все браузеры поддерживают элементы языка HTML5. Это правда. Но подавляющее большинство элементов HTML5 использовались десятилетиями в более ранних версиях языка HTML, поэтому они поддерживаются повсеместно. Элементы, появившиеся в HTML5, могут поддерживаться не полностью или вовсе не поддерживаться. Здесь они выделены маркером: Новый в HTML5. Так что, если я четко не указываю на проблемы поддержки, вы можете считать, что описания и примеры разметки, представленные здесь, будут работать во всех браузерах.

Абзацы

`<p>...</p>`

Элемент абзаца

Абзацы (параграфы) — это самые первичные элементы текстового документа. Вы указываете абзац с помощью элемента **p**, вставляя открывающий тег ** ** в начале и закрывающий тег ** ** в конце, как показано в следующем примере.

```
<p>В шрифтах семейства Serif имеются небольшие засечки на концах букв. В общем случае шрифты семейства Serif могут повысить удобочитаемость больших объемов текста.</p>

<p>Шрифты Sans-Serif не имеют засечек на концах букв, их окончания ровные. Шрифты Helvetica и Arial — это примеры шрифтов семейства sans-serif. В общем случае шрифты семейства sans-serif кажутся более сглаженными и более современными.</p>
```

Графически браузеры почти всегда отображают абзацы с новой строки, по умолчанию оставляя между ними небольшое пустое пространство (используя терминологию CSS можно сказать, они отображаются *блоком*). Абзацы могут содержать текст, изображения и другие встроенные элементы (называемые *строчным контентом* в спецификации HTML5), однако они не могут содержать заголовки, списки, элементы разделов или любые другие, обычно отображающиеся блоками по умолчанию.

ПРИМЕЧАНИЕ

Вы должны назначить элементы для всего текста в документе. Другими словами, каждая его часть должна быть заключена в теги. Текст, не ограниченный ими, называется «неопределенным», и содержащий его документ не пройдет проверку валидности. Для получения дополнительной информации о проверке валидности документов обращайтесь к главе 4.

В языке HTML приемлемо опустить закрывающий тег ** **. Браузер просто решит, что абзац закрыт, когда дойдет до следующего блочного элемента. Однако в более строгом синтаксисе XHTML закрывающий тег обязателен (что неудивительно). Из соображений последующей совместимости, многие веб-разработчики, и я в том числе, предпочитают закрывать элемент абзаца, впрочем, как и все остальные элементы. Я рекомендую тем, кто только изучает разметку, поступать так же.

Заголовки

`<h1>...</h1>`
`<h2>...</h2>`
`<h3>...</h3>`
`<h4>...</h4>`
`<h5>...</h5>`
`<h6>...</h6>`

Элементы заголовка

В предыдущей главе мы использовали элементы **h1** и **h2**, чтобы выделить заголовки для страницы сайта «Бистро "Черный гусь"». Фактически в HTML существует шесть уровней заголовков, от **h1** до **h6**. Когда вы добавляете к контенту заголовки, браузер использует их для формирования *структуры документа* страницы. Вспомогательные устройства, такие как программы экранного доступа, используют структуру документа, чтобы помочь пользователям быстро просматривать страницу и перемещаться по ней. Кроме того, поисковые системы рассматривают уровни заголовка как часть написанных для них алгоритмов (информация в заголовках высших уровней имеет больший вес). В силу этих причин рекомендуется начать с заголовка 1-го уровня (**h1**) и двигаться вниз по порядку (см. примечание), создавая логическую иерархию или структуру документа.

Этот пример демонстрирует разметку для четырех уровней заголовков. Дополнительные будут отмечены подобным образом.

```
<h1>Дизайн шрифтов</h1>

<h2>Serif</h2>

<p>В шрифтах семейства Serif имеются небольшие засечки на концах букв. В общем случае шрифты семейства Serif могут повысить удобочитаемость больших объемов текста.</p>

<h3>Baskerville</h3>

<h4>Описание</h4>

<p>Описание шрифта Baskerville.</p>

<h4>История</h4>

<p>История шрифта Baskerville.</p>

<h3>Georgia</h3>

<p>Описание и история шрифта Georgia.</p>

<h2>Sans-Serif</h2>

<p>Шрифты Sans-Serif не имеют засечек на концах букв</p>
```

Разметка в этом примере создает следующую структуру документа:

1. Дизайн шрифтов
    1. Serif
    + абзац текста
        1. Baskerville
            1. Описание
            + абзац текста
            2. История
            + абзац текста
        2. Georgia
        + абзац текста
    2. Sans-Serif
    + абзац текста

По умолчанию заголовки в этом примере отображаются полужирным начертанием, начиная с очень крупного шрифта для заголовков **h1** и используя более мелкий шрифт для каждого последующего заголовка, как показано на рис. 5.1. Изменить их внешний вид можно с помощью таблицы стилей.

ПРИМЕЧАНИЕ

В спецификации HTML5 используется новая система структуризации, опирающаяся не только на заголовки. Подробнее об этом читайте во врезке «Элементы разделов» далее в этой главе.

ПРИМЕЧАНИЕ

Все примеры (и, соответственно, иллюстрации для книги) в данной книге выполнялись в браузере Firefox на компьютере под управлением операционной системы Windows 8, если не указано иное.

h1 **Дизайн шрифтов**

h2 **Serif**

В шрифтах семейства Serif имеются небольшие засечки на концах букв. В общем случае, шрифты семейства Serif могут повысить удобочитаемость больших объемов текста.

h3 **Baskerville**

h4 **Описание**

Описание шрифта Baskerville.

h4 **История**

История шрифта Baskerville.

h3 **Georgia**

Описание и история шрифта Georgia.

h2 **Sans-Serif**

Шрифты Sans-Serif не имеют засечек на концах букв

Рис. 5.1. Представление по умолчанию четырех уровней заголовков

Обозначение новой темы

<hr>

Горизонтальная линия

Если вы хотите указать, что одна тема или мысль завершена, и начинается другая, можете вставить в HTML5 так называемый *тематический разрыв* на уровне абзаца с помощью элемента **hr**. Он добавляет разделитель между логическими разделами или абзацами текста без введения нового уровня заголовка.

В спецификациях HTML до версии HTML5 элемент **hr** определен как «горизонтальная линия», так как именно это он добавляет на страницу. Браузеры по-прежнему отображают **hr** как затененную горизонтальную линию, размещенную на отдельной строке, по умолчанию оставляя свободное пространство сверху и снизу, но сейчас он обретает новое семантическое значение. Если вам нужна только декоративная линия, то лучше создайте правило, указав цветные границы до или после выбранного элемента с помощью CSS.

Элемент **hr** — пустой, вы лишь помещаете его в позицию, в которой хотите указать смену темы, как показано в этом примере и на рис. 5.2. Обратите внимание, что в языке XHTML элемент **hr** должен быть закрыт слешем: **<hr />**.

```
<h3>Times</h3>
<p>Описание и история шрифта Times.</p>
<hr>
<h3>Georgia</h3>
<p>Описание и история шрифта Georgia.</p>
```

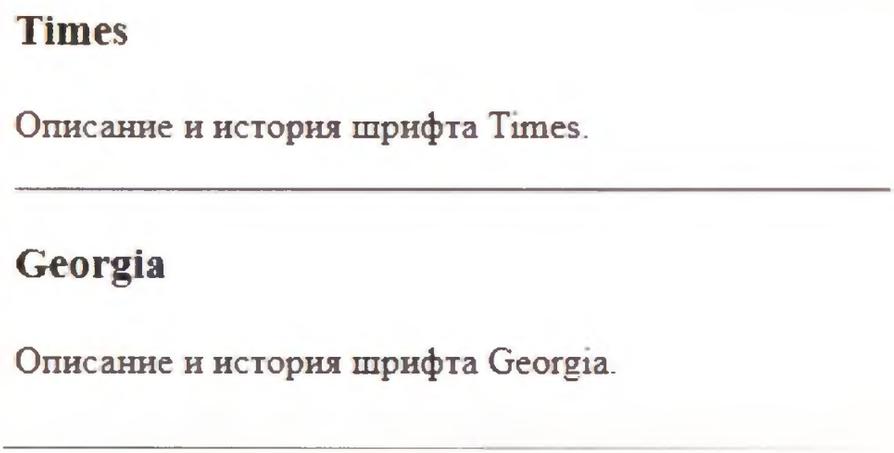


Рис. 5.2. Представление по умолчанию горизонтальной черты

Группы заголовков

<hgroup> ... </ hgroup>

Группа составных заголовков

Новый в HTML5

Часто заголовки снабжены поясняющими подзаголовками или ключевыми фразами. Возьмем, например, следующий заголовок:

Создание простой страницы

(обзор HTML)

В прошлом разметка нескольких составных заголовков и подзаголовков была несколько проблематичным занятием. В первой строке, «Создание простой страницы», явно используется элемент **h1**, но если применить ко второй строке элемент **h2**, можно случайно ввести новый уровень в структуру документа. Лучшим вариантом было разметить его как абзац, но в этом нет семантического значения.

По этой причине для определения нескольких заголовков как группы* в HTML5 используется элемент **hgroup.** Браузеры, поддерживающие элемент **hgroup**, знают, что в структуре нужно учитывать только за-

* Хотя элемент **hgroup** потенциально полезен, его будущее неясно. Если вы хотите использовать этот элемент на публикуемом сайте, сначала изучите спецификацию HTML5.

ОСОБЕННОСТИ ПОДДЕРЖКИ

Элемент **hgroup** не поддерживается браузером Internet Explorer версии 8* и более ранних (для поиска решения проблемы, прочитайте врезку «Поддержка элементов языка HTML5 в браузере Internet Explorer»).

головок первого уровня, а остальные — игнорировать. Ниже показано, как можно использовать элемент **hgroup** для разметки заголовка. Тогда в структуре документа будет представлен только элемент **h1**, «Создание простой страницы».

```
<hgroup>
<h1>Создание простой страницы</h1>
<h2>(обзор HTML)</h2>
</hgroup>
```

Списки

Для человека естественно составлять списки, а язык HTML предоставляет элементы для разметки трех типов:

- **Маркированный список.** Набор пунктов, очередность которых неважна.
- **Нумерованный список.** Перечень, в котором последовательность элементов имеет значение.
- **Список определений.** Состоит из пар «имя/значение», в том числе терминов и определений.

Все элементы списков, собственно списки и тексты пунктов являются блочными по умолчанию, поэтому они начинаются с новой строки, а над элементом и под ним остается пустое пространство, но это можно изменить с помощью CSS. В данном разделе мы подробно рассмотрим каждый вид списка.

Маркированные списки

`<ul>...</ul>`

Маркированный список

`<li>...</li>`

Пункты внутри маркированного списка

Практически любой список примеров, имен, компонентов, мыслей или вариантов квалифицируется как *маркированный* или *неупорядоченный*. Большинство списков попадают в эту категорию. По умолчанию маркированные списки отображаются с пометкой в виде точки перед каждым элементом, но с помощью таблиц стилей, как вы увидите далее, можно мгновенно изменить форматирование.

Чтобы определить маркированный список, отметьте его как элемент **ul**. Открывающий тег **
** вводится до первого пункта списка, а закрывающий тег **
** — после последнего. Затем, заключая каждую часть в открывающий и закрывающий теги **li**, мы помечаем ее в качестве элемента списка (**li**), как показано в этом примере. Обратите внимание, что в исходном документе нет маркеров списка. Они автоматически добавляются в браузере (рис. 5.3).

* Версия 8 браузера Internet Explorer все еще популярна среди пользователей (ок. 12,5% на момент написания книги, согласно сайту gs.statcounter.com)

```
<ul>
<li><a href="">Serif</a></li>
<li><a href="">Sans-serif</a></li>
<li><a href="">Script</a></li>
<li><a href="">Display</a></li>
<li><a href="">Dingbats</a></li>
</ul>
```

- Serif
- Sans-serif
- Script
- Display
- Dingbats

Рис. 5.3. Вид по умолчанию примера маркированного списка. Маркеры добавляются браузером автоматически

Теперь кое-что более интересное. Мы можем взять ту же разметку маркированного списка и радикально изменить его внешний вид, применяя различные таблицы стилей, как показано на рис. 5.4. Как видите, я убрала маркеры списка, расположила пункты по горизонтали и даже сделала их похожими на графические кнопки. Разметка осталась прежней.

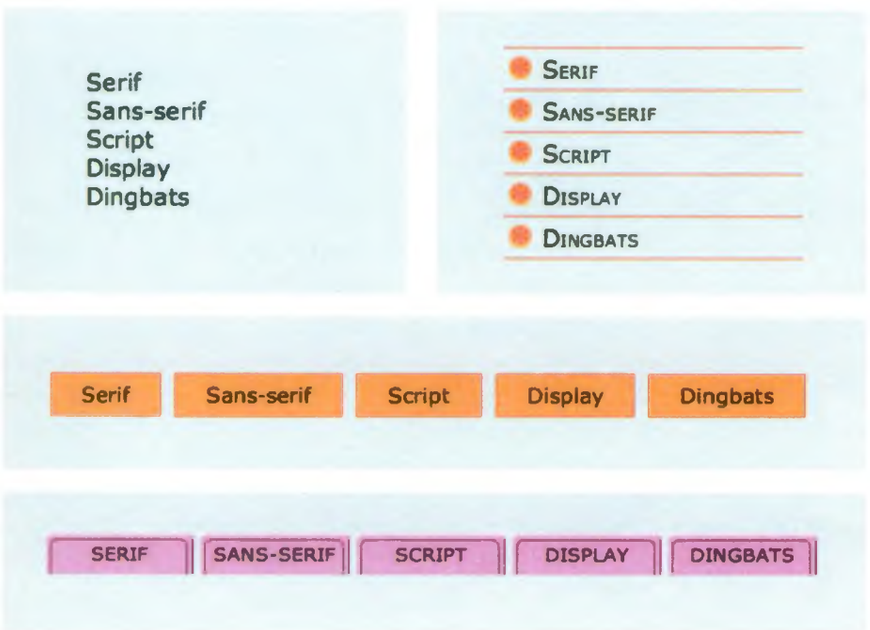


Рис. 5.4. Используя таблицы стилей, мы можем придать маркированному списку самый различный вид

ПРИМЕЧАНИЕ

Единственное, что разрешено в пределах разметки маркированного списка (то есть между открывающим и закрывающим тегами `ul`), — это один или несколько элементов списка. Вы не можете вставить туда другие элементы, и там не может быть неразмеченного текста. Однако допустимо поместить любой элемент потокового контента внутрь элемента списка (`li`).

Вложенные списки

Любой список может быть вложен в другой. Следующий пример демонстрирует структуру маркированного списка, вложенного во второй пункт нумерованного.

```
<ol>
<li></li>
<li>
<ul>
<li></li>
<li></li>
<li></li>
</ul>
</li>
</ol>
```

При вложении одного маркированного списка в другой браузер автоматически меняет стиль маркера для списка второго уровня. К сожалению, при вложении нумерованных списков стиль нумерации не меняется по умолчанию. Вам необходимо настроить это самостоятельно с помощью таблиц стилей.

Нумерованные списки

`<ol>...</ol>`

Нумерованный список

`<li>...</li>`

Список элементов внутри нумерованного списка

Нумерованные или *упорядоченные* списки предназначены для элементов, которые следуют в определенном порядке, например пошаговые инструкции. Они функционируют так же, как и описанные ранее маркированные списки, за исключением того, что определены с помощью элемента **ol**. Вместо маркеров браузер автоматически указывает номера перед элементами списка, поэтому вам не нужно проставлять их в исходном документе. Это позволяет легко менять местами элементы списка без повторной нумерации.

Элементы нумерованного списка должны содержать несколько элементов списка, как показано в следующем примере и на рис. 5.5:

```
<ol>
<li>Гутенберг развивает подвижную печать (1450)</li>
<li>Появляется линотип (1890)</li>
<li>Фотонабор завоевывает популярность (1950)</li>
<li>Печать становится цифровой (1980)</li>
</ol>
```

1. **Гутенберг развивает подвижную печать** (1450)
2. **Появляется линотип** (1890)
3. **Фотонабор завоевывает популярность** (1950)
4. **Печать становится цифровой** (1980)

Рис. 5.5. Вид по умолчанию нумерованного списка. Номера добавляются браузером автоматически

Если вы хотите, чтобы список начинался с номера, отличного от 1, следует указать это при помощи атрибута **start**, как показано в следующем примере:

```
<ol start="17">
<li>Выделите текст с помощью инструмента выбора текста.</li>
<li>Выберите символ табуляции.</li>
<li>Выберите начертание символа из всплывающего меню.</li>
</ol>
```

Элементы получившегося списка будут пронумерованы: 17, 18 и 19 соответственно.

Изменение маркеров и нумерации списка

Для изменения маркеров и номеров списков стоит использовать свойство **list-style-type** таблицы стилей. Например для маркированных списков вы можете изменить форму маркера с точки на квадрат или незаштрихованный круг, на собственное изображение или удалить маркер полностью. Для нумерованных списков — заменить традиционные арабские цифры римскими (I., II., III. или i., ii., iii.), буквами (A., B., C., или a., b., c.) и рядом других типов нумерации. Фактически, если список семантически размечен, в нем вообще не обязательно отображать маркеры или числа. Изменение стиля списков с помощью CSS рассматривается в главе 12.

Списки определений

Списки определений используются для формирования любого типа пар «имя/значение», таких как термины и их значения, вопросы и ответы, прочие виды понятий и связанной с ними информации. Этот формат немного отличается от двух других типов. Весь список размечен с помощью элемента **dl**. Его содержимое — это несколько элементов **dt**, обозначающих имена, и **dd**, обозначающих соответствующие им значения. Ниже представлен пример списка определений (рис. 5.6).

```
<dl>
<dt>Линотип</dt>
<dd>Отливка строк позволила набирать и применять шаблон строки, который затем многократно использовался в машине. Это достижение резко увеличило скорость набора и печати.</dd>
<dt>Фотонабор</dt>
<dd>Гарнитуры шрифтов хранятся на пленке, а затем проецируются на фотобумагу. Линзы регулируют размер печати.</dd>
<dt>Цифровая печать</dt>
<dd><p> Цифровые гарнитуры шрифтов сохраняют контур формы шрифта в форматах, подобных Postscript. Для вывода контур может масштабироваться до любого размера.</p>
<p> Postscript стал стандартом, благодаря поддержке графики и давнего использования на компьютерах Mac и лазерных принтерах Apple.</p>
</dd>
</dl>
```

Линотип
: Отливка строк позволила набирать и использовать шаблон строки, который затем многократно использовался в машине. Это достижение резко увеличило скорость набора и печати.

Фотонабор
: Гарнитуры шрифтов хранятся на пленке, а затем проецируются на фотобумагу. Линзы регулируют размер печати.

Цифровая печать
: Цифровые гарнитуры шрифтов сохраняют контур формы шрифта в форматах, подобных Postscript. Для вывода контур может масштабироваться до любого размера.

  Postscript стал стандартом, благодаря поддержке им графики и его давней поддержки на компьютерах Mac и лазерных принтерах Apple.

Рис. 5.6. Вид по умолчанию списка определений. Они отделяются от терминов отступами

<dl>...</dl>

Список определений

<dt>...</dt>

Имя — например термин или метка

<dd>...</dd>

Значение — например описание или определение

Корневые элементы разделов

Элемент **blockquote** относится к категории так называемых *корневых элементов разделов*, заголовки которых не входят в основную в структуру документа. Это означает, что в пределах элемента **blockquote** можно создавать сложную иерархию заголовков и не волноваться о том, как она повлияет на общую структуру документа. Другие корневые элементы разделов — **figure**, **details**, **fieldset** (для организации полей формы), **td** (ячейки таблицы) и **body** (так как у него есть собственная структура, которая одновременно является структурой документа).

Элемент **dl** содержит только элементы **dt** и **dd**. Считается правильным, когда один термин имеет несколько определений, и наоборот. Вы не можете поместить блочные элементы (например, абзацы) внутрь термина (**dt**), но определения (**dd**) могут содержать любой элемент потокового контента.

Другие элементы контента

Мы рассмотрели абзацы, заголовки и списки, но существует еще несколько специальных элементов текста, не вписывающихся в определенные категории: длинные цитаты (**blockquote**), предварительно отформатированный текст (**pre**) и рисунки (**figure** и **figcaption**). Все они считаются «элементами группировки контента» в спецификации HTML5 (наряду с элементами **p**, **hr**, списками и универсальным элементом **div**, который мы рассмотрим далее в этой главе).

Другая общая черта — то, что браузеры обычно отображают их по умолчанию как блочные элементы.

Длинные цитаты

<blockquote>...</blockquote >

Длинная блочная цитата

Если нужно отформатировать длинную цитату, рекомендацию или фрагмент текста из другого источника, особенно если он занимает четыре строки и больше, вы должны разметить такой текст элементом **blockquote**. Рекомендуется, чтобы в содержимом **blockquote** использовались другие элементы, — абзацы, заголовки или списки — как показано в следующем примере (см. врезку «Корневые элементы разделов»).

```
<p>Известный дизайнер шрифтов, Мэтью Картер рассказывает
о своей профессии:</p>

<blockquote>

<p>Наш алфавит не менялся веками, существует не так много
свободы в том, что дизайнер может сделать с отдельными буквами.</p>

<p> В этом много сходства с партитурой классического музыкального произведения. Не искажая того, что записано нотами, каждый дирижер, тем не менее, интерпретирует эту партитуру по-своему, и варианты прочтения могут заметно различаться.</p>

</blockquote>
```

ПРИМЕЧАНИЕ

Существует также встроенный элемент **q** для коротких цитат в потоке текста. Я расскажу о нем позже в этой главе.

На рис. 5.7 показано представление по умолчанию примера цитирования. Форматирование можно изменить с помощью CSS.

Известный дизайнер шрифтов, Мэтью Картер рассказывает о своей профессии:

> Наш алфавит не менялся веками, существует не так много свободы в том, что дизайнер может сделать с отдельными буквами.
>
> Подобно отрывку из классического музыкального произведения, чья партитура записана, - это не означает ее искажение, - и тем не менее, каждый дирижер интерпретирует эту партитуру по-разному. Интерпретации могут быть противоречивыми.

Рис. 5.7. Вид по умолчанию элемента **`blockquote`**

Предварительно отформатированный текст

`<pre>...</pre>`

Предварительно отформатированный текст

В предыдущей главе вы узнали, что браузеры игнорируют некоторые символы, например переводы строк и пробелы между символами в исходном документе. Но для определенных типов информации, таких как примеры кода или стихи, пробельные символы имеют важное по смыслу значение. Для этих целей используется элемент предварительно отформатированного текста (**`pre`**). Только он позволяет отображать все точно так, как напечатано в исходном документе, в том числе переводы строк и множественные символы табуляции. По умолчанию шрифт предварительно отформатированного текста отображается как *моноширинный*, например Courier.

Элемент **`pre`** в этом примере выглядит, как показано на рис. 5.8. Для сравнения во второй части рисунка представлен контент, размеченный как абзац с помощью элемента (**`p`**).

```
<pre>
Это –        пример
      текста с                        большим количеством
            странных
                  пробелов.
</pre>
<p>
Это –        пример
      текста с                        большим количеством
            странных
                  пробелов.
</p>
```

ПРИМЕЧАНИЕ

Свойство **`white-space:pre`** каскадных таблиц стилей также может быть использовано для сохранения пробелов и возвратов каретки в исходном коде. В отличие от элемента **`pre`**, шрифт текста, отформатированного с помощью свойства **`white-space`**, не отображается как моноширинный.

```
Это —          пример
      текста с                                          большим количеством
            странных
                  пробелов.
```

```
Это — пример текста с большим количеством странных пробелов.
```

Рис. 5.8. Предварительно форматированный текст уникален тем, что браузер отображает пробелы именно так, как они введены в исходном документе. Сравните его с элементом абзаца, в котором игнорируются переводы строки и пробелы

Рисунки

<figure>...</figure>

Связанное с контентом изображение или ресурс

Новый в HTML5

<figcaption>...</figcaption>

Подпись под рисунком

Новый в HTML5

Элемент **figure** используется для обозначения контента, иллюстрирующего или поддерживающего определенную идею текста. Элемент **figure** может содержать изображение, видеоролик, фрагмент кода, текст или даже таблицу — почти все, что может встретиться в потоке веб-контента и должно восприниматься как автономная единица. Это означает, что, если удалить рисунок с его исходного местоположения в потоке (во врезку или в приложение, например), смысл рисунка и основного потока контента должен сохраниться.

Конечно, можно просто добавить изображение в текст, однако заключение в теги **figure** четко указывает его назначение. Кроме того, вы можете применять специальные стили к таким рисункам, но не к другим изображениям на странице.

```
<figure>
<img src="piechart.png" alt="Диаграмма, показывающая использование шрифтов на мобильных устройствах">
</ figure>
```

Подпись можно присоединить к рисунку с помощью дополнительного элемента **figcaption**, разместив ее выше или ниже содержимого элемента **figure**.

```
<figure>
<pre><code>
body {
background-color: #000;
color: red;
}
</code></pre>
<figcaption>
Образец правила CSS.
</figcaption>
</figure>
```

ОСОБЕННОСТИ ПОДДЕРЖКИ

Элементы **figure** и **figcaption** не поддерживаются в браузере Internet Explorer версии 8 и более ранних (см. врезку «Поддержка элементов языка HTML5 в браузере Internet Explorer»).

УПРАЖНЕНИЕ 5.1. РАЗМЕТКА ТЕКСТА РЕЦЕПТА

Владельцы бистро «Черный гусь» решили запустить блог, где они будут делиться рецептами и публиковать объявления. В упражнениях этой главы мы поможем им разметить контент.

Ниже дан исходный текст веб-страницы с рецептом. Вы должны решить, какой элемент семантически лучше всего подходит для каждого блока контента. Используйте абзацы, заголовки, списки и хотя бы один из специальных элементов контента.

Можете писать теги прямо на этой странице. Если же вы хотите использовать текстовый редактор и посмотреть результаты в браузере, этот текстовый файл находится на диске, прилагающемся к книге, в файле *Примеры\глава-05\ex5-1_recipe.txt*. Полученный код приводится в приложении А.

```
Сельдь под шубой

"Сельдь под шубой" — популярный в России салат из сельди и овощей. Как я прочита-
ла во Всемирной паутине свое название салат получил из-за рецепта, согласно кото-
рому мелко нарезанное филе из сельди вместе с кольцами репчатого лука укладывается на
плоское блюдо и последовательно покрывается слоями из вареного картофеля, моркови,
и свеклы. После этого сверху наносится майонез. Готовый салат можно украсить зеленью
и тертым вареным желтком.

Ингредиенты

Две сельди средних размеров

Две свеклы средние

Две моркови

Две картофелины

Луковица

Майонез

Способ приготовления:

Сельдь освободить от костей, порезать на куски. Картофель, морковь и свеклу отва-
рить, остудить, очистить. Натереть на крупной терке овощи. Лук порезать кольцами.
Уложить на блюдо слоями в такой последовательности: сельдь, лук, картофель, морковь,
свекла, картофель, морковь, свекла, промазывая каждый слой овощей (кроме лука) майо-
незом. Сверху по желанию украсить зеленью и тертым вареным желтком. Поставить в хо-
лодильник на 2-3 часа для пропитки.
```

В упражнении 5.1 у вас появится возможность самостоятельно разметить документ и попытаться применить описанные выше основные текстовые элементы.

Организация контента страницы

До сих пор элементы, которые мы рассматривали, управляли определенными фрагментами контента: абзацем, заголовком, рисунком и т. д. До появления спецификации HTML5 не существовало возможности сгруппировать эти фрагменты в более крупные части, кроме как заключив их в универсальный общий элемент **(div)** (более подробно я рас-

Поддержка элементов языка HTML5 в браузере Internet Explorer

Большинство современных браузеров поддерживают новые семантические элементы языка HTML5, а для тех, которые этого не делают, достаточно создать правило стилей, инструктирующее браузеры форматировать каждый элемент как блочный, чтобы заставить их работать правильно.

```
section, article, nav, aside, header,
footer,

hgroup { display: block; }
```

К сожалению, такое исправление не будет работать в программе Internet Explorer версии 8 и более ранних (начиная с версии 9 этот браузер полностью поддерживает данные элементы). Ранние версии браузера Internet Explorer не только не распознают элементы, но и игнорируют любые стили, применяемые к ним. Решением является использование кода JavaScript для создания каждого элемента, чтобы браузер Internet Explorer «знал», что элемент существует, и позволял применять к нему вложение и стили. Ниже представлен код команды на языке JavaScript для создания элемента **section**:

```
document.createElement("section");
```

К счастью, Реми Шарп разработал сценарий, который создает все элементы HTML5 для браузера Internet Explorer за один прием. Он называется «HTML5 Shiv» (или Shim) и доступен на сервере Google, поэтому вы можете указать ссылку на него в своих документах. Чтобы убедиться, что новые элементы HTML5 функционируют должным образом в браузере Internet Explorer 8 и более ранних версий, скопируйте код в раздел заголовка документа и с помощью таблицы стилей обозначьте новые элементы как блочные.

```
<!--[if lt IE 9]>

<script src="http://html5shiv.googlecode.
com/svn/

trunk/html5-els.js"></script>

<![endif]-->
```

Найти подробную информацию о HTML5 Shiv можно по адресу **html5doctor.com/how-to-get-html5-working-in-ie-and-firefox-2/**.

HTML5 Shiv также является частью сценария полизаполнений Modernizr, добавляющего функциональность HTML5 и CSS3 в старых, не поддерживающих их браузерах. Подробнее об этом написано на сайте **modernizr.com**. Данная тема также обсуждается в главе 22.

скажу о нем позже). В языке HTML5 были введены новые элементы, придающие семантическое значение разделам обычной веб-страницы или приложения, включая разделы (**section**), статьи (**article**), навигацию (**nav**), косвенно связанный контент (**aside**), верхние (**header**) и нижние (**footer**) колонтитулы. Они созданы на основе исследования корпорации Google, составившей список 20 лучших имен элементов для разделения контента (**code.google.com/webstats/2005-12/classes.html**).

Элементы, обсуждаемые в этом разделе, полностью поддерживаются текущими версиями настольных и мобильных браузеров, но известна проблема с версией браузера Internet Explorer 8 и более старыми. Чтобы найти решение, смотрите врезку «Поддержка элементов языка HTML5 в браузере Internet Explorer».

Разделы и статьи

Длинные документы проще воспринимать, когда их контент разделен на более мелкие части.

Например книги делятся на главы, в газетах есть разделы местных новостей, спортивная колонка, юмор и т. д. Чтобы разделить длинные веб-документы на тематические секции, используйте элемент **section**. У разделов обычно имеется заголовок (внутри элемента **section**) и любой другой контент, для группировки которого есть разумная причина.

Существует множество применений элемента **section**, от деления целой страницы на основные разделы до определения тематических разделов в рамках одной статьи.

В следующем примере документ с информацией о ресурсах по типографике был разделен на две части по типу ресурсов.

```
<section>
<h2>Книги по типографике</h2>
<ul>
<li>...</li>
</ul>
</section>
<section>
<h2>Онлайн-пособия</h2>
<p>Это лучшие учебные пособия во Всемирной паутине.</p>
<ul>
<li>...</li>
</ul>
</section>
```

Используйте элемент **article** для независимых работ, которые могут использоваться отдельно или многократно в ином контексте (например, при синдикации). Это полезно для журнальных и газетных статей, сообщений в блогах, комментариев или других элементов, которые можно извлечь для внешнего использования. Можете представить его как специализированный элемент разделения, отвечающий утвердительно на вопрос: «Может ли этот контент появиться на другом сайте и не потерять свой смысл?»

<section>...</section>

Тематическая группа контента

Новый в HTML5

<article>...</article>

Автономная композиция для многократного использования

Новый в HTML5

ПРИМЕЧАНИЕ

Спецификация HTML5 рекомендует в случаях группировки элементов «для красоты» использовать универсальный элемент **div**.

Чтобы повысить интерес, можно разбить длинный элемент **article** на несколько разделов, как показано здесь:

```
<article>
<h1>Знакомство со шрифтом Helvetica</h1>
<section>
<h2>история возникновения шрифта Helvetica</h2>
<p>…</p>
</section>
<section>
<h2>Helvetica сегодня</h2>
<p>…</p>
</section>
</article>
```

И наоборот, элемент **section** веб-документа может состоять из нескольких статей.

```
<section id="essays">
<article>
<h1>Свежий взгляд на шрифт Futura</h1>
<p>…</p>
</article>
<article>
<h1>Знакомимся со шрифтом Humanist</h1>
<p>…</p>
</article>
</section>
```

Элементы **section** и **article** легко перепутать, особенно потому, что их можно вкладывать друг в друга. Помните, что, если контент автономный и может появляться вне данного контекста, его лучший разметить как **article**.

Элементы разделов

Еще одна скрытая общая черта элементов **section** и **article** — то, что оба они, относятся к так называемым *элементам разделов* спецификации HTML5. Когда браузер встречает элемент раздела, он автоматически создает новый пункт в структуре документа.

В предыдущих версиях языка HTML только заголовки (**h1**, **h2**, и т. д.) инициировали создание новых элементов структуры. Новые элементы **nav** (основная навигация) и **aside** (для информации во врезках) также являются элементами разделов.

В спецификации HTML5 у элемента раздела может быть собственная внутренняя иерархия, начинающаяся с **h1**, независимо от его положения в документе, что позволяет поместить элемент **article** с его внутренней структурой внутрь другого элемента в потоке и знать, что структура документа не нарушится. Новый алгоритм построения структуры документа призван сделать так, чтобы разметка отвечала потребностям первичного и последующего использования контента в современной Всемирной паутине.

На момент написания книги браузеры еще не поддерживали систему построения структуры документа спецификации HTML5, поэтому, чтобы сделать документ доступным и логически структурированным для всех пользователей уже сейчас, безопаснее использовать заголовки в убывающем порядке даже в элементах разделов.

Для получения дополнительной информации, я рекомендую прочитать статью Люка Стивенса на сайте **webknowledge.ru/pravda-o-strukture-documenta-html5-standarta/**.

Косвенное содержимое (врезки)

<aside>...</aside>

Косвенно связанные с контентом материалы

Новый в HTML5

Элемент **aside** определяет контент, косвенно связанный с окружающим содержимым. Это эквивалент боковой врезки в печати, но элемент нельзя было назвать **sidebar** потому, что слово «боковой» (side) указывает на положение, а не назначение. Тем не менее боковая врезка — хороший наглядный пример того, как используется **aside**. Его можно применить для размещения цитат, справочной информации, элементов списка, выносок или другого материала, связанного с контентом (но не имеющего особого значения).

В данном примере элемент **aside** используется для создания списка ссылок, связанных с основной статьей.

```
<h1>Веб-типографика</h1>

<p>В 1997 году существовали конкурирующие между собой форматы шрифтов и инструменты для их создания...</p>

<p>Теперь у нас есть несколько способов использования прекрасных шрифтов на веб-страницах...</p>

<aside>

<h2>Ресурсы веб-шрифтов</h2>

<ul>

<li><a href="http://typekit.com/">Сервис Typekit</a></li>

<li><a href="http://www.google.com/webfonts">Шрифты Google</a></li>

</ul>

</aside>
```

Элемент **aside** не отображается по умолчанию, поэтому вам необходимо превратить его в блочный элемент и настроить внешний вид и положение с помощью таблиц стилей.

Навигация

<nav>...</nav>

Ссылки основной навигации

Новый в HTML5

Новый элемент **nav** предоставляет разработчикам семантический способ указать навигацию для сайта. Ранее в этой главе нам встречался неупорядоченный список, который можно использовать как средство навигации в верхней части страницы для сайта каталога шрифтов. Заключение этого списка в элемент **nav** четко укажет его назначение.

```
<nav>
<ul>
<li><a href="">Serif</a>/li>
<li><a href="">Sans-serif</a></li>
<li><a href="">Script</a></li>
<li><a href="">Display</a></li>
<li><a href="">Dingbats</a>/li>
</ul>
</nav>
```

Однако не все списки ссылок следует окружать тегами **nav**. В спецификации четко указано, что его нужно применять для ссылок, обеспечивающих основную навигацию по всему сайту, длинному разделу или статье.

Элемент **nav** может быть особенно полезен с точки зрения доступности. Как только программы экранного доступа и другие устройства станут совместимы со спецификацией HTML5, пользователи смогут легко обнаружить или пропустить раздел навигации без длительных поисков.

Верхние и нижние колонтитулы

<header>...</header>

Верхний колонтитул страницы, раздела или статьи

Новый в HTML5

Так как верстальщики веб-страниц на протяжении многих лет размечали в документах разделы верхнего и нижнего колонтитула, не вызывало сомнений, что полноценные элементы для этих действий окажутся полезны. Начнем с верхнего колонтитула.

Верхние колонтитулы

<footer>...</footer>

Нижний колонтитул страницы, раздела или статьи

Новый в HTML5

Элемент **header** используется для размещения вводного материала, который обычно расположен в начале веб-страницы либо в верхней части раздела или статьи. Не существует определенных требований к содержанию элемента **header**; допустимо добавлять все, что подходит в качестве введения к странице или разделу.

ПРЕДУПРЕЖДЕНИЕ

Элементы **header** и **footer** не могут содержать вложенные элементы верхнего или нижнего колонтитула.

В следующем примере верхний колонтитул элемента содержит логотип, название сайта и средства навигации.

```
<header>
<img src="/images/logo.png">
<hgroup>
<h1>Коротко о веб-шрифтах</h1>
<h2>Последние новости из мира веб-типографики</h2>
</hgroup>
<nav>
```

```
<ul>
<li><a href="">Главная</a></li>
<li><a href="">Блог</a></li>
<li><a href="">Магазин</a></li>
</ul>
/nav>
</header>
… Контент страницы …
```

При использовании в отдельной публикации элемент **header** может включать в себя название статьи, имя автора и дату публикации, как показано здесь:

```
<article>
<header>
<h1>Дополнительные сведения о WOFF</h1>
<p>Дженнифер Роббинс, <time datetime="11-11-2011"
pubdate>11 ноября, 2011</time></p>
</header>
<p>...здесь начинается текст статьи…</p>
</article>
```

Нижние колонтитулы

Элемент **footer** используется для указания типа информации, которая обычно сообщается в конце страницы или статьи, например имя автора, информация об авторских правах, относящиеся к статье документы или элементы навигации. **footer** может применяться ко всему документу или быть связанным с определенным разделом или статьей. Если нижний колонтитул находится прямо внутри элемента **body** до или после остального контента этого элемента, то он применяется ко всей странице. Если он помещен внутрь элемента раздела (**section**, **article**, **nav** или **aside**), элемент **footer** анализируется как нижний колонтитул только данного раздела. Отметим, что, хотя его и называют «нижним колонтитулом», нет никаких требований, предписывающих, что он должен быть указан на последнем месте в документе или элементе раздела.

ПРИМЕЧАНИЕ

Вы также можете добавить верхние и нижние колонтитулы к корневым элементам разделов: **body**, **blockquote**, **details**, **figure**, **td** и **fieldset**.

В этом простом примере мы видим типичную информацию, указанную в нижней части статьи или сообщения в блоге, размеченную как элемент **footer**.

```
<article>
<header>
<h1>Дополнительные сведения о WOFF</h1>
<p>Дженнифер Роббинс, <time datetime="11-11-2011"
```

```
pubdate>11 ноября, 2011</time></p>
</header>
<p>...здесь начинается текст публикации...</p>
<footer>
<p><small>Все права защищены &copy;2013 Дженнифер Роббинс.</
small></p>
<nav>
<ul>
<li><a href="">Предыдущая</a></li>
<li><a href="">Следующая</a></li>
</ul>
</nav>
</footer>
</article>
```

Адреса

<address>...</address>

Контактная информация

Последний и редко используемый элемент — это **address**, который применяется для создания области, предоставляющей контактную информацию об авторе или о тех, кто поддерживает данный документ. Эти данные, как правило, располагаются в начале или в конце документа, а также в разделе или статье документа. Подходящее расположение адреса — в элементе **footer**.

ПРИМЕЧАНИЕ

У вас появится возможность потренироваться в работе с элементами разделов в упражнении 5.3 в конце этой главы.

Важно отметить, что элемент **address** нельзя использовать для всех типов адресов, например почтовых, поэтому сфера его применения довольно ограничена. Он предназначен специально для указания контактных данных автора (хотя потенциально эти данные могут содержать почтовый адрес). Ниже представлен пример использования этого элемента по назначению. **href** — это часть разметки для указания ссылки; мы доберемся до нее в главе 6.

```
<address>
Редактор <a href="../editors/rightman/index.html">Михаил
Райтман</a>, <a href="http://www.ozon.ru">Ozon.ru</a>
</address>
```

Встроенные элементы

Теперь, выделив крупные фрагменты контента, мы можем придать смысловое значение фразам в этих фрагментах с помощью элементов, называемых в спецификации HTML5 *семантическими текстовыми элементами*. В разговоре их чаще всего называют *встроенными элемен-*

ами, так как по умолчанию они отображаются в потоке текста и не вызывают перевода строк. Так их называли в версиях языка HTML, предшествующих HTML5.

Текстовые (встроенные) элементы

Несмотря на все типы информации, которые вы могли бы добавить в документ, в языке HTML5 доступны только элементы, перечисленные в табл. 5.1.

Табл. 5.1. Семантические встроенные текстовые элементы

Элемент	Описание
a	Якорь или гиперссылка (подробнее см. в главе 6)
abbr	Аббревиатура
b	Зрительно привлекает внимание, например, к ключевым словам (начертание полужирным шрифтом)
bdi	Новый в HTML5. Указывает на текст, имеющий особые требования к направлению размещения.
bdo	Переопределяет двунаправленный алгоритм; четко указывает направление текста (слева направо **ltr** или справа налево **rtl**)
br	Перевод строки
cite	Цитата; ссылка на заголовок работы, например, на название книги
code	Пример кода программы
data	Только WHATWG. Машиночитаемые эквиваленты дат, времени, веса и других измеряемых значений.
del	Удаленный текст; указывает редактирование, выполненное для документа
dfn	Термин определения или первое появление термина
em	Выделенный текст
i	Альтернативное отображение текста (курсивом)
ins	Вставленный текст; указывает вставку в документе
kbd	Клавиатура; текст, введенный пользователем (для технических документов)
mark	Новый в HTML5. Контекстуально значимый текст
q	Короткое, встроенное цитирование
ruby, rt, rp	Новый в HTML5. Предоставляет аннотации или фонетическую транскрипцию для восточноазиатских шрифтов и идеограмм
s	Неверный текст (зачеркивание)
samp	Образец вывода программы (листинг)

Предыстория встроенных элементов

Многие из встроенных элементов, существовавших с момента появления Всемирной паутины, были введены для изменения визуального форматирования выделенных областей текста, так как таблиц стилей не существовало. Если был нужен полужирный текст, его размечали элементом **b**. Курсив? Применялся элемент **i**. В самом деле, когда-то применялся **font**, используемый исключительно для изменения гарнитуры, цвета и размера шрифта. Неудивительно, что этот элемент, ответственный сугубо за представление, исключен из спецификации HTML5. Тем не менее многие встроенные элементы «старой школы» (например, **u** для подчеркивания и **s** для зачеркивания) были сохранены в HTML5 и получили новые семантические определения (**b** сейчас используется для «ключевых слов», а **s** — для «неверного текста»).

Некоторые встроенные элементы — сугубо семантические (например, **abbr** или **time**) и не имеют визуализаций по умолчанию. Необходимо использовать CSS, если требуется изменить их внешний вид при отображении.

В описаниях элементов в данном разделе я указываю определения встроенных элементов, а также ожидаемое отображение их браузером по умолчанию, если таковое существует.

Элемент	Описание
small	Мелкий шрифт, например, в сообщениях о защите авторских прав или уведомлении (отображается шрифтом меньшего размера)
span	Общий фразовый контент
strong	Выделенный текст, которому придают особое значение
sub	Нижний индекс
sup	Верхний индекс
time	Новый в HTML5. Машиночитаемое время
u	Подчеркнутый
var	Переменная величина или аргумент программы (для технических документов)
wbr	Перенос слова

Устаревшие текстовые элементы спецификации HTML 4.01

Из спецификации HTML5 исключено множество элементов, которые были помечены как *устаревшие* (не рекомендуемые к использованию). Я привела их здесь на случай, если они встретятся вам в старой разметке. Однако нет никаких причин для их использования — для большинства есть аналогичные свойства таблиц стилей или же они недостаточно хорошо поддерживаются.

Элемент	Описание
acronym	Обозначает аббревиатура (например, МЧС); верстальщикам следует использовать вместо него элемент abbr
applet	Вставляет апплет на языке Java
basefont	Устанавливает заданные по умолчанию параметры настройки шрифта для документа
big	Несколько увеличивает размер шрифта размеченного текста по сравнению с остальным
center	Выравнивает контент по центру
dir	Список каталогов (заменен маркированными списками)
font	Начертание, цвет и размер шрифта
isindex	Вставляет поле поиска
menu	Список меню (заменен маркированными списками, теперь используется для обеспечения команд контекстных меню)
strike	Перечеркнутый текст
tt	Телетайп, отображается моноширинным шрифтом

Акцентированный текст

`<em>...</em>`

Выделенный встроенный текст

Для указания части предложения, которую необходимо акцентировать или выделить, используйте элемент **em**. Расположение элемента **em** влияет на понимание смысла предложения. Рассмотрим две следующие фразы; они идентичны за исключением выделенных слов.

```
<p><em>Миша</em> очень умен.</p>
<p>Миша <em>очень</em> умен.</p>
```

В первом предложении указывается, *кто* очень умен. Во втором говорится, *насколько* он умен.

Элементы выделенного текста (**em**) почти всегда отображаются по умолчанию курсивом (рис. 5.9), но, конечно, вы можете оформить их любым способом, который вам нравится, с помощью таблицы стилей. В программах экранного доступа может использоваться другая интонация, чтобы передать выделенный текст, поэтому вы должны использовать элемент **em** только когда есть смысловая необходимость, а не просто чтобы выделить текст курсивом.

Важный текст

`<strong>...</strong>`

Текст, имеющий важное значение

Элемент **strong** указывает, что слово или фраза — важны. В следующем примере с помощью элемента **strong** обозначены фрагменты инструкции, требующие особого внимания.

```
<p>Возвращая машину, <strong>оставьте ключи в красном ящике
у стойки регистрации</strong>.</p>
```

По умолчанию графические браузеры отображают текст, заключенный в элементе **strong**, полужирным шрифтом. Программы экранного доступа могут использовать другую интонацию для акцентирования важного контента, поэтому помечайте текст элементом **strong** только тогда, когда это влияет на смысл, а не просто чтобы выделить текст полужирным шрифтом.

Ниже показаны небольшие примеры текста с элементами **em** и **strong** (рис. 5.9).

Миша очень умен.

Миша *очень* умен.

Возвращая машину, **оставьте ключи в красном ящике у стойки регистрации**.

Рис. 5.9. *Вид по умолчанию акцентированного и важного текста*

Элементы с новыми смысловыми значениями

`<b>...</b>`

Ключевые слова или визуально выделенный текст (полужирный)

`<i>...</i>`

Альтернативная речь (курсив)

`<s>...</s>`

Неверный текст (зачеркивание)

`<u>...</u>`

Аннотированный текст (подчеркивание)

`<small>...</small>`

Юридический текст; мелкий шрифт (меньший размер шрифта)

Говоря о выделении текста полужирным начертанием и курсивом, давайте рассмотрим, где сейчас используются старые элементы **b** и **i**. Элементы **b**, **i**, **u**, **s** и **small** давно появились во Всемирной паутине как способ предоставления инструкций по набору текста (полужирный, курсив, подчеркивание, зачеркивание и текст меньшего размера соответственно). Несмотря на их изначальное предназначение, они были включены в спецификацию HTML5 и получили обновленные семантические определения на основе шаблонов их использования в прошлом. Браузеры по-прежнему отображают их по умолчанию так, как вы ожидаете (рис. 5.10). Однако если вы хотите только изменить стиль шрифта, более подходящим решением будет применить таблицу стилей. Оставьте эти элементы для тех случаев, когда они семантически уместны. Рассмотрим элементы и их правильное использование, а также предлагаемые таблицами стилей альтернативы.

b

Определение HTML 4.01: Полужирный

Определение HTML5: Ключевые слова, названия продуктов и другие фразы, которые необходимо выделить из окружающего текста без придания дополнительного значения или акцента.

Альтернатива CSS: Для получения полужирного начертания используйте свойство **font-weight**. Например, **font-weight: bold**.

Пример:
```
<p>Черточки на концах букв называются <b>засечками</b>.</p>
```

ПРИМЕЧАНИЕ

Чтобы выбрать наилучший вариант, я представляю, как текст будет считан программой экранного доступа. Если я не хочу, чтобы слово было прочитано громким голосом с выразительной интонацией, но его нужно выделить полужирным, тогда элемент **b** предпочтительнее, чем **strong**.

i

Определение HTML 4.01: Курсив

Определение HTML5: Указывает, что текст отличается по стилю речи или настроению от окружающего его контента, например, фразы на другом языке, технический термин или мысль.

Свойство CSS: Для выделения текста курсивом, используйте свойство **font-style**. Например: **font-style: italic**.

Пример:
```
<p>Просто смените шрифт и <i>Вуаля!</i>, текст воспринимается по-другому!</p>
```

s

Определение HTML 4.01: Зачеркнутый текст

Определение HTML5: Обозначает неверный текст.

Свойство CSS: Чтобы зачеркнуть выбранный текст, примените свойство **text-decoration**.

Например: **text-decoration: line-through**.

Пример: `<p>Шрифт Scala Sans был создан <s>Эриком Гиллом</s> Мартином Майуром.</p>`

u

Определение HTML 4.01: Подчеркивание

Определение HTML5: В некоторых случаях подчеркивание имеет смысловое значение, например подчеркивание формального имени в китайском языке или указание неправильно написанного слова при проверке орфографии. Обратите внимание, что подчеркнутый текст легко спутать со ссылкой, и поэтому в целом следует избегать его использования, за исключением нескольких специальных случаев.

Свойство CSS: Для создания подчеркнутого текста примените свойство **text-decoration**.

Например: **text-decoration: underline**.

Пример: `<p>Таблички в нью-йоркском метро набраны шрифтом <u>Helvetica</u>.</p>`

small

Определение HTML 4.01: Отображает шрифт меньшего размера, чем окружающий текст

Определение HTML5: Обозначает дополнение или примечание к основному тексту, такое как мелкий шрифт юридических уведомлений в нижней части документа.

Свойство CSS: Чтобы уменьшить размер шрифта текста, примените свойство **font-size**. *Пример*:

```
<p><small>Данный шрифт можно использовать в коммерческих целях.</small></p>
```

Черточки на концах букв называются **засечками**.

Просто смените шрифт и *Вуаля!*, текст воспринимается по-другому!

Шрифт Scala Sans был создан ~~Эриком Гиллом~~ Мартином Майуром.

Таблички в нью-йоркском метро набраны шрифтом Helvetica.

Данный шрифт можно использовать в коммерческих целях.

Рис. 5.10. Вид по умолчанию элементов **b**, **i**, **u**, **s** *и* **small**

Короткие цитаты

`<q>...</q>`

Короткие встроенные цитаты

Используйте элемент цитаты (**q**), чтобы разметить короткие цитаты типа «Быть или не быть» в потоке текста, как показано в этом примере.

```
Мэтью Картер сказал: <q>Наш алфавит не изменялся веками.</q>
```

Согласно спецификации HTML, браузеры должны автоматически добавлять кавычки вокруг элементов **q**, поэтому вам не нужно включать их в исходный код. Многие браузеры именно так и делают, за исключением Internet Explorer версии 7 и более ранних. К счастью, на момент написания книги только 5–8% используемых браузеров не поддерживали элемент **q**, и их станет еще меньше к тому времени, как вы будете ее читать. Если переживаете, что небольшой процент пользователей увидит цитаты без кавычек, добавьте кавычки в исходный код — это неплохая альтернатива.

Мэтью Картер сказал: "Наш алфавит не изменялся веками."

Рис. 5.11. Практически все браузеры автоматически добавляют кавычки вокруг элементов **q**

ПАМЯТКА

Вложение элементов

К одной строке текста можно применить два элемента (например, к фразе на иностранном языке, являющейся одновременно цитатой), но убедитесь, что они вложены правильно. Это означает, что внутренний элемент, включая его закрывающий тег, должен полностью находиться внутри внешнего элемента и не пересекаться с ним.

```
<q><i>Je ne sais pas.</i></q>
```

Аббревиатуры и сокращения

`<abbr>...</abbr>`

Аббревиатуры, акронимы и сокращения

Разметка сокращений, акронимов и аббревиатур с помощью элемента **abbr** предоставляет полезную информацию для поисковых систем, программ экранного доступа и других устройств. Сокращения представляют собой укороченные варианты слов, оканчивающиеся точкой (например, «обл.» для слова «область»). Аббревиатуры — сокращенная форма написания словосочетания, например ФСБ (по первым буквам, Федеральная Служба Безопасности). Акронимы — это аббревиатуры, составленные из начальных звуков слов исходного словосочетания (например, ГЭС — гидроэлектростанция). В отличие от аббревиатур, образованных начальными буквами, акронимы образованы начальными звуками. Полная версия сокращенного слова указывается в атрибуте **title**, как показано в этом примере.

ПРИМЕЧАНИЕ

В языке HTML 4.01 применялся элемент **acronym**, используемый специально для акронимов, однако в спецификации HTML5 его сочли устаревшим, оставив в обоих случаях элемент **abbr**.

```
<abbr title="Ракетные войска стратегического назначения">РВСН</abbr>

<abbr title="Сантиметры">см.</abbr>
```

Цитаты

`<cite>...</cite>`

Цитата

Элемент **cite** используется для оформления цитаты из другого документа, такого как книга, журнал, статья и пр. Цитаты по умолчанию представлены курсивным текстом. Ниже продемонстрирован пример:

```
<p>На написание статьи меня вдохновило <cite>Полное руководство по книгопечатанию</cite> Джеймса Феличи.</p>
```

Определение терминов

`<dfn>...</dfn>`

Определение термина

В документе первое и поясняемое употребление слова или термина часто выделяется тем или иным способом. В этой книге значимые термины выделены курсивным шрифтом голубого цвета.

В HTML-документе вы можете обозначить их с помощью элемента **dfn** и отформатировать визуально, используя таблицы стилей.

```
<p><dfn>Молоко</dfn> питательная жидкость, вырабатываемая
молочными железами самок млекопитающих.</p>
```

Элементы программного кода

`<code>...</code>`

Код

`<var>...</var>`

Переменная величина

`<samp>...</samp>`

Пример программы

`<kbd>...</kbd>`

Вводимые пользователем нажатия клавиш

Многие встроенные элементы используются для того, чтобы описать части технических документов, такие как код (**code**), переменные величины (**var**), образцы программ (**samp**) и вводимые пользователем нажатия клавиш (**kbd**). Для меня это причудливое напоминание о происхождении языка HTML из научного мира (Тим Бернерс-Ли разработал язык HTML для совместного использования документов в ЦЕРНе в 1989 году).

Элементы кода, примеров и нажатий клавиш по умолчанию представлены моноширинным шрифтом, таким как Courier. Переменные величины обычно отображают курсивом.

Нижний и верхний индекс

`<sub>...</sub>`

Нижний индекс

`<sup>...</sup>`

Верхний индекс

Элементы нижнего (**sub**) и верхнего индексов (**sup**) форматируют выделенный текст шрифтом меньшего размера и располагают его чуть ниже (**sub**) или выше (**sup**) основной строки. Эти элементы могут быть полезны при обозначении химических формул или математических уравнений.

На рис. 5.12 показано, как верхний и нижний индексы обычно отображаются в браузере.

```
<p>H<sub>2</sub>0</p>
<p>E=MC<sup>2</sup></p>
```

H_20

$E=MC^2$

Рис. 5.12. *Нижний и верхний индексы*

Выделенный текст

<mark>...</mark>

Текст, значимый в контексте

Новый в HTML5

Новый элемент **mark** выделяет информацию, которая особенно важна для читателя. Можно использовать его для поиска слова на странице результатов, для привлечения внимания к фрагменту текста или для выделения текущей страницы из нескольких. Некоторые дизайнеры (и браузеры) помещают текст, размеченный элементом **mark**, на неярком цветном фоне, словно он выделен цветным маркером, как показано на рис. 5.13.

```
<p> ... Конституция Российской Федерации. Глава 7.
<mark>Судебная власть</mark>. Статья 118, п. 2. <mark>Судебная
власть</mark> осуществляется посредством конституционного,
гражданского, административного и уголовного судопроизвод-
ства.</p>
```

... Конституция Российской Федерации. Глава 7. Судебная власть. Статья 118, п. 2. Судебная власть осуществляется посредством конституционного, гражданского, административного и уголовного судопроизводства.

Рис. 5.13. *Искомые термины, размеченные с помощью элемента* **mark** *и выделенные желтым фоном с помощью таблицы стилей, чтобы читателю легче было их найти*

ОСОБЕННОСТИ ПОДДЕРЖКИ

Элемент **mark** не поддерживается в браузере Internet Explorer версии 8 и более ранних (чтобы найти альтернативное решение, прочитайте врезку «Поддержка элементов языка HTML5 в браузере Internet Explorer»).

Обозначения времени и машиночитаемые данные

<time>...</time>

Обозначение времени

Новый в HTML5

Когда мы видим фразу «полдень, 4 ноября», то знаем, что это дата и время. Однако для компьютерной программы содержимое может быть не так понятно. Элемент **time** не только позволяет размечать даты и время наиболее удобным для прочтения образом, но и кодирует их в стандартизированной форме, чтобы данные могли использовать компьютеры. Содержимое элемента представляет информацию людям, а атрибут **datetime** представляет ту же информацию в машиночитаемом формате.

Элемент **time** обозначает дату, время или их сочетание. Его можно использовать для передачи информации о дате и времени приложению, например при сохранении записи о мероприятии в персональном календаре. Его могут использовать поисковые системы, чтобы найти недавно опубликованные статьи. Или его можно применить для изменения стиля отображения времени на альтернативный формат (например, сменить с 18:00 на 6 вечера).

ПРИМЕЧАНИЕ

Элемент **time** не предназначен для разметки обозначений без точного времени и даты, таких как «конец прошлого года», «рубеж веков».

Атрибут **datetime** указывает информацию о дате и времени в стандартном формате, показанном на рис. 5.14. Обозначение начинается с даты (год, месяц, день), а затем вводится буква T, чтобы обозначить время, указываемое в часах, минутах, секундах (не обязательно) и миллисекундах (не обязательно). Наконец, часовой пояс указывает, на сколько часов время отстает (-) или обгоняет (+) Всемирное координированное

время (UTC). Например, «+03:00» — Московское время, на три часа обгоняющее Всемирное координированное время.

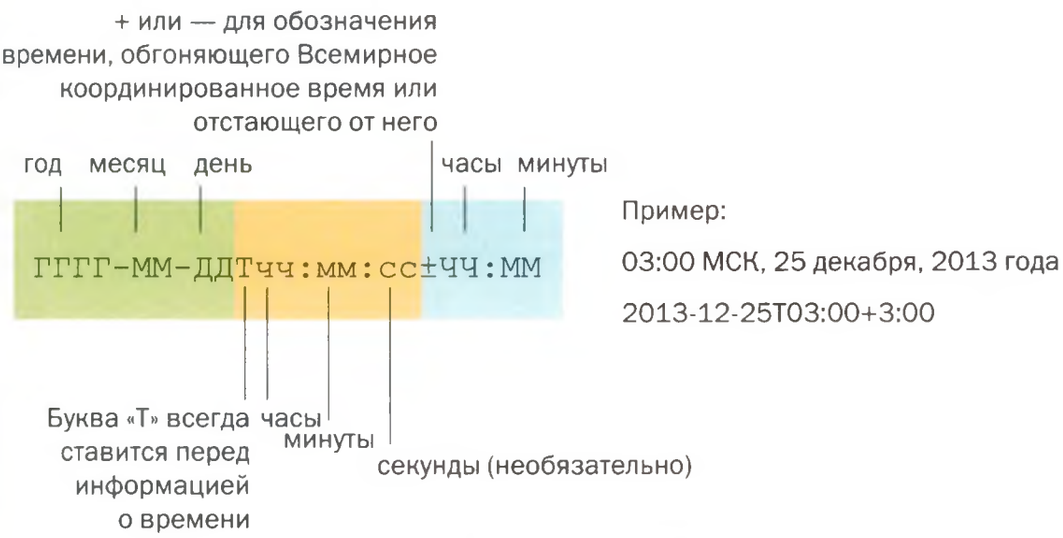


Рис. 5.14. Стандартизированный синтаксис обозначения даты и времени

<data>...</data>

Машиночитаемые данные

Только WHATWG

Спецификация WHATWG HTML содержит атрибут **pubdate**, используемый для указания времени публикации документа, как показано в этом примере. Атрибут **pubdate** не был включен в спецификацию HTML5 консорциума Всемирной паутины на момент написания книги, но, возможно, его добавят позднее, если он приобретет широкое распространение.

```
Автор Иванов Сергей(<time datetime="2012-09-01T 20:00-05:00"
pubdate>1 сентября, 2012 года, 20:00 МСК</time>)
```

Спецификация WHATWG также содержит элемент **data**, чтобы помочь компьютерам распознать контент, в котором могут быть использованы всевозможные типы данных, включая дату, время, единицы измерения, вес и т. д. Он использует атрибут **value** для указания машиночитаемой информации. Ниже представлено несколько примеров:

```
<data value="12">Двенадцать</data>
<data value="2011-11-12">Прошлая суббота</data>
```

Я не буду подробно рассматривать элемент **data**, так как на момент написания книги все еще велось обсуждение, как именно он должен работать, и консорциум Всемирной паутины не принял его в спецификации HTML5. Кроме того, как новичок, вы в любом случае вряд ли сразу же будете иметь дело с машиночитаемыми данными. Тем не менее, интересно понять, как можно использовать разметку для предоставления полезной информации и компьютерным программам и сценариям, и людям.

ОСОБЕННОСТИ ПОДДЕРЖКИ

Элементы **time** и **data** — новые и на момент написания книги еще не получили повсеместной поддержки. Однако вы можете стилизовать их, и они будут распознаваться браузерами, за исключением Internet Explorer 8 и более ранних версий.

Вставленный и удаленный текст

<ins>...</ins>

Вставленный текст

~~...~~

Удаленный текст

Элементы **ins** и **del** используются для пометки исправлений текста и указания частей документа, которые, соответственно, были вставлены или удалены. При представлении эти элементы опираются на правила стилей (если нет соответствующих настроек браузера по умолчанию). Элементы **ins** и **del** могут содержать строковые или блочные элементы в зависимости от того, какой тип контента в них заключен.

```
Генеральный директор: <del title="уволен">Питер Пэн</del>
<ins>Пэппи Длинныйчулок</ins>
```

Перевод строк

**
**

Перевод строки

Иногда требуется добавить перевод строки в поток текста. Так как мы знаем, что браузер игнорирует строки в исходном документе, нам нужны конкретные инструкции, чтобы сообщить браузеру: начать с новой строки здесь.

Встроенный элемент перевода строки (**br**) создан именно для этого. Классический пример использования **br** — перевод строк адреса или стихотворения. Это пустой элемент, то есть он не имеет контента. Просто добавьте элемент **
** (**
** в XHTML) в ту позицию текста, где вы хотите сделать перевод строки, как показано на рис. 5.15.

```
<p>
Но когда ей окраситься время настанет <br>На радость птицам
в цвет золотой, <br>Тогда она поневоле проглянет, <br>Упадет –
и раздавлена будет пятой! <br><br><br>Симандзаки Тосон "При-
дорожная слива"
</p>
```

Но когда ей окраситься время настанет
На радость птицам в цвет золотой,
Тогда она поневоле проглянет,
Упадет – и раздавлена будет пятой!

Симандзаки Тосон "Придорожная слива"

Рис. 5.15. Каждый перевод строки создан элементом **br**

К сожалению, элементом **br** легко злоупотребить (см. предупреждение далее). Подумайте, возможно, использование свойства **white-space** таблицы стилей (описывается в главе 12) будет лучшей альтернативой, чтобы сохранить переводы строк исходного кода без дополнительной разметки.

ПРЕДУПРЕЖДЕНИЕ

Старайтесь не использовать элементы **br**, чтобы добавить перевод строк в текст, который на самом деле должен быть списком. Например, не делайте следующее:

```
<p>молоко<br>
хлеб<br>
апельсиновый сок<br>
</p>
```

Если это список, используйте семантически правильный элемент маркированного списка и исключите маркеры с помощью таблиц стилей.

```
<ul>
<li>молоко</li>
<li>хлеб</li>
<li>апельсиновый сок</li>
</ul>
```

Адаптирование для неевропейских языков

Поскольку паутина — всемирная, существует несколько элементов, предназначенных для удовлетворения потребностей неевропейских языков.

Изменение направления

Элемент **bdo** (переопределение двунаправленного алгоритма) позволяет включать фразу, написанную на языках, где текст читается справа налево (**rtl**) (например, иврите или арабском), в поток текста, идущего слева направо (**lrt**) или наоборот.

Ниже представлен пример, как написать «Шалом»:

```
<bdo dir="rtl">&#x05E9;&#x05DC;&#x05D5;&
#x05DD;</bdo>
```

Элемент **bdi** (двунаправленная изоляция) действует подобным образом, но он используется, чтобы изолировать выделенный фрагмент, который, *возможно*, будет читаться в другом направлении, например имя или комментарий пользователя.

Советы для восточноазиатских языков

В спецификации HTML5 также присутствуют элементы **ruby**, **rt** и **rp**, используемые для добавления фуриган к восточноевропейским языкам. Фуриганы — это маленькие значки, которые обычно ставятся над иероглифами и содержат подсказки по их произношению и переводу.

В элементе **ruby** полезный текст фуриганы обозначается элементом **rt**. Браузеры, поддерживающие текст фуриганы, обычно отображают его мелким шрифтом над основным текстом. В качестве запасного варианта для браузеров, которые не поддерживают фуриганы, вы может указать этот текст в скобках, пометив каждые элементом **rp**. Браузеры, не поддерживающие элемент **ruby**, отобразят весь текст на одной строке с фуриганой в скобках. А те, что поддерживают, проигнорируют содержание элементов **rp** и отобразят над иероглифами только текст, указанный в элементе **rt**. На момент написания книги система элементов для отображения текста фуриган поддерживалась браузерами лишь частично.

```
<ruby>
字<rp>(</rp><rt>hàn</rt><rp>)</rp>
汉<rp>(</rp><rt>zì</rt><rp>)</rp>
</ruby>
```

Перенос слов

Элемент переноса (**<wbr>**) позволяет отметить позицию, в которой слово должно быть при необходимости разделено («возможность переноса строки», как указано в спецификации). Благодаря этому, браузерам дается четкая команда, а верстальщики могут указать, в какой позиции лучше всего разделить слово для переноса на следующую строку. Имейте в виду, что слово разбивается с помощью элемента **<wbr>**, только если это необходимо (рис. 5.16). Когда пространства достаточно, слово остается целым. Браузеры давно поддерживают этот элемент, но в стандарт HTML он был включен недавно.

<wbr>

Перенос слов

```
<p>Самое длинное слово, которое вам приходилось слышать, пи-
шется так:<em>никотинамид<wbr>адениндинуклеотид<wbr>фосфатг
идрин</em>!</p>
```

Самое длинное слово, которое вам приходилось слышать пишется так:*никотинамид адениндинуклеотидфосфатгидрин*!

Рис. 5.16. Когда слову не хватает пространства на одной строке, его разбивают на части с помощью элемента **wbr**

УПРАЖНЕНИЕ 5.2. ОПРЕДЕЛЕНИЕ ВСТРОЕННЫХ ЭЛЕМЕНТОВ

Это небольшое сообщение в блоге бистро «Черный гусь» даст вам возможность определить и разметить различные встроенные элементы. Узнайте, сможете ли вы найти фразы, которые необходимо разметить с помощью следующих элементов:

b br cite dfn em i q small time

Так как данный процесс всегда немного субъективен, разметка, получившаяся у вас в результате, может отличаться от показанной в примере в приложении А, но это шанс применить все элементы, перечисленные выше. В качестве дополнительного задания в одной из фраз нужно применить два элемента (не забывайте правильно вкладывать их, закрывая внутренний элемент прежде, чем закрыть наружный).

Вы можете вносить свои изменения прямо на этой странице или найти файл на диске, прилагаемом к книге, и редактировать его в текстовом редакторе. Результирующая разметка предоставлена в приложении А.

```
<article>
<header>
<p>Разместил Игорь, 15 ноября, 2012</p>
</header>
<h1>Вакуумные деликатесы</h1>
<p>На этой неделе мне особенно хочется поделиться с вами
новым способом приготовления пищи в вакууме. Готовя
пищу в вакууме, вы помещаете продукт (обычно в вакуумной
пластиковой упаковке) в емкость с водой той температу-
ры, в которой вы собираетесь этот продукт готовить. Джеф
Поттер в книге Кухня для фанатов описывает эту технику,
как варка при очень низкой температуре.</p>
<p>В следующем месяце в нашем бистро будет подаваться ло-
сось под чесночно-сливочным соусом, приготовленный в ва-
кууме. Зарезервировать столик можно до 30 ноября. </p>
<p>blackgoose@example.com
555-336-1800</p>
<p> Внимание: Лосось в вакууме не подвергается паste-
ризации. Не рекомендуется беременным женщинам и людям
с ослабленным иммунитетом.</p>
</article>
```

Общие элементы (div и span)

** ... **

Общий блочный элемент

...

Общий встроенный элемент

Что если ни один из элементов, о которых мы говорили, не описывает ваш контент достаточно точно? В мире бесконечно много типов информации, а семантических элементов гораздо меньше. К счастью, язык HTML обеспечивает два общих элемента, которые могут быть настроены для отличного описания вашего контента. Элемент **div** обозначает разделение контента, в то время как элемент **span** используется для обозначения слова или фразы, для которых пока не существует тек-

стового элемента. Вы присваиваете общему элементу имя, используя атрибут **id** или **class** (мы поговорим о них чуть позже).

Элементы **div** и **span** не имеют собственных изначальных качеств представления, но вы можете использовать таблицы стилей, чтобы отформатировать контент, как вам понравится. Фактически, общие элементы — первичный инструмент в основанном на стандартах веб-дизайне, потому что они позволяют верстальщикам точно описать контент и предлагают множество «трюков», с помощью которых можно добавлять правила стилей. Они также позволяют получать доступ к элементам на странице и применять к ним сценарии JavaScript.

Я собираюсь потратить немного времени на элементы **div** и **span** (а также на атрибуты **id** и **class**) и выяснить, как верстальщики используют их, чтобы структурировать контент.

Разделение контента с помощью элемента div

Элемент **div** используется для определения логического разделения контента или элементов на странице. Он указывает, что вместе элементы образуют некую смысловую единицу и должны рассматриваться CSS и JavaScript как одна единица. Отмечая связанные элементы в качестве раздела **div** и предоставляя ему уникальный идентификатор **id** или указывая, что это часть элемента **class**, вы предоставляете контекст для элементов на странице. Давайте рассмотрим несколько примеров элементов **div**.

СОВЕТ ПО РАЗМЕТКЕ

Существует возможность вкладывать элементы **div** внутрь других элементов **div**, но не перегибайте палку. Вы должны всегда стремиться, чтобы разметка была как можно проще, поэтому добавляйте элемент **div**, только если он необходим для логической структуры, добавления стиля или сценария.

В этом примере элемент **div** используется как контейнер, чтобы сгруппировать изображение и два абзаца.

```
<div class="listing">
<img src="mcdonald.gif" alt="">
<p><cite>Создание Web-сайтов. Основное руководство</cite>,
Мэтью МакДональд</p>
<p>Из книги вы узнаете, как создавать веб-страницы и многое
другое.</p>
</div>
```

Помещая эти элементы в **div**, я уточнила, что они концептуально связаны. Это также позволяет мне применять стили к элементам **p** в пределах листингов иначе, чем к остальным абзацам на странице.

Ниже представлен пример другого привычного использования элемента **div**, употребляемого, чтобы разбить страницу на части в целях компоновки макета. В этом примере заголовок и несколько абзацев заключены в элемент **div** и определены как раздел «novosti».

```
<div id="novosti">
<h1>Новости этой недели</h1>
<p>Мы продолжали работать над...</p>
<p>И последнее, но не в последнюю очередь...</p>
</div>
```

Теперь, когда у меня есть элемент, известный как «новости», я могла бы использовать таблицу стилей, чтобы поместить его в столбец с правого или левого края страницы. Возможно, вы спросите: «А разве нельзя применить для этого элемент **section**?» Можно! На самом деле, сейчас, когда у нас появились лучшие семантические элементы группировки в спецификации HTML5, верстальщики реже прибегают к использованию общих элементов **div**.

Встраивание с помощью элемента span

Элемент **span** предлагает те же преимущества, что и **div.** Также он используется для встроенных элементов, которые не вводят переводов строк. Поскольку элементы **span** встроенные, они могут содержать только текст и другие встроенные элементы (иными словами, вы не можете поместить туда заголовки, списки, элементы группировки контента и т. д.). Для правильного понимания обратимся к некоторым примерам.

Не существует элемента **telephone,** но мы можем использовать элемент **span** для придания значения телефонным номерам. В этом примере каждый номер телефона размечен как **span** и классифицирован как «tel».

```
<ul>
<li>Владимир: <span class="tel">999-8282</span></li>
<li>Сергей: <span class="tel">888-4889</span></li>
<li>Леонид: <span class="tel">888-1628</span></li>
<li>Любовь: <span class="tel">999-3220</span></li>
</ul>
```

Видно, как элементы **span** добавляют значение к контенту, который иначе мог бы быть случайной последовательностью цифр. Помимо этого элемент **span** позволяет применять один и тот же стиль ко всем телефонным номерам на сайте (например, гарантировать, что они никогда не будут разделены на две строки с помощью определения **white-space: nowrap** каскадных таблиц стилей). Так информация становится распознаваемой не только для людей, но и для компьютерных программ, которые «знают», что сделать с «телефонной» информацией. Фактически, некоторые значения (в том числе «tel») были приняты в стандарт системы разметки, известной как «Микроформаты», которая повышает полезность контента для программ (см. врезку «Микроформаты и метаданные»).

Идентификаторы и классы

В предыдущих примерах мы видели атрибуты **id** (*идентификаторы*) и **class** (*классы*), используемые для обеспечения контекста общих элементов **div** и **span**. У этих атрибутов, однако, не одинаковое предназначение, и важно знать различие.

Идентификаторы

Идентификаторы используются, чтобы создавать *уникальные* идентификаторы в документе. Другими словами, значение идентификатора должно использоваться только однажды в документе. Это делает его полезным для назначения имени специфическому элементу, как если бы это была часть данных. Смотрите врезку «Значения идентификаторов и классов» для получения информации относительно допустимых значений идентификатора.

Этот пример использует номер ISBN, чтобы уникально идентифицировать каждую книгу. Два издания не могут иметь общий идентификатор.

```
<div id="ISBN9785699401246">
<img src="mcdonald.gif" alt="">
<p><cite>Создание Web-сайтов. Основное руководство</cite>,
Мэтью МакДональд</p>
<p> Из книги вы узнаете, как создавать веб-страницы и многое
другое.</p>
</div>
<div id="ISBN9785699429974">
<img src="croll_power.gif" alt=" /">
<p><cite>Комплексный веб-мониторинг</cite>, Алистер Кролл,
Шон Пауэр</p>
<p>В книге рассматриваются все аспекты вашего присутствия
в Интернете и способы их измерения.</p>
</div>
```

Веб-дизайнеры также используют идентификаторы, определяя различные секции страницы. В приведенном примере документ не может содержать более одного идентификатора для каждого из значений — «header», «main», «links» или «news».

```
<section id="main">
<!-- здесь располагаются элементы основного контента-->
</section>
<section id="news">
<!-- здесь располагается боковая панель тем новостей-->
</section>
<aside id="links">
<!-- здесь располагается список ссылок-->
</aside>
```

Значения идентификаторов и классов

Значения идентификаторов и классов должны начинаться с латинской буквы (A-Z или a-z) или символа подчеркивания. Они не должны содержать никаких пробелов или специальных символов. Кириллические буквы недопустимы. Буквы, цифры, дефисы, подчеркивания, двоеточия и точки применимы. Кроме того, значения чувствительны к регистру, таким образом значение «sectionB» отлично от «Sectionb».

Не только для элементов div

Идентификаторы и классы могут использоваться почти всеми элементами языка HTML5, а не только **div** и **span**. Например, вы могли идентифицировать маркированный список как «navigation», а не заворачивать его в элемент **div**.

```
<ol id="navigation">
<li>... </li>
<li>... </li>
<li>... </li>
</ol>
```

Обратите внимание, что в спецификации HTML 4.01 идентификаторы и классы могут использоваться со всеми элементами кроме **base**, **basefont**, **head**, **html**, **meta**, **param**, **script**, **style** и **title**.

Микроформаты и метаданные

Как вы видели, элементы HTML не способны описать каждый тип информации. Группа разработчиков решила, что, если удастся стандартизировать имена классов (например, всегда использовать атрибут «tel» для обозначения телефонных номеров), можно будет создать системы для описания данных, чтобы сделать классы более полезными. Эта система называется *Микроформаты*. Она расширяет семантику разметки HTML-документов путем установки стандартных *значений* для атрибутов **id**, **class** и **rel** вместо создания новых элементов.

Существует несколько «словарей» микроформатов, используемых для определения таких данных, как контактная информация (**hCard**) или элементы календаря (**hCalendar**). Узнать о них можно на сайте **microformats.org**. Чтобы дать вам общее представление, в следующем примере описываются фрагменты события с помощью словаря «hCalendar» системы «Микроформаты», чтобы браузер мог автоматически добавить их в ваш электронный календарь.

```
<section class="vevent">

<span class="summary">"Веб-дизайн 2012".
Техническая конференция </span>,

<time class="dtstart" datetime="20110306">
6 марта</time>

<time class="dtend"
datetime="20110310">10, 2011</time>

<div class="location">МФТИ, Москва, Рос-
сия</div>

<a class="url" href="http://events.
example.com pub/e/403">Ссылка</a>

</section>
```

Словарь «hCard» определяет компоненты типичной контактной информации (хранятся в формате vCard), в том числе: адрес (**adr**), почтовый индекс (**postal-code**), область (**region**), номер телефона (**tel**) и другие. Браузер может захватить информацию с веб-страницы и автоматически добавить ее в адресную книгу.

О микроформатах можно сказать намного больше, чем я могу описать в этой книге. И не только это — в разработке консорциума Всемирной паутины находятся две дополнительные, более сложные системы для добавления метаданных на веб-страницы: *RDFa* и *Микроданные*. Непонятно, как они будут сочетаться в долгосрочной перспективе, и я думаю, что вам в любом случае не хочется разбираться в метаданных прямо сейчас. Но когда вы будете готовы узнать больше, на сайте **websitesmaderight.com** собрано множество вступительных статей и учебных пособий по всем трем системам.

СОВЕТ

Атрибут **id** (идентификатор) используется для *идентификации*. Атрибут **class** (класс) используется для *классификации*.

Классы

Классы используются для классификации элементов по общим группам; поэтому в отличие от атрибута **id**, множество элементов может разделять общее имя класса. Делая элементы частью одного класса, вы можете применить стили ко всем маркированным элементам одновременно с помощью единственного правила стилей или управлять всеми ими сразу с помощью сценария. Давайте начнем с классификации некоторых элементов в приведенном выше примере. Здесь я добавила атрибуты **class** к каждому элементу **div**, чтобы классифицировать их как «zapis», и к определенным абзацам, чтобы классифицировать их как «opisanie».

```
<div id="ISBN9785699401246" class="zapis">

<img src="mcdonald.gif" alt="">

<p><cite>Создание Web-сайтов. Основное руководство</cite>,
Мэтью МакДональд</p>

<p class="opisanie">Из книги вы узнаете, как создавать веб-
страницы и многое другое.</p>

</div>
```

```
<div id="ISBN9785699429974" class="zapis">
<img src="croll_power.gif" alt=" /">
<p><cite>Комплексный веб-мониторинг</cite>, Алистер Кролл,
Шон Пауэр</p>
<p class="opisanie">В книге рассматриваются все аспекты ва-
шего присутствия в Интернете и способы их измерения.</p>
</div>
```

Обратите внимание, что один и тот же элемент может иметь одновременно класс и идентификатор. Также элементы могут принадлежать многим классам. При перечислении нескольких значений атрибута **class** разделяйте их пробелами. В этом примере я классифицировала каждый элемент **div** как «kniga», чтобы настроить их отдельно от записей для «cd» или «dvd» в других позициях документа.

```
<div id="ISBN9785699401246" class="zapis kniga">
<img src="mcdonald.gif" alt="">
<p><cite>Создание Web-сайтов. Основное руководство</cite>,
Мэтью МакДональд</p>
<p class="opisanie">Из книги вы узнаете, как создавать веб-
страницы и многое другое.</p>
</div>
<div id="ISBN9785699429974" class="zapis kniga">
<img src="croll_power.gif" alt="">
<p><cite>Комплексный веб-мониторинг</cite>, Алистер Кролл,
Шон Пауэр</p>
<p class="opisanie">В книге рассматриваются все аспекты ва-
шего присутствия в Интернете и способы их измерения.</p>
</div>
```

Теперь вы должны понять, как используются элементы **div** и **span**, чтобы обеспечить для документов смысл и организацию. Мы познакомимся с ними поближе в главах, посвященных таблицам стилей в части III.

Некоторые специальные символы

Осталась еще только одна связанная с текстом тема, прежде чем мы перейдем к следующей главе.

Некоторые общие символы, такие как символ авторского права ©, не являются частью стандартного набора ASCII, который содержит только буквы, цифры и несколько основных символов. Другие обозначения тоже доступны, например знак меньше (<), но если вы поместите его отдельно в HTML-документ, то браузер будет интерпретировать его как начало тега.

В исходном документе нужно *избегать* подобных символов. Избегать – значит не вводить непосредственно символ, а указывать его числовую или именованную *символьную ссылку*. Когда браузер «видит» такую ссылку, он подставляет надлежащий значок в этой позиции при выводе страницы.

Существует два способа обратиться к определенному символу: назначенным числовым значением (*числовая сущность*) или используя предопределенное сокращенное имя для символа (называемое *именованной сущностью*). Все символьные ссылки начинаются с амперсанда (`&`) и заканчиваются точкой с запятой (`;`).

Это станет яснее, когда мы рассмотрим примеры. Так, я хотела бы добавить символ авторского права к моей странице, поэтому должна использовать именованную сущность **`&copy;`** (или ее числовой эквивалент **`&#169;`**) в той позиции, где я хочу отобразить символ (рис. 5.17).

```
<p>Все права защищены &copy; 2013, Дженнифер Роббинс</p>
```

или:

```
<p>Все права защищены &#169; 2013, Дженнифер Роббинс</p>
```

Язык HTML определяет сотни именованных символьных сущностей как часть языка разметки, то есть вы не можете создать собственную. В табл. 5.2 перечислены некоторые обычно используемые символьные ссылки. Полный список символьных ссылок представлен на сайте **www.webstandards.org/learn/reference/charts/entities/**.

ПРИМЕЧАНИЕ

В языке XHTML *нужно* избегать амперсанда, чтобы он не интерпретировался как начало символьной сущности даже когда появляется в значении атрибута. Например,

```
<img src="jj.
jpg" alt="Компания
Джонсон&amp;Джонсон"/>
```

Все права защищены © 2013, Дженнифер Роббинс

Рис. 5.17. *Специальный символ подставляется вместо символьной ссылки, когда документ отображается в браузере*

Табл. 5.2. *Распространенные специальные символы и их символьные ссылки*

Символ	Описание	Имя	Число
	Символ пробела (неразрывный)	` `	` `
&	Амперсанд	`&`	`&#038;`
'	Апостроф	`&apos;`	`&#039;`
<	Символ меньше (используется для показа разметки на веб-странице)	`<`	`&#060;`
>	Символ больше (используется для показа разметки на веб-странице)	`>`	`&#062;`
©	Авторское право	`&copy;`	`&#169;`
®	Зарегистрированная торговая марка	`&reg;`	`&#174;`
™	Торговая марка	`&trade;`	`&#8482;`
£	Фунт	`&pound;`	`&#163;`

Неразрывные пробелы

Одним из интересных символов, о котором стоит узнать, является неразрывный пробел (**` `**). Его цель в том, чтобы между двумя словами не было перевода строки. Поэтому, например, если я размечаю свое имя таким образом:

```
Дженнифер Роббинс
```

то могу быть уверена, что имя и фамилия всегда будут оставаться вместе на одной строке.

Символ	Описание	Имя	Число
¥	Иена	&yen;	&#165;
€	Евро	&euro;	&#8364;
-	Короткое тире	&ndash;	&#8211;
–	Длинное тире	&mdash;	&#8212;
‘	Левая изогнутая одиночная кавычка	&lsquo;	&#8216;
’	Правая изогнутая одиночная кавычка	&rsquo;	&#8217;
“	Левая изогнутая двойная кавычка	&ldquo;	&#8220;
”	Правая изогнутая двойная кавычка	&rdquo;	&#8221;
•	Маркер	&bull;	&#8226;
…	Горизонтальные эллиптические маркеры	&hellip;	&#8230;

Резюме

К настоящему времени вы узнали, как размечать элементы, и познакомились со всеми HTML-элементами, предназначенными для добавления структуры и значения к текстовому содержимому. Теперь дело только за практикой. Упражнение 5.3 позволяет применить все, что мы изучили к настоящему времени: элементы структуры документа, блочные элементы, встроенные элементы, элементы разделов и символьные сущности. Развлекайтесь!

СОВЕТ

Помните, что последовательное выделение каждого иерархического уровня в исходном HTML-коде с помощью отступов упрощает просмотр и обновление документа в дальнейшем.

УПРАЖНЕНИЕ 5.3. БЛОГ НА САЙТЕ БИСТРО «ЧЕРНЫЙ ГУСЬ»

Теперь, когда вы ознакомились со всеми текстовыми элементами, можете испытать их в работе, размечая меню для блога бистро «Черный Гусь». Ниже показан сырой текст (второе сообщение уже размечено с помощью встроенных элементов из упражнения 5-2).

Откройте текстовый файл, расположенный на диске, прилагающемся к книге. Код, который должен получиться в результате, показан в приложении А и доступен там же.

```
Блог Бистро "Черный Гусь"
Главная
Меню
Блог
Контакты
Летнее меню
Разместил Игорь, 15 июня 2013
Наш шеф-повар потратил немало времени, составляя идеальное меню для летнего сезона. Приходите к нам теплыми летними вечерами и наслаждайтесь закусками и основными блюдами.
Закуски
Колбаски
Эльзасские колбаски из свиного окорока с черным перцем, луком и кориандром. 435 руб.
Расколбас -- Новинка!
Ассорти из колбасок с картофельным пюре, тушеной капусткой, картофелем фри, горчицей и кетчупом. 1450 руб.
Основные блюда
Лосось на углях в устричном соусе
Также подается тонкий лаваш, салат из маринованной капустки, зелень, классический соус. 700 руб.
```

```
Шашлычки с тигровой креветкой -- Новинка!

Тигровые креветки, помидорки Черри, соус
терияки. 720 руб.

Вакуумные деликатесы

Разместил Игорь, 15 ноября, 2012

<p>На этой неделе мне <em>особенно</em>
хочется поделиться с вами новым способом
приготовления пищи <dfn><i>в вакууме</
i></dfn>. Готовя пищу <i>в вакууме</i>,
вы помещаете продукт (обычно в вакуумной
пластиковой упаковке) в емкость с водой
той температуры, в которой вы собираетесь
этот продукт готовить. Джеф Поттер в книге
<cite>Кухня для фанатов</cite> описывает
эту технику, как <q>варка при очень низкой
температуре</q>.</p>

<p>В следующем месяце в нашем бистро бу-
дет подаваться <b>лосось под чесночно-
сливочным соусом, приготовленный в вакуу-
ме</b>. Зарезервировать столик можно до
30 ноября. </p>

<p>blackgoose@example.com

555-336-1800</p>

<p> Внимание: Лосось в вакууме не под-
вергается пастеризации. Не рекомендуется
беременным женщинам и людям с ослабленным
иммунитетом.</p>

Наш адрес: Рязань, ул. Электровольтная,
17, Часы работы: С понедельника по четверг
с 11 до 21, в пятницу и субботу: с 11 до
полуночи

Все права защищены &copy; 2013, бистро
"Черный гусь"
```

1. Сначала введите структурные элементы документа (**html**, **head**, **meta**, **title** и **body**). Задайте ему название «Бистро «Черный Гусь»: Блог».
2. Прежде всего, выделите заголовок верхнего уровня и список ссылок, заключив их в **header** (не забудьте про закрывающий тег). Заголовок внутри этого элемента должен быть создан с помощью тега **h1**, а ссылки организованы в виде неупорядоченного списка (**ul**). Не волнуйтесь о том, как превратить элементы списка в ссылки; мы поговорим об их создании в следующей главе. Придайте списку дополнительное значение, определив его в качестве основной навигации сайта (**nav**).

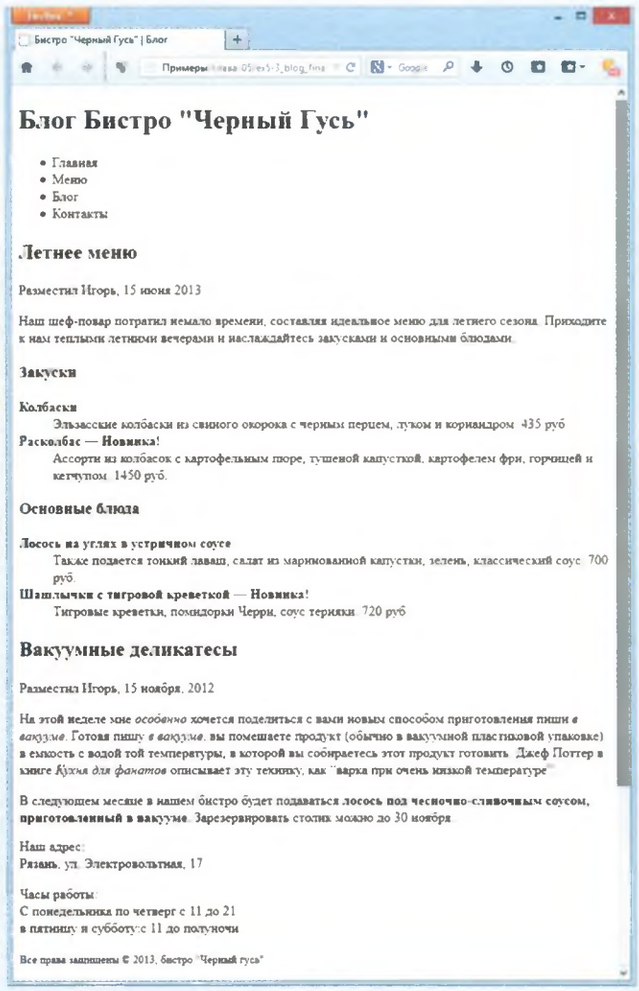


Рис. 5.18. Законченная страница меню

3. На странице блога размещены два поста (публикации), называющиеся «Летнее меню» и «Вакуумные деликатесы». Разметьте каждый из них с помощью элемента **article**.
4. Теперь приведем в порядок первую статью! Создадим для нее элемент **header**, содержащий заголовок (в этот раз **h2**, так как опустились вниз по иерархии документа) и сведения о публикации (**p**). Обозначьте дату публикации статьи с помощью элемента **time**, как вы делали в упражнении 5.2.
5. Контент после заголовка — явно обычный абзац. Однако у меню есть интересные особенности. Оно поделено на два значимых раздела («Закуски» и «Основные блюда»), поэтому разметьте их как элементы **section**. Внимательно проследите, чтобы закрывающий тег раздела (**</section>**) располагался после закрывающего тега статьи (**</article>**), так элементы будут вложены верно и не пересекутся. Наконец, определите разделы с помощью атрибута **id**. Назовите первый идентификатор — «appetizers», а второй — «entrees».

6. Разделы на месте, теперь можно разметить контент. Для заголовков каждого раздела используем элемент **h3**. Выберите наиболее подходящие элементы списка для перечисления названий блюд и их описаний. Разметьте списки и их пункты.
7. Теперь можно добавить несколько последних штрихов. *Классифицируйте* все цены как «price», применив элемент **span**.
8. Два блюда — новинки. Замените двойные дефисы символами длинных тире и придайте особое значение новым блюдам. Классифицируйте название каждого нового блюда как «newitem» (подсказка: используйте имеющийся элемент **dt**; в этот раз нет необходимости применять элемент **span**). Это позволит нам выделить названия блюд в меню с помощью класса «newitem» и применить к ним иные стили, чем к остальным блюдам в меню.
9. Так мы закончили первую статью. Вторая в основном уже была размечена в прошлом упражнении, но вам нужно разметить элемент **header**, содержащий подходящий заголовок и сведения о публикации.
10. Теперь разметьте оставшийся контент применимым ко всей странице целиком элементом **footer**. Разметьте каждую строку контента в элементе **footer** как отдельный абзац.
11. Далее создадим определенный контекст для сведений о местонахождении бистро и его распорядке работы, поместив их в блок **div** с именем «about». Сделайте так, чтобы метки «Location» и «Hours» появлялись на отдельных строках, добавив после них перенос строки. Если хотите, можете также разметить часы работы с помощью элемента **time**.
12. Наконец, сведения об авторских правах обычно отображаются мелким шрифтом, поэтому разметьте их соответственно. Последним штрихом добавьте символ авторских прав после фразы «Все права защищены».

Сохраните файл с именем *bistro_blog.html* и проверьте свою страницу в современном браузере (помните, что Internet Explorer 8 и его более ранние версии не поддерживают новые элементы разделов спецификации HTML5). Просмотрите результат.

Подсказки по разметке:

- Выберите элемент, который лучше всего соответствует значению выбранного текста.
- Не забывайте закрывать элементы закрывающими тегами.
- Помещайте все значения атрибутов в кавычки.
- Последовательность команд «скопировать и вставить» — ваш помощник при добавлении идентичной разметки к множеству элементов. Только убедитесь, что вы скопировали корректную информацию, прежде чем вы вставите ее повсюду в документе.

Ниже приведена итоговая таблица элементов, которые мы охватили в этой главе. Новые элементы HTML5 обозначены символом (5). На момент написания книги элемент **data** присутствовал только в версии HTML сообщества WHATWG.

Разделы страницы

address	контактная информация автора
article (5)	автономный контент
aside (5)	косвенный контент (боковая врезка)
footer (5)	связанный с сайтом контент
header (5)	вводный контент
nav (5)	основная навигация
section (5)	тематически объединенная группа элементов контента

Заголовки

h1... h6	заголовки
hgroup	группа заголовков

Элементы группировки контента

blockquote	длинная цитата

Нужно больше практики?

Попытайтесь разметить собственное резюме. Начните с «сырого» текста, затем добавьте элементы структуры документа, блочные элементы, затем встроенные элементы, как мы делали в упражнении 5.3. Если вы сразу не видите элемент, который соответствует вашему контенту, попробуйте создать его, используя элементы **div** или **span**.

`div`	общий раздел
`figure (5)`	связанное изображение или ресурс
`figcapture (5)`	текстовое описание (подпись) рисунка
`hr`	тематическое разделение на уровне абзаца (горизонтальная линия)
`p`	абзац
`pre`	предварительно отформатированный текст

Элементы списка

`dd`	определение
`dl`	список определения
`dt`	термин
`li`	пункт списка (для элементов `ul` и `ol`)
`ol`	нумерованный список
`ul`	маркированный список

Переносы

`br`	перевод строки
`wbr (5)`	перенос слова

Семантические встроенные элементы

`abbr`	аббревиатура, акроним или сокращение
`b`	визуально привлекает внимание (полужирный)
`bdi (5)`	возможное изменение направления
`bdo`	переопределение двунаправленного алгоритма
`cite`	цитата
`code`	образец кода
`data (WHATWG)`	машиночитаемый эквивалент
`del`	удаленный текст
`dfn`	термин определения
`em`	выделенный текст
`i`	альтернативная речь (курсив)
`ins`	вставленный текст
`kbd`	текст с клавиатуры
`q`	короткая цитата
`ruby (5)`	раздел, содержащий текст фуриганы
`rp (5)`	скобки в тексте фуриганы
`rt (5)`	текст фуриганы
`s`	перечеркивание
`samp`	пример результата
`small`	маленький текст
`span`	общая фраза или текст
`strong`	придание особого значения
`sub`	нижний индекс
`sup`	верхний индекс
`time (5)`	машиночитаемое указание времени
`u`	привлечение внимания (подчеркнутый)
`var`	переменная величина

ГЛАВА 6

ДОБАВЛЕНИЕ ССЫЛОК

Создавая страницу для Всемирной паутины, вы, вероятно, станете вставлять ссылки на другие веб-страницы, будь то разделы собственного сайта или сторонние ресурсы. На ссылках построена вся Всемирная паутина. В этой главе я рассмотрю принципы разметки, благодаря которым работают ссылки на сторонние сайты, другие разделы собственного сайта, а также ссылки в пределах одной страницы.

Есть элемент, который делает возможной работу ссылок — это *якорь* (`a`).

```
<a>...</a>
```

Элемент якоря (привязки) (гипертекстовая ссылка)

Чтобы превратить выделенный текст в ссылку, поместите его между открывающим и закрывающим тегами — `<a>...</a>` — и с помощью атрибута **`href`** укажите URL-адрес страницы, на которую будет осуществляться переход. Содержимое элемента якоря становится гипертекстовой ссылкой. В данном примере сформирована ссылка на веб-сайт «Эксмо»:

```
<a href="http://www.eksmo.ru/">Перейти на сайт Эксмо</a>
```

Чтобы сделать ссылкой изображение, укажите в элементе якоря элемент **`img`**:

```
<a href="http://www.eksmo.ru/"><img src="logo.png"
alt="Логотип Эксмо"></a>
```

В большинстве обозревателей текст ссылки отображается синим цветом с подчеркиванием. В некоторых старых версиях браузеров изображения-ссылки выделяются синей рамкой, но в большинстве современных этого нет. Посещенные ссылки, как правило, становятся фиолетовыми. У пользователей есть возможность изменить эти цвета в настройках браузера, а вы, конечно же, можете изменять внешний вид ссылок своего сайта с помощью таблиц стилей. Как это сделать, вы узнаете из главы 13.

ПРЕДУПРЕЖДЕНИЕ

Короткое замечание: если вы решите изменить цвета ссылок, рекомендуем соблюдать единство их оформления на всем сайте, чтобы пользователи могли легко ориентироваться на его страницах.

В этой главе

- Создание ссылок на внешние страницы
- Создание относительных ссылок на документы на собственном сервере
- Ссылки на определенную часть страницы
- Добавление ссылок на адрес электронной почты и номер телефона
- Открытие ссылок на новой вкладке (в новом окне) браузера

НА ЗАМЕТКУ

Синтаксис якоря

Ниже приведена упрощенная структура элемента якоря:

```
<a href="url-адрес">текст ссылки или изображение</a>
```

В спецификации HTML5 внутрь элемента `a` *можно поместить любые другие элементы, даже блочные.*

В момент, когда пользователь щелкает мышью по текстовой или графической ссылке, в окне браузера загружается страница, указанная вами в элементе якоря. Приведенный выше пример разметки будет выглядеть примерно так, как показано на рис. 6.1.

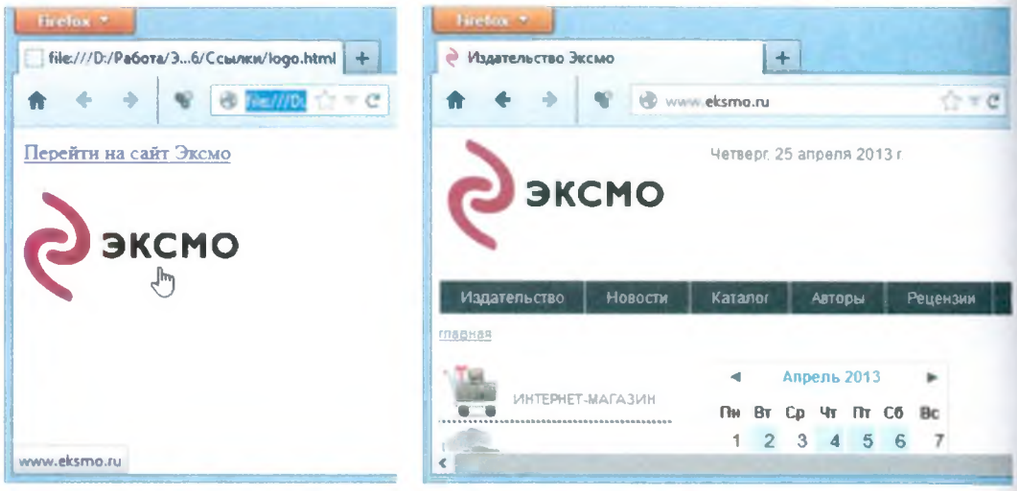


Рис. 6.1. *Когда пользователь щелкнет мышью по тексту или изображению-ссылке, в окне браузера загрузится страница, адрес которой вы указали в элементе якоря*

URL против URI

Консорциум Всемирной паутины вместе с сообществом разработчиков отказываются от термина *унифицированный указатель ресурса* (*URL*, Uniform Resource Locator) в пользу более общего и технически более точного понятия *унифицированный идентификатор ресурса* (*URI*, Uniform Resource Identifier). Однако в разговорах и на работе вы, скорее всего, пока еще будете слышать термин URL.

Вот в чем кроется тайная причина противостояния URL и URI. URL — это один из типов URI, который определяет ресурс по его местонахождению (L в аббревиатуре означает «location» — местонахождение) в сети. Другой тип URI — URN определяет ресурс по имени или пространству имен (N в аббревиатуре означает «name» — имя).

Поскольку термин URL более известен, в данной главе я буду придерживаться именно его. Просто знайте, что URL-адреса — это подмножество URI-адресов, и данные термины часто взаимозаменяемы.

Если вы хотите углубиться в изучение такого рода вещей, я могу посоветовать вам посетить сайт **citforum.ru/internet/xml/uri/**.

Атрибут `href`

Браузеру необходимо сообщить, на какой документ осуществлять переход по ссылке. Атрибут **`href`** (гипертекстовая ссылка) содержит в себе адрес страницы (URL-адрес) и передает его браузеру. URL-адрес обязательно должен быть заключен в кавычки. В большинстве случаев вам придется ссылаться на другие HTML-документы; однако возможно создание ссылок и на другие объекты, например изображения, аудио- и видео-файлы. Поскольку в том, чтобы расставить теги якоря вокруг той или иной части контента веб-страницы, нет ничего сложного, истинное мастерство заключается в правильном указании URL-адреса.

Существует два варианта:

- ***Абсолютная ссылка*** содержит полный URL-адрес, включая протокол (**`http://`**), доменное имя и соответствующий путь к файлу. При указании пути к документу, находящемуся на стороннем веб-ресурсе, ссылка должна быть абсолютной.

Пример: `href="`**`http://`**`www.eksmo.ru/"`

Случается, что страница, на которую вы ссылаетесь, имеет длинный URL-адрес, и тогда ссылка может выглядеть довольно несуразно (рис. 6.2). Просто помните, что структура ссылки представляет собой простой элемент-контейнер с одним атрибутом. Пусть длинный путь вас не пугает.

- ***Относительная ссылка*** описывает путь к указанному документу относительно текущего. Относительные URL-пути указываются при создании ссылок на другие документы на том же сайте (то есть на одном и том же сервере). В ней не нужно указывать протокол или домен, а только путь к файлу.

Пример: `href="recipes/index.html"`

В этой главе мы будем создавать ссылки на кулинарный сайт «Кухня Кристины» с использованием абсолютных и относительных URL-адресов (см. рис. 6.3). С абсолютными URL-адресами работать проще, поэтому, в первую очередь, разберемся с ними.

Рис. 6.2. *Пример длинного URL-адреса. Несмотря на несколько странный вид тега якоря, его структура остается неизменной*

СОВЕТ ПО РАЗМЕТКЕ

Укрощение URL-путей

Если необходимо создать ссылку на страницу с длинным URL-адресом, это удобнее сделать, скопировав данные в адресной строке браузера и вставив их в создаваемый документ. Таким образом вы избежите ситуации, когда пропуск одной буквы в адресе приводит к тому, что ссылка не работает.

Ссылки на веб-страницы других сайтов

Часто вам понадобится создавать ссылки на страницы, размещенные на других серверах во Всемирной паутине. Они называются *внешними ссылками*, потому что ведут на страницы, размещенные вне вашего сервера или сайта. Чтобы создать внешнюю ссылку, необходимо указать абсолютный URL-адрес, начинающийся с символов **`http://`** (с протокола).

Сейчас мы вставим несколько внешних ссылок на главную страницу сайта «Кухня Кристины» (рис. 6.3). Сначала создадим ссылку для элемента списка «Рецепты онлайн», ведущую на сайт **www.rezepty.ru**. Текст ссылки размечаем открывающим и закрывающим тегами якоря. Обратите внимание, что мы вставляем теги якоря внутри элемента списка (**`li`**), потому что блочные элементы, каким является **`li`**, не могут входить в состав встроенного элемента якоря. Разместить элемент **`a`** внутри элемента **`ul`** будет неверно с точки зрения спецификации HTML.

ПОТРЕНИРУЙТЕСЬ

Изучайте материал, работая над сайтом «Кухня Кристины»

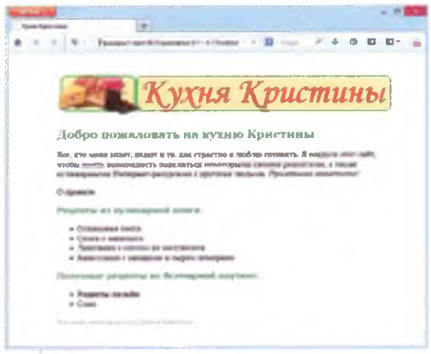


Рис. 6.3. Вид страницы сайта «Кухня Кристины»

Все файлы, необходимые для создания веб-сайта «Кухня Кристины» вы найдете на прилагающемся диске в папке *Примеры\глава-06\Упражнения 6-1 — 6-7\kriskitchen*. Скопируйте на жесткий диск вашего компьютера эту папку целиком, не нарушая порядка организации файлов.

Итоговая разметка для всех упражнений приведена в Приложении А.

Страницы не слишком красочны, но они помогают приобрести навыки работы со ссылками.

ПРИМЕЧАНИЕ

На компьютерах под управлением операционных систем Windows и OS X файлы организованы в «папки», однако среди сетевых разработчиков принято употреблять равнозначный и более технический термин *каталог*. Папка — это всего лишь каталог с красивым значком.

УПРАЖНЕНИЕ 6.1. СОЗДАНИЕ ВНЕШНЕЙ ССЫЛКИ

Откройте файл *index.html* в папке *Примеры\глава-06\Упражнения 6-1 — 6-7\kriskitchen*. Следуя приведенному ниже примеру, создайте для элемента списка «Смак» ссылку на веб-страницу телепередачи «Смак», **www.1tv.ru/sprojects/si=23**:

```
<ul>
<li><a href="http://www.rezepty.ru">Рецепты онлайн</
a></li>
<li>Смак</li>
</ul>
```

Когда все будет готово, можно сохранить файл *index.html* и открыть его в браузере. Щелкните мышью по созданной вами ссылке и перейдите на сайт телепередачи «Смак». Если ссылка не работает, вернитесь к предыдущим шагам и проверьте правильность разметки.

```
<li><a>Рецепты онлайн</a></li>
```

Следующим шагом добавим атрибут **href** с полным путем к сайту.

```
<li><a href="http://www.rezepty.ru">Рецепты онлайн</a></li>
```

На этом все. Теперь слова «Рецепты онлайн» будут отображаться в виде ссылки, щелкнув по которой, мы перейдем на указанный сайт.

Ссылки на страницы собственного сайта

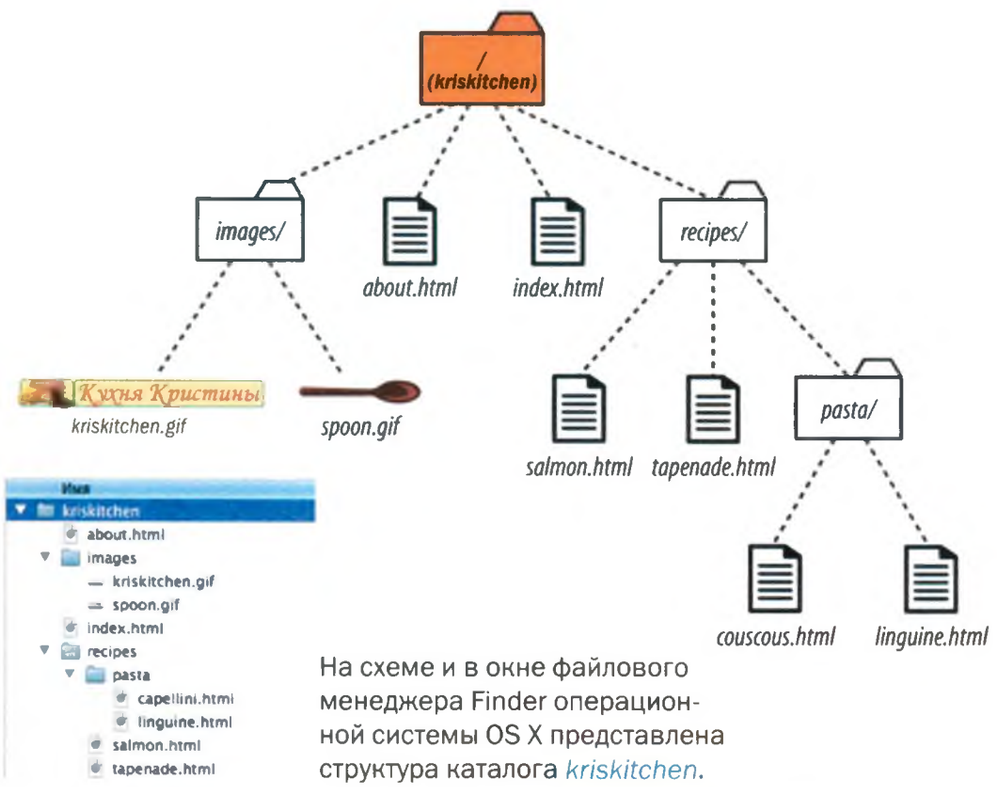
Большинство ссылок, которые вам предстоит создать, будут вести на страницы вашего же сайта: с главной — на страницы разделов, с них — на страницы с контентом и так далее. В этом случае вам понадобятся относительные ссылки — те, что обращаются к какой-либо странице на том же сервере.

Если браузер не находит в ссылке символов «http://», он выполняет поиск указанного документа на том же сервере. *Путь к файлу* — обозначение, призванное указать на определенный файл — сообщает браузеру, где его искать. Формат веб-путей к файлам основывается на конвенции Unix, согласно которой каталоги и имена файлов отделяются друг от друга слешем (/). *Относительный путь к файлу* описывает, как добраться по ссылке до нужного файла, начиная с позиции текущего документа.

Освоение работы с относительными путями может оказаться непростой задачей. Исходя из моего опыта преподавания, ничто так не сбивает с толку новичков, как написание относительного пути к файлу, поэтому разберемся с этим по порядку. В тексте вам будут встречаться

упражнения, которые я рекомендую выполнять по мере изучения материалов книги.

Все примеры путей к файлам, приводимые в данном разделе, основаны на структуре сайта «Кухня Кристины», приведенной на рис. 6.4. Если каталоги сайта представить в виде схемы, то в конечном счете она будет выглядеть как перевернутое дерево с корневым каталогом в вершине иерархии. В случае с сайтом «Кухня Кристины» корневой каталог называется *kriskitchen*. Другой способ увидеть такую структуру предоставляет файловый менеджер Finder (в операционной системе OS X), где отображаются каталоги и подкаталоги (пользователи операционной системы Windows видят их по одному).



На схеме и в окне файлового менеджера Finder операционной системы OS X представлена структура каталога *kriskitchen*.

Рис. 6.4. *Схема, представляющая структуру сайта «Кухня Кристины»*

Чего не стоит делать при создании путей файлов

При создании относительных путей к файлам чрезвычайно важно следовать следующим правилам во избежание наиболее распространенных ошибок:

- **Не используйте обратный слеш (\).** В сетевых URL-путях используется только прямой слеш (/).
- **Не начинайте путь с буквы диска (D:, C: и т. д.).** Ссылки между страницами будут работать, пока сайт находится на локальном компьютере, но когда вы загрузите его на удаленный сервер, названия дисков помешают работе ссылок.
- **Не начинайте ссылки с file://.** Данное обозначение также свидетельствует о том, что файл является локальным, поэтому ссылка не будет работать на веб-сервере.

Создание ссылок в пределах одного каталога

Наиболее простым относительным URL-адресом является ссылка на другой файл в пределах одного каталога. При создании ссылки на файл в том же каталоге нужно указать только *имя файла* (с расширением). Когда URL-адрес представляет собой лишь имя файла, сервер ищет его в текущем каталоге (то есть в том, где находится HTML-документ со ссылкой).

Ссылка, содержащая в себе только имя файла, свидетельствует о том, что он находится в одном каталоге с текущим документом.

В этом примере сделаем ссылку с главной страницы (*index.html*) на страницу с информацией о сайте (*about.html*). Оба файла находятся в одном каталоге (*kriskitchen*). Таким образом, нам достаточно просто указать имя файла (рис. 6.5):

```
<a href="about.html">О проекте</a>
```

Из диаграммы видно, что файлы *index.html* и *about.html* находятся в одном и том же каталоге.

Из файла *index.html*:

```
<a href="about.html">О проекте</a>
```

Сервер производит поиск файла в том же каталоге, что и документ, с которого осуществляется переход.

Рис. 6.5. *Указание относительного URL-адреса к другому документу в том же каталоге*

УПРАЖНЕНИЕ 6.2. СОЗДАНИЕ ССЫЛКИ НА ДОКУМЕНТ В ПРЕДЕЛАХ ОДНОГО КАТАЛОГА

Откройте файл *about.html* в папке *Примеры\глава-06\Упражнения 6-1 — 6-7\kriskitchen*. В нижней части страницы добавьте абзац «Вернуться на главную страницу», ведущий на страницу *index.html*. Не забудьте, что элемент якоря должен быть включен в элемент `p`, а не наоборот.

```
<p>Вернуться на главную страницу</p>
```

Когда все будет готово, сохраните страницу *about.html* и откройте ее в браузере. Для локальной проверки работоспособности ссылок (то есть проверки работы на вашем компьютере) подключение к Интернету не требуется. При щелчке мышью по ссылке должен происходить возврат на главную страницу.

Создание ссылки на файл во вложенном каталоге

А как быть, если файлы находятся в разных каталогах? Нужно указать браузеру направление поиска путем включения в URL-адрес полного пути к файлу. Давайте разберемся, как это сделать.

Возвращаясь к нашему примеру, файлы с рецептами лежат в подкаталоге с именем *recipes*. В файле *index.html* нам необходимо создать ссылку на файл *salmon.html* в каталоге *recipes*. URL-путь сообщает браузеру, что в текущем каталоге нужно найти каталог с именем *recipes*, а затем перейти к файлу *salmon.html* (рис. 6.6):

```
<li><a href="recipes/salmon.html">Семга с чесноком</a></li>
```

Из диаграммы видно, что файл *salmon.html* находится одним уровнем ниже файла *index.html*

Из файла *index.html*:

```
<a href="recipes/salmon.html">Семга
с чесноком</a>
```

Сервер ищет каталог *recipes* в каталоге с текущим документом.

Рис. 6.6. *Указание относительного URL-адреса к документу, который находится на уровень ниже каталога текущего*

УПРАЖНЕНИЕ 6.3. СОЗДАНИЕ ССЫЛКИ НА ДОКУМЕНТ, НАХОДЯЩИЙСЯ ОДНИМ КАТАЛОГОМ НИЖЕ

Откройте файл *index.html* в папке *Примеры\глава-06\Упражнения 6-1 — 6-7\kriskitchen*. Создайте для элемента списка «Оливковая паста» ссылку на файл *tapenade.html*, который находится в каталоге *recipes*. Не забудьте правильно вложить элементы.

```
<li>Оливковая паста</li>
```

Когда все будет готово, сохраните страницу *index.html* и откройте ее в браузере. Новая ссылка должна быть действующей и открывать страницу с рецептом оливковой пасты. Если этого не происходит, проверьте разметку и убедитесь, что структура каталогов сайта «Кухня Кристины» совпадает с таковой в примерах.

Теперь создадим ссылку на файл *capellini.html*, размещенный в подкаталоге *pasta*. Все, что нужно для этого сделать — проложить маршрут к файлу *capellini.html* через два подкаталога (*recipes* и *pasta*) (рис. 6.7):

```
<li><a href="recipes/pasta/capellini.html">Капеллини с овощами и сыром пекорино</a></li>
```

Каталоги отделяются друг от друга слешем. Завершенный тег якоря сообщает браузеру: «Найди в текущем каталоге каталог с именем *recipes*. Там найдешь еще один, *pasta*, содержащий файл *capellini.html*, на который нужно перейти по ссылке».

При создании ссылки на файл, расположенный ниже текущего каталога, путь к нему должен содержать имена подкаталогов, через которые необходимо пройти, чтобы добраться до этого файла.

Теперь, когда мы прошли два уровня каталогов, вы уже должны были разобраться в принципах составления путей к файлам. Эти же принципы применяются и при составлении относительных путей, которые уходят в глубь сайта на любое число уровней. Просто начните с имени каталога, который находится в той же папке, что и текущий файл, и добавляйте в конце каждого каталога слеш до тех пор, пока не придете к файлу, на который создается ссылка.

Рис. 6.7. *Указание относительного URL-адреса к документу, находящемуся на два каталога ниже текущего*

УПРАЖНЕНИЕ 6.4. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛ, РАСПОЛОЖЕННЫЙ НА ДВА КАТАЛОГА НИЖЕ

Откройте файл *index.html* в папке *Примеры\глава-06\Упражнения 6-1 — 6-7\kriskitchen*. Создайте для элемента списка «Лингвини с соусом из моллюсков» ссылку на файл *linguine.html*, расположенный в каталоге *pasta*.

```
<li>Лингвини с соусом из моллюсков</li>
```

Когда закончите, сохраните страницу *index.html* и откройте ее в браузере. Щелкните мышью по только что созданной ссылке и прочитайте рецепт.

Создание ссылки на вышестоящий каталог

На этот раз нам нужно пойти в противоположном направлении и создать ссылку со страницы с рецептом семги обратно на главную, расположенную на уровень выше.

Специально для этой цели в Unix существует условное обозначение «точка-точка-слеш» (`../`). Если путь к файлу начинается с символов `../`, это равносильно тому, что вы бы сказали браузеру: «вернись на

уровень выше и следуй по пути к указанному файлу». Если вам приходилось искать файлы на рабочем компьютере, то знайте, что символы . . / действуют так же, как щелчок мышью по кнопке «Назад» в файловом менеджере Проводник (Windows Explorer) или по направленной влево стрелке в программе Finder (OS X).

Обозначение . . / в начале пути к файлу указывает браузеру, что для поиска необходимо подняться на один каталог выше.

Давайте начнем с того, что создадим ссылку назад на главную страницу (*index.html*) со страницы *salmon.html*. Поскольку файл *salmon.html* находится в подкаталоге *recipes*, то для перехода к файлу *index.html* нам необходимо вернуться на один уровень назад к каталогу *kriskitchen*. Этот путь указывает браузеру на необходимость подняться на один уровень и затем найти файл *index.html* (рис. 6.8):

```
<p><a href="../index.html">Вернуться на главную страницу</
a></p>
```

Обратите внимание, что в пути к файлу не нужно писать имя вышестоящего каталога (*kriskitchen*). Его заменяет обозначение . . /.

На диаграмме видно, что файл *index.html* находится на каталог выше файла *salmon.html*.

Из файла *salmon.html*:

```
<a href="../index.html">Вернуться
на главную страницу</a>
```

С помощью обозначения . . / происходит переход на один уровень вверх: из каталога *recipes* вверх к каталогу *kriskitchen*, где находится файл *index.html*.

Рис. 6.8. *Указание относительного URL-адреса к документу, находящемуся на уровень выше текущего*

УПРАЖНЕНИЕ 6.5. СОЗДАНИЕ ССЫЛКИ НА ВЫШЕСТОЯЩИЙ КАТАЛОГ

Откройте файл *tapenade.html* в папке *recipes*. В нижней части страницы вы найдете следующий абзац.

```
<p>Вернуться на главную страницу</p>
```

С помощью условного обозначения, описанного в данном разделе, сделайте из этого текста ссылку на главную страницу (*index.html*), находящуюся одним каталогом выше.

ПРИМЕЧАНИЕ

Признаться, я до сих пор иногда проговариваю про себя «на-уровень-выше, на-уровень-выше» на каждое обозначение . . /, когда пытаюсь расшифровать сложную относительную ссылку. Это помогает мне во всем разобраться.

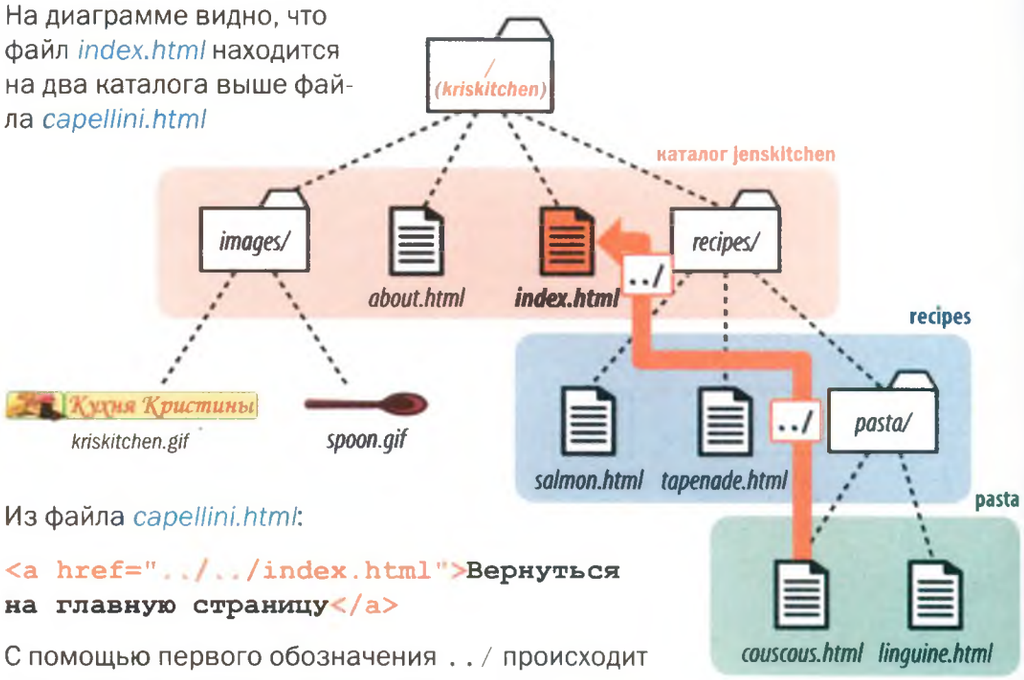
А как насчет создания ссылки на главную страницу со страницы *capellini.html*? Угадайте, как подняться на два каталога выше? Легко! Просто напишите обозначение «точка-точка-слеш» дважды (рис. 6.9).

Ссылка со страницы *capellini.html* на главную страницу (*index.html*) будет выглядеть следующим образом:

```
<p><a href="../../index.html">Вернуться на главную страницу</a></p>
```

Первое обозначение . . / осуществляет возврат в каталог *recipes*; второе . . / — в верхний каталог, где находится файл *index.html*. Опять же, не нужно писать имена каталогов — обозначения . . / сделают это за вас.

На диаграмме видно, что файл *index.html* находится на два каталога выше файла *capellini.html*



Из файла *capellini.html*:

```
<a href="../../index.html">Вернуться на главную страницу</a>
```

С помощью первого обозначения . . / происходит переход на один уровень вверх: из каталога *pasta* в каталог *recipes*.

С помощью второго обозначения . . / происходит переход из каталога *recipes* в каталог *kriskitchen*, где находится файл *index.html*.

Рис. 6.9. *Указание относительного URL-адреса к документу, расположенному двумя уровнями выше текущего*

УПРАЖНЕНИЕ 6.6. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛ, РАСПОЛОЖЕННЫЙ ДВУМЯ УРОВНЯМИ ВЫШЕ

Итак, пришла очередь проверить теорию на практике. Откройте файл *linguine.html* и, придерживаясь примера, приведенного выше, в последнем абзаце создайте обратную ссылку на главную страницу при помощи символов . . / . . /.

```
<p>Вернуться на главную страницу</p>
```

Когда все сделаете, сохраните файл и откройте его в браузере. Ссылка на главную страницу должна работать.

Пути к файлам относительно корня сайта

Любой сайт имеет *корневой каталог*, в котором содержатся все остальные каталоги и файлы сайта. Все пути к файлам, которые мы рассматривали до этого момента, строились относительно документа со ссылкой. Существует альтернативный способ записи пути: начиная с корневого, перечислять имена каталогов до тех пор, пока не будет достигнут файл, на который должен осуществляться переход по ссылке. Такой путь называется *путем относительно корня сайта*.

В целом, ссылки относительно корня сайта более предпочтительны по причине их большей гибкости.

Согласно принятому в Unix обозначению, слеш (/) в начале пути к файлу вызывает обращение к корневому каталогу. В указанной ниже ссылке путь относительно корня сайта передает следующую команду: «Зайди в самый верхний каталог этого сайта, открой каталог *recipes* и найди в нем файл *salmon.html* (рис. 6.10):

```
<a href="/recipes/salmon.html">Семга с чесноком</a>
```

Обратите внимание, что имя корневого каталога (*kriskitchen*) в URL-пути указывать не нужно — его заменяет слеш (/), поэтому браузер переходит на верхний уровень. Далее указываются каталоги, в которые браузер должен перейти.

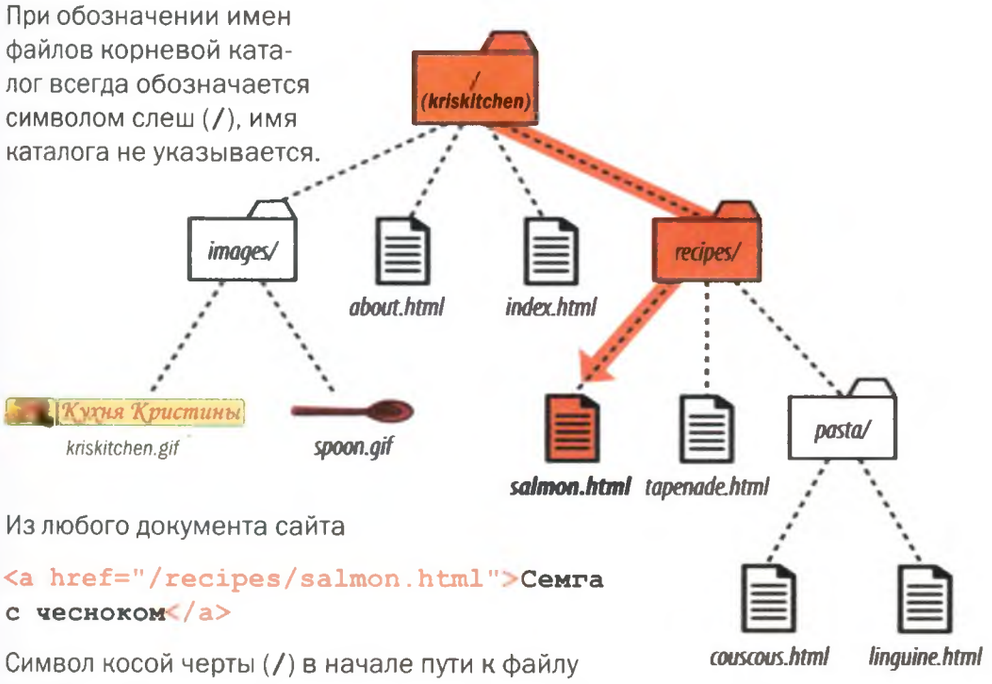


Рис. 6.10. *Указание относительного URL-пути, начиная с корневого каталога*

Поскольку такая ссылка начинается с корневого каталога, она будет работать в любом документе, независимо от того, где именно на сервере он находится. Ссылки относительно корня могут быть полезными в случае,

если содержимое сайта предполагается перемещать из каталога в каталог, а также в случае создания ссылок на динамические материалы. Кроме того, они облегчают процесс копирования и вставки ссылок из одного документа в другой. Однако есть и обратная сторона медали: такие ссылки не будут работать на локальном компьютере, поскольку они обращаются к корню логического диска. Чтобы проверить работоспособность ссылок, придется выполнить загрузку сайта на удаленный сервер.

Изображения-ссылки

Когда речь идет об определении URL-пути, атрибут **src** элемента **img** работает так же, как атрибут **href** в тегах якоря. Поскольку файлы изображений, вероятнее всего, будут находиться на вашем же сервере, то в тегах, описывающих изображение, для атрибута **src** следует указывать относительный URL-путь. Обратимся к некоторым примерам с сайта «Кухня Кристины». Во-первых, для вставки изображения на страницу *index.html* необходим следующий код:

```
<img src="images/kitchen.gif" alt="">
```

В URL-пути говорится: «Найди в текущем каталоге (*kriskitchen*) папку *images*; в ней найдешь изображение *kitchen.gif*».

Теперь перейдем к главному. Вставим изображение в файл *capellini.html*:

```
<img src="../../images/spoon.gif" alt="">
```

Этот код несколько сложнее того, с которым мы сталкивались ранее. Согласно инструкциям в данной строке, браузер должен подняться на

Небольшая помощь со стороны инструментов

Если для создания сайта вы используете инструментальные средства, основанные на принципе WYSIWYG, относительные ссылки за вас генерирует программа. Некоторые, такие как Adobe Dreamweaver и Microsoft Expression Web, обладают встроенными функциями управления сайтом, предназначенными для настройки относительных URL-путей, даже если вы измените структуру каталогов.

УПРАЖНЕНИЕ 6.7. ПОТРЕНИРУЙТЕСЬ ЕЩЕ НЕМНОГО

Прежде чем мы пойдем дальше, вы, возможно, захотите потренироваться в написании относительных ссылок, чтобы наверняка усвоить этот материал. Ответы можно записывать прямо здесь, или, если вы захотите проверить разметку в действии, вносите изменения непосредственно в файлы. Вам придется добавить туда несколько слов, которые будут задействованы в ссылке (например, в первом вопросе: «Оливковая паста»). Ответы вы найдете в Приложении А.

1. В файле *salmon.html* создайте ссылку на файл *tapenade.html*.

   `Оливковая паста`

2. В файле *capellini.html* создайте ссылку на документ *salmon.html*.

   `Семга с чесноком`

3. В документ *tapenade.html* вставьте ссылку на файл *linguine.html*.

   `Лингвини с соусом из моллюсков`

4. В файле *linguine.html* создайте ссылку на документ *about.html*.

   `О проекте`

5. В файле *tapenade. html* создайте ссылку на сайт *www.rezepty.ru*.

   `Рецепты онлайн`

два каталога выше и, оказавшись там, найти в каталоге *images* изображение с именем *spoon.gif*.

Конечно, можно было бы упростить путь, пройдя по маршруту относительно корня сайта. В этом случае путь к файлу *spoon.gif* (а также к любому другому файлу каталога *images*) имел бы следующий вид:

```
<img src="/images/spoon.gif" alt="">
```

Ценой такого упрощения станет тот факт, что вы не увидите изображений на своих местах до тех пор, пока не загрузите сайт на сервер; однако когда это произошло, обслуживание сайта упрощается.

ПРИМЕЧАНИЕ

Все пути в упражнении 6.7 можно написать относительно корня сайта, тем не менее в учебных целях составьте их относительно указанных документов.

Создание ссылки на определенную позицию страницы

Знаете ли вы, что ссылка может вести на конкретную позицию веб-страницы? Такая функция бывает полезной при организации быстрого доступа к информации, которая находится в нижней части длинной страницы с прокруткой, или возврата к началу страницы одним щелчком мыши. Иногда ссылки на определенные точки на странице называют ссылками на *фрагмент* документа.

Создание ссылки, ведущей на определенную область страницы, состоит из двух этапов. Сначала вы определяете расположение, а затем делаете ссылку на него. В следующем примере я создам в верхней части страницы алфавитный указатель, каждая буква которого будет связана ссылкой с соответствующим разделом глоссария (рис. 6.11). Когда пользователь щелкнет мышью по букве «Н», он перейдет к заголовку, начинающемуся на букву «Н», расположенному ниже на этой же странице.

ПРИМЕЧАНИЕ

Ссылки на фрагмент документа подходят для длинных страниц с прокруткой, на небольших переход заметен не будет.

Шаг 1: Определение расположения

Этот шаг я определяю для себя как установку в документе флага, к которому я легко смогу вернуться. Чтобы создать расположение, следует с помощью атрибута **id** присвоить целевому элементу уникальное имя (уникальное оно потому, что может встречаться в документе только один раз, а не потому, что должно быть броским и интересным). На профессиональном языке веб-разработчиков это *идентификатор фрагмента*.

Вы уже сталкивались с атрибутом **id** (идентификатором) в главе 5, где с его помощью мы задавали имена для обобщенных элементов **div** и **span**. Здесь же с его помощью мы присвоим элементу имя, которое будет служить идентификатором фрагмента, иными словами, расположением ссылки.

Приведем пример кода страницы глоссария. Так как нам нужно, чтобы пользователи могли переходить по ссылке напрямую к заголовку «Н», добавим к нему идентификатор и присвоим значение **startN** (рис. 6.11 1).

```
<h1 id="startN">H</h1>
```

СЕКРЕТ ВЕБ-ДИЗАЙНЕРА

Наверх!

При создании переходов на длинных текстовых страницах, общепринятой практикой является создание ссылки назад к верхней части страницы, чтобы после каждого перехода по ссылке не приходилось прокручивать страницу назад вручную.

❶ Определите расположение с помощью атрибута **id**.

```
<h2 id="startN">Н</h2>
<dl>
<dt>наследование</dt>
<dd>Принцип объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.</dd>
```

❷ Создайте ссылку на расположение. Символ # перед именем необходим для обозначения того, что это фрагмент, а не имя файла.

```
<p>... | Л | М | <a href="#startN">Н</a> | О | П ...</p>
```

❸

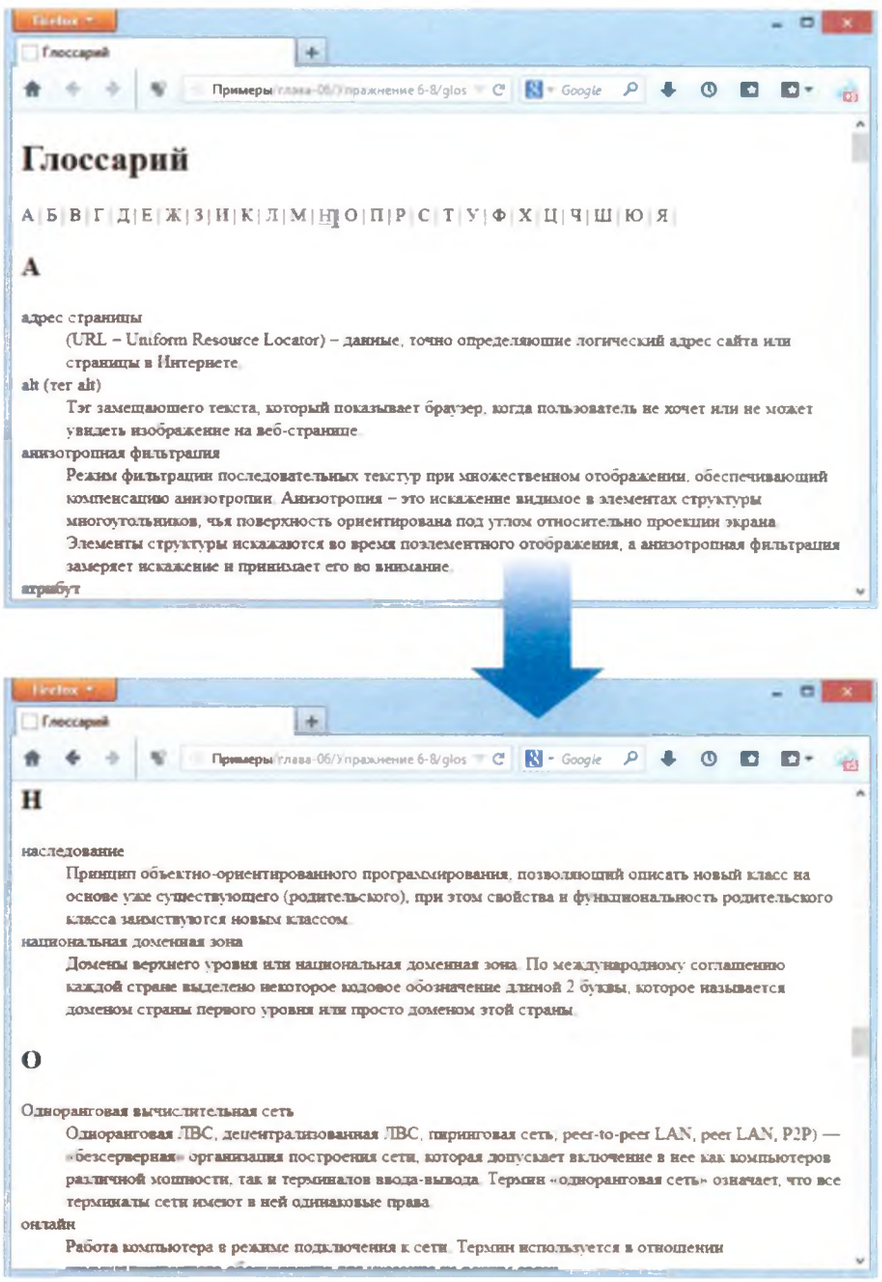


Рис. 6.11. Создание ссылки на определенное расположение на той же странице

Шаг 2: Создание ссылки на расположение

Идентификатор готов, теперь можно приступать к ссылке.

В верхней части страницы создадим ссылку на фрагмент «startN» 2. Как и при создании любой другой ссылки, для указания ее назначения понадобится элемент **a** с атрибутом **href**. Для обозначения ссылки на фрагмент перед его именем ставим хеш-символ (**#**), называемый также решеткой или знаком номера.

```
<p>... Л | М | <a href="#startN">Н</a> | О | П ...</p>
```

Теперь, если щелкнуть мышью по букве «Н» в списке в верхней части страницы, браузер перейдет вниз и отобразит раздел, начинающийся с заголовка «Н» 3.

ПРИМЕЧАНИЕ

Помните, что значения атрибута **id** могут начинаться с латинской буквы или знака подчеркивания (хотя в некоторых версиях браузера Internet Explorer знак подчеркивания может интерпретироваться некорректно).

Перед именем фрагмента указывается хеш-символ (#)

Создание ссылки на фрагмент другого документа

Чтобы создать ссылку на фрагмент, находящийся в другом документе, нужно в конец URL-пути к документу (абсолютному или относительному) добавить имя этого фрагмента. Например, ссылка на заголовок «Н» на странице глоссария из другого документа в том же каталоге будет выглядеть следующим образом:

```
<a href="glossary.html#startN">См. глоссарий, букву Н</a>
```

Возможно даже создание ссылок на определенное расположение на странице другого сайта — для этого нужно лишь указать в конце абсолютного URL-пути идентификатор фрагмента, как здесь:

```
<a   href="http://www.example.com/glossary.html#startN">См.
глоссарий, букву Н</a>
```

Вы, конечно, не можете управлять именованными фрагментами на чужих страницах сайтов (см. примечание). Чтобы получить возможность воспользоваться точками назначения, они должны быть созданы самим автором. Единственный способ узнать, есть ли они на странице и где находятся — посмотреть исходный код и найти соответствующую разметку. Если обозначенные фрагменты во внешнем документе будут удалены или перемещены, страница все равно загрузится, просто браузер, как и в случае с обычными ссылками, отобразит страницу с самого начала.

ПРИМЕЧАНИЕ

Некоторые разработчики облегчают жизнь коллегам, заранее добавляя идентификаторы в качестве якорей к началу любого тематического раздела контента (на допустимом уровне и в зависимости от сайта). Так другие люди смогут сделать ссылки обратно на любой раздел в вашем контенте.

УПРАЖНЕНИЕ 6.8. СОЗДАНИЕ ССЫЛКИ НА ФРАГМЕНТ

Хотите потренироваться создавать ссылки на определенные фрагменты документа? Зайдите в папку с материалами для данного урока и в папке *Примеры\глава-06\Упражнение 6-8* откройте файл *glossary.html*. Он выглядит точно так же, как документ, показанный на рис. 6.11.

1. Укажите заголовок «А» в теге **h2** в качестве расположения ссылки, присвоив ему с помощью идентификатора имя **startA**.

```
<h2 id="startA">A</h2>
```

2. Букву А в верхней части страницы сделайте ссылкой на именованный фрагмент. Не забудьте про символ #.

```
<a href="#startA">A</a>
```

Повторяйте шаги 1 и 2 для каждой буквы в ряду, расположенном в верхней части страницы, до тех пор, пока не усвоите суть ваших действий (или до тех пор, пока не кончится терпение). Также вы можете помочь пользователям вернуться к началу страницы.

3. Сделайте заголовок «Глоссарий» расположением с именем **top**.

```
<h1 id="top">Глоссарий</h1>
```

4. В конце каждого раздела, соответствующего отдельной букве, добавьте абзац со словом «НАВЕРХ». Сделайте это слово ссылкой на идентификатор, который вы только что разместили в верхней части страницы.

```
<p><a href="#top">НАВЕРХ</a></p>
```

Скопируйте и вставьте этот код в конец раздела для каждой буквы. Теперь ваши посетители смогут легко возвращаться к началу страницы из любой точки документа.

Открытие ссылки в новой вкладке или окне браузера

Недостаток создания внешних ссылок состоит в том, что после того, как люди перейдут по ним, они могут больше не вернуться на ваш сайт. Одним из решений этой проблемы может стать открытие целевой страницы в новой вкладке или новом окне браузера. Таким образом посетители могут посмотреть ссылку и при этом остаться на вашей странице.

Прежде чем предоставить инструкции по созданию таких ссылок, я настоятельно порекомендую вам не увлекаться этим. Вкладки в браузерах уменьшают вероятность того, что пользователи найдут путь обратно на исходную страницу. Кроме того, недостатком открытия новых вкладок/окон является ограничение доступа к специальным возможностям. Новые вкладки/окна могут запутать некоторых пользователей, в особенности тех, кто читает ваш сайт с помощью программы экранного доступа или другого вспомогательного устройства. Есть вероятность, что новые вкладки/окна будут вызывать скорее чувство раздражения, чем комфорта. Так как в браузерах пользователей обычно установлена настройка, которая блокирует всплывающие окна, они могут просто не увидеть содержимое страницы, открывающейся в новой вкладке/окне.

Поэтому подумайте хорошенько, нужно ли вам вообще отдельное окно, но я расскажу, как его создать, на случай, если у вас возникнут веские причины для этого. Выбор метода открытия ссылки в новой вкладке/окне зависит от того, нужно ли вам контролировать размер окна. Если он не имеет значения, можно работать только в рамках средств язы-

ка HTML. Однако если необходимо открыть новое окно определенного размера в пикселах, то для этого понадобится сценарий на языке JavaScript.

Открытие новой вкладки/окна средствами разметки

Чтобы открыть новую вкладку/окно при помощи разметки, нужно посредством атрибута **target** элемента якоря (**a**) сообщить браузеру имя вкладки/окна, в котором следует открыть документ при переходе по ссылке. Присвойте атрибуту **target** значение **_blank** или любое другое на ваш выбор. Не забудьте, что такой способ не позволяет контролировать размер окна, однако чаще всего оно имеет те же размеры, что и последнее открытое в браузере пользователя.

Установка параметра **target="_blank"** всегда приводит к открытию новой вкладки/окна. Например:

```
<a href="http://www.eksmo.ru/" target="_blank">Эксмо</a>
```

Если установить значение **_blank** для всех ссылок, то каждая будет открываться в новой вкладке/окне, что теоретически приведет к ситуации, когда перед пользователем окажется множество открытых вкладок/окон.

Предпочтительнее присвоить целевой вкладке/окну определенное имя, которое затем можно использовать в последующих ссылках. Имя может быть любое (**new**, **sample**, какое угодно), но оно не должно начинаться со знака подчеркивания. Следующая ссылка откроется в новой вкладке/окне с именем **display**:

```
<a href="http://www.eksmoprofi.ru/" target="display">Эксмо</a>
```

Если в каждой ссылке, присутствующей на странице, указать в качестве цели вкладку/окно с именем «display», то документы, к которым будет осуществляться переход по ссылке, будут открываться на той же самой второй вкладке/окне. К сожалению, если эта вторая вкладка/окно, в данный момент не видна пользователю, то может показаться, что ссылка не работает.

Всплывающие окна

Можно открыть окно определенного размера, содержащее различные элементы (панели инструментов, полосы прокрутки и т. д.), включить или выключить его, однако для этого потребуется сценарий на языке JavaScript. Такие окна известны как *всплывающие* и обычно используются для рекламы. На самом деле они настолько надоели, что во многих браузерах установлены настройки, отключающие их полностью.

Кроме того, в мире, где доступ к сайтам возможен с небольших мобильных устройств, всплывающим окнам фиксированного размера уже нет места.

Спам-боты

Имейте в виду, что, указывая адрес электронной почты в коде документа, вы рискуете начать получать на него нежелательные рекламные письма (известные как *спам*). Люди, которые занимаются составлением спам-списков, иногда делают это с помощью автоматизированных программ (так называемых *ботов*) для поиска в сети электронных адресов.

Если вы хотите, чтобы ваш электронный адрес отображался на странице таким образом, что люди смогут его прочитать, а роботы — нет, можно оформить его, например, так: **ivanov [собака] yandex [точка] ru**. Эта уловка не сработает для ссылки **mailto**, так как в качестве значения атрибута должен быть указан точный электронный адрес. Решением данной проблемы может стать шифрование электронного адреса при помощи сценария JavaScript. Программа Easy HTML To Any Script Converter сделает это за вас (**www.easyhtools.com/ru/ehsdescription.html**). Введите текст ссылки и электронный адрес, и программа создаст код, который вы сможете скопировать и вставить в свой документ. В противном случае, если вы не хотите получать спам, вообще не указывайте адрес электронной почты на HTML-страницах.используете инструментальные средства, основанные на принципе WYSIWYG, относительные ссылки за вас генерирует программа. Некоторые, такие как Adobe Dreamweaver и Microsoft Expression Web, обладают встроенными функциями управления сайтом, предназначенными для настройки относительных URL-путей, даже если вы измените структуру каталогов.

Тем не менее если у вас есть веская причина для открытия нового окна определенного размера, рекомендуется прочитать обучающую статью по адресу **regac.ru/sozdanie_vsplyvayuchich_okon.html**.

Ссылки на адрес электронной почты

Есть один маленький хитрый трюк: ссылка вида **mailto**. Если в ссылке задействовать протокол **mailto**, ее можно связать с электронным адресом. Тогда при щелчке мышью по такой ссылке браузер запустит ассоциированную почтовую программу, в которой откроется окно нового сообщения с уже введенным в соответствующее поле адресом.

Пример почтовой ссылки приведен ниже:

```
<a href="mailto:aivanov@yandex.ru">Написать сообщение Иванову А.И.</a>
```

Как видим, это обычный элемент якоря с атрибутом **href**, которому присвоено значение вида **mailto:name@address.ru**.

Чтобы почтовая программа смогла запуститься, в браузере должны быть установлены соответствующие настройки. Таким образом, не у всех пользователей ссылка сработает надлежащим образом. А если в качестве текста ссылки вы введете сам адрес электронной почты, им смогут воспользоваться даже те, у кого функция **mailto** не выполняется (пример прогрессивного улучшения).

Ссылки на номер телефона

Помните, что смартфоны, с которых люди заходят на ваш сайт, позволяют использовать ссылки и для совершения телефонных вызовов! Почему бы не избавить посетителей от лишнего действия, позволив им набрать номер телефона, указанный на вашем сайте, просто коснувшись его на странице? Синтаксис, используемый для формы с атрибутом **tel**: очень прост:

```
<a href="tel: 18005551212">Звоните нам бесплатно по телефону (800) 555-1212</a>
```

При касании ссылки пользователи смартфонов увидят окно сообщения с просьбой подтвердить, что они хотят позвонить по этому номеру. Данная функция поддерживается в большинстве мобильных устройств, в том числе под управлением операционных систем iOS, Android, Blackberry OS, Symbian и Windows Phone. Когда по ссылке щелкнет пользователь настольного компьютера, ничего не произойдет. Если вас это беспокоит, примените правило CSS, которое скроет ссылки от всех устройств, кроме мобильных (к сожалению, это выходит за рамки данной книги).

Ниже представлено несколько рекомендаций по использованию ссылок на номер телефона:

- Рекомендуется в качестве значения атрибута **tel:** указывать полный телефонный номер для международной связи, включая код страны, так как нельзя быть уверенным, из какой страны пользователь перейдет на ваш сайт.
- Кроме того, добавьте номер телефона в текст ссылки так, что если она не сработает, номер телефона по-прежнему будет доступен.
- В устройствах под управлением операционных систем Android и iOS есть функция, которая определяет номера телефонов и автоматически превращает их в ссылки. К сожалению, некоторые 10-значные номера (не являющиеся телефонными) также могут оказаться преобразованными в ссылки. Если в вашем документе содержатся наборы чисел, которые можно перепутать с телефонными номерами, можете отключить автоматическое обнаружение, добавив в раздел заголовка документа следующий код:

```
<meta name="format-detection" content="telephone=no">
```

Для устройств под управлением операционной системы Blackberry OS, используйте следующий код:

```
<meta http-equiv="x-rim-auto-match" content="none">
```

СОВЕТ

Обозначение . . / или их комбинация всегда указывается в начале пути к файлу и не может находиться в середине. Если вы заметите, что символы . . / расположены в середине составленного вами пути, знайте — вы сделали что-то не так.

Резюме

В действительности существует только один элемент, отвечающий за создание ссылок:

Элемент и его атрибуты	Описание
a	Элемент якоря (гипертекстовой ссылки)
href="*url-адрес***"**	Расположение целевого файла

ГЛАВА 7

ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ

Страница с текстом, но совершенно без изображений выглядит довольно скучно. Всемирная паутина приобрела массовую популярность частично благодаря тому, что среди зарослей текста стали появляться картинки. Изображения размещаются на веб-страницах двумя способами: как внутренний контент и как повторяющиеся фоновые изображения. Последние вставляются на страницу с помощью каскадных таблиц стилей, о чем пойдет речь в главе 13. С приходом стандартизованного дизайна, основанного на принципах отделения представления документа от его структуры, использование встроенных изображений в сугубо декоративных целях отошло на второй план. Более подробно читайте об этой тенденции далее во врезке «Декоративные изображения». В этой главе мы рассмотрим, как вставлять на страницу изображения с помощью встроенного элемента `img`. Он применяется, когда изображения являются частью контента, например фотографии товаров, галереи изображений, рекламные объявления, иллюстрации и т. д.

В этой главе

- Вставка изображений на веб-страницу
- Работа с атрибутами `src`, `alt`, `width` и `height`

Форматы изображений

Очень скоро мы перейдем к описанию элемента `img` и примерам разметки, но прежде всего следует знать, что на веб-страницу можно вставить не всякое изображение. Чтобы встроенное изображение могло отображаться на веб-странице, оно должно быть сохранено в формате GIF, JPEG или PNG. Эти форматы и наиболее подходящие для них типы изображений подробно обсуждаются в главе 19. Чтобы браузер распознал изображение, оно не только должно быть в правильном формате, но и файл обязан иметь такие расширения, как *.gif*, *.jpg* (или *.jpeg*) и *.png* соответственно.

Если ваше изображение представлено в другом формате, таком, например, как TIFF, BMP или EPS, то прежде, чем вставлять на веб-страницу, его следует преобразовать в формат, подходящий для Всемирной паутины. Если по какой-то причине нужно сохранить исходный формат, например, файла для САПР (систем автоматизированного проектирования) или векторного изображения, то в этом случае можно обеспе-

Декоративные изображения

Изображения, которые используются исключительно в декоративных целях, больше похожи на презентации, чем на документы со структурой и контентом. По этой причине они должны управляться таблицами стилей, а не разметкой HTML-документа. С помощью CSS можно сделать изображение фоновым или поместить в текстовый элемент (**div**, **h1**, **li** или любой другой на ваш выбор). Эти методы рассматриваются в главе 13. Определение декоративных изображений через внешние таблицы стилей и вынесение их из структуры документа имеет несколько преимуществ. Это не только делает структуру более понятной и доступной, но также упрощает задачу внесения изменений во внешний вид сайта по сравнению со структурой, в которой элементы представления перемешаны с контентом. Чтобы убедиться, насколько визуально богатой может быть веб-страница, которая не содержит в своем коде ни одного элемента **img**, зайдите на сайт CSS Zen Garden по ссылке **www.csszengarden.com** и посмотрите примеры в разделе «Select a Design».

чить к нему доступ как к *внешнему изображению*, указав ссылку непосредственно на файл с изображением:

```
<a href="architecture.eps">Открыть изображение</a>
```

Для отображения данных, с которыми браузеры не могут справиться самостоятельно, применяются вспомогательные приложения. Браузер сопоставляет расширение файла в ссылке с соответствующим этому расширению приложением. Внешнее изображение может быть открыто в окне отдельного приложения или в окне браузера, если вспомогательное приложение является надстройкой (плагином), как, например, проигрыватель QuickTime. Браузер может также спросить пользователя, какое действие следует выполнить: сохранить файл или открыть приложение вручную. Кроме того, возможно, браузер и вовсе не сумеет открыть файл.

Теперь рассмотрим элемент **img** и его необходимые и рекомендуемые атрибуты.

Элемент img

<img>

Добавляет встроенное изображение

Элемент **img** указывает браузеру: «Вставь изображение сюда». Вы видели, как он применяется для размещения баннерной графики в главах 4 и 5. Элемент следует вставлять прямо в поток текста в той позиции, где должно быть изображение, как показано в этом примере. Изображения остаются в потоке текста и не становятся причиной перевода строк. См. рис. 7.1.

```
<p>Я давно хотел поехать в Тоскану <img src="countryside.
jpg" alt="">, и я не был разочарован.</p>
```

Я давно хотел поехать в Тоскану  , и я не был разочарован.

Рис. 7.1. *По умолчанию встроенные изображения выравниваются по базовой линии окружающего их текста и поэтому не разрывают строки*

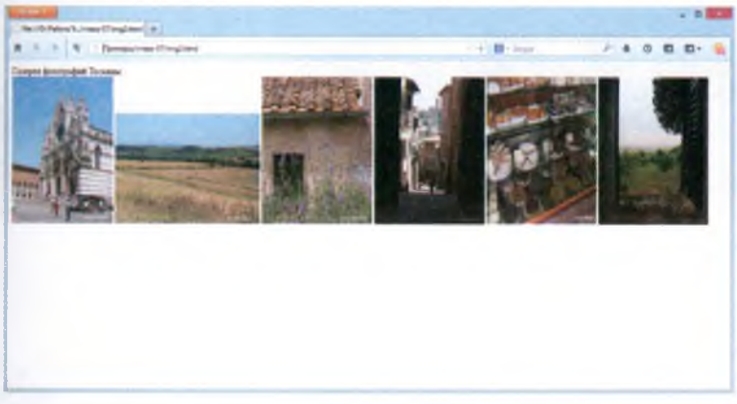
Когда браузер встречает элемент **img**, прежде чем вывести изображение на страницу, он посылает запрос на сервер и извлекает соответствующий файл. Несмотря на то что для каждого файла производится отдельный запрос, благодаря быстродействию сетевых соединений и компьютеров, страница, как правило, появляется мгновенно. Мы прекрасно знаем, что на компьютерах и мобильных устройствах с низкой скоростью подключения к Интернету вам придется подождать, пока изображения появятся по одному. При создании сайтов для мобильных устройств, разумно ограничить число запросов, отправляемых на сервер в целом, а это значит, что количество изображений на странице нужно четко продумать.

Приведенные в примере атрибуты **src** и **alt** являются обязательными. Атрибут **src** сообщает браузеру расположение файла изображения. Атрибут **alt** передает замещающий текст, который отображается на странице, если изображение недоступно. Более подробно мы поговорим об атрибутах **src** и **alt** в следующих разделах.

*Атрибуты **src** и **alt** в элементе **img** обязательны.*

Есть еще несколько моментов, которые следует отметить, говоря об элементе **img**:

- Это пустой элемент, который не содержит в себе каких-либо данных. Вы просто указываете его в той позиции текста, где должно отображаться изображение.
- Если вы решите писать по более строгим правилам синтаксиса XHTML, пустые элементы нужно будет закрыть с помощью слеша (/), например: **<img />**.
- Это встроенный элемент, поэтому он ведет себя как любой другой подобный элемент текстового потока. Рис. 7.2 демонстрирует встроенную природу элементов изображения. При изменении размеров окна браузера, изображения переходят на следующую строку и подстраиваются под новую ширину.
- Элемент **img** известен как *заменяемый элемент*, потому что при отображении страницы он заменяется внешним файлом. Этим он отличается от текстовых элементов (*не являющихся заменяемыми*), содержание которых хранится непосредственно в коде HTML-документа.
- По умолчанию нижний край изображения выравнивается по базовой линии шрифта текста, как показано на рис. 7.1 и 7.2. С помо-



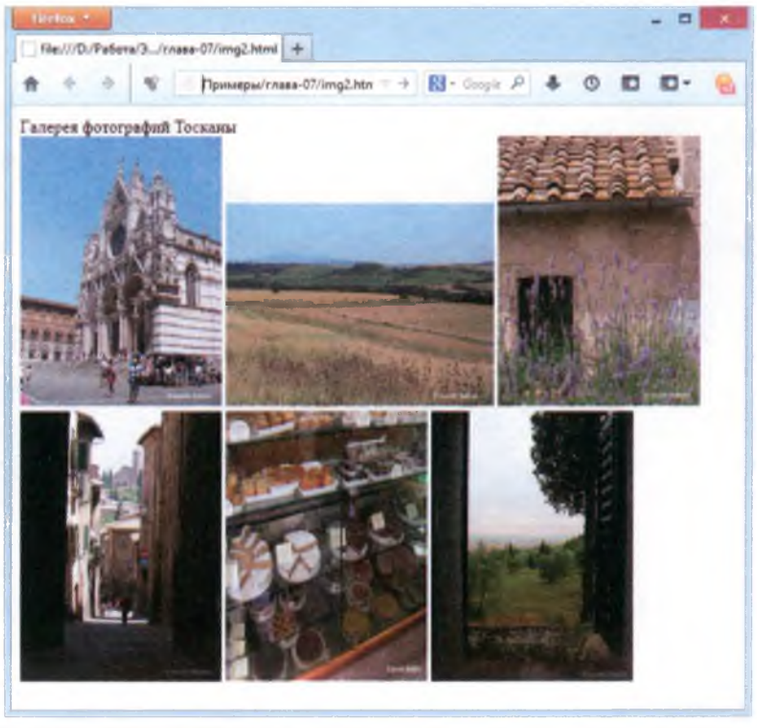


***Рис. 7.2.** Встроенные изображения — стандартная часть контента документа. При изменении окна браузера они перестраиваются под его новые размеры*

щью каскадных таблиц стилей можно выравнивать изображение по правому или левому краю, а также разрешить обтекание изображения текстом, управлять величиной расстояния между изображением и соседними объектами, задавать параметры границ вокруг изображения и изменять выравнивание по вертикали. Мы поговорим об этих стилях в части III.

Указание расположения объекта с помощью элемента `src`

`src="`*`URL`*`"`

Источник (расположение) изображения

Значение атрибута **`src`** представляет собой URL-путь к файлу с изображением. В большинстве случаев изображения, отображаемые на страницах сайта, будут находиться на том же сервере — соответственно, вы будете прописывать к ним относительные URL-пути. Если вы прочитали главу 6, то к настоящему моменту уже должны легко справляться с составлением относительных URL-адресов. В двух словах, если изображение находится в том же каталоге, что и HTML-документ, к нему можно обратиться по имени файла, указав его в атрибуте **`src`**:

```
<img src="icon.gif" alt="">
```

Разработчики обычно располагают изображения для сайта в каталоге с именем *images* или *graphics*. В них даже могут быть вложены подкаталоги с изображениями отдельно для каждого из разделов сайта. Если изображение и документ со ссылкой на него находятся в разных каталогах, то необходимо указывать относительный путь к файлу изображения.

```
<img src="/images/arrow.gif" alt="">
```

Конечно, вы также можете размещать изображения и с других вебсайтов (убедитесь только, что вы имеете на это право). Для этого задайте абсолютный URL-путь, например:

```
<img src="http://www.example.com/images/smile.gif" alt="">
```

Указание замещающего текста в атрибуте `alt`

`alt="text"`

Замещающий текст

Каждый элемент **`img`** должен также содержать атрибут **`alt`**, включающий в себя краткое описание изображения для людей, которые не смогут его увидеть; например это посетители, которые пользуются программами экранного доступа, дисплеями Брайля и даже мобильными устройствами с небольшими экранами. *Замещающий текст* (также

СОВЕТ

Преимущества кэширования

Существует один способ заставить изображения отображаться быстрее и при этом снизить нагрузку на сервер. Если у вас на разных страницах сайта размещены одинаковые изображения, проверьте, чтобы каждый элемент **`img`** ссылался на один и тот же файл изображения.

Когда браузер загружает изображение, оно сохраняется в *кэше* (место временного хранения файлов на жестком диске). Таким образом, если браузеру необходимо обновить страницу, он может просто извлечь локальную копию исходного документа и графических файлов без повторного обращения к удаленному серверу.

Если на странице или сайте многократно повторяется какое-либо изображение, браузер загружает изображение только один раз. Каждый последующий экземпляр изображения берется из локального кэша, а это приводит к снижению нагрузки на сервер и увеличению скорости отображения для конечного пользователя.

Браузер обращается к изображению в соответствии с полным путем к нему, а не только по имени файла, поэтому, чтобы воспользоваться преимуществами кэширования, убедитесь, что каждый экземпляр изображения ссылается на один и тот же файл на сервере (а не на множественные копии одного и того же файла в разных каталогах).

называемый *альтернативным текстом*) призван служить заменой графического содержимого, передавая ту же информацию.

```
<p>Если ты <img src="happyface.gif" alt="счастлив">, хлопай
в ладоши.</p>
```

Программа экранного доступа определила бы изображение и значение его атрибута **alt** следующим образом:

«Если ты рисунок счастлив, хлопай в ладоши»

Если изображение носит исключительно декоративный характер или не несет никакой полезной нагрузки, рекомендуется оставлять значение атрибута **alt** пустым, как показано в этом и других примерах данной главы. Обратите внимание, что между кавычками нет символа пробела.

```
<img src="bullet.gif" alt="">
```

Тем не менее не стоит забывать об атрибуте **alt**, поскольку такой документ не пройдет валидацию (процесс валидации документов рассматривается в главе 3). Подготовьте замещающий текст для каждого встроенного изображения, присутствующего на веб-странице, подумайте, как он будет звучать при чтении вслух и окажется ли он полезным или будет только мешать пользователю программы экранного доступа.

Замещающий текст также может пригодиться пользователям графических браузеров. Если такой посетитель веб-страницы отключил изображения в настройках браузера или изображение попросту не загрузилось, вместо него браузер отобразит замещающий текст, который даст пользователю представление о недостающих объектах. Однако, как видно из рис. 7.3, различные браузеры отображают замещающий текст по-разному или вовсе не выводят его.

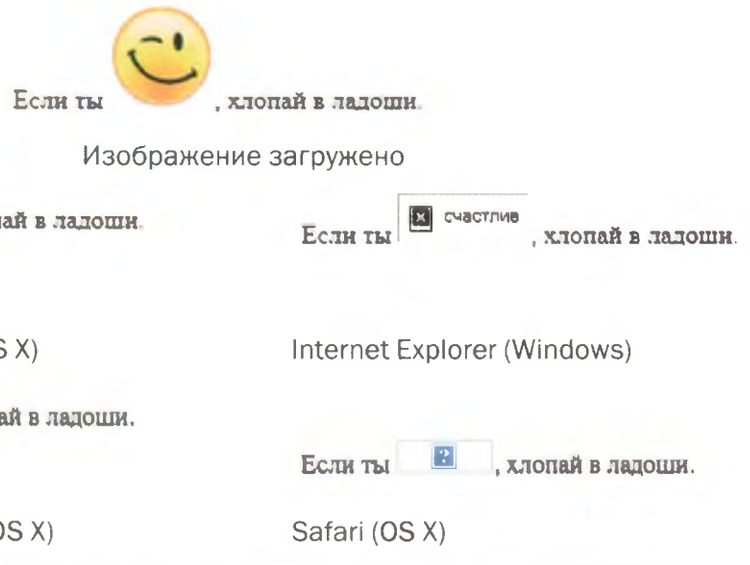


Рис. 7.3. *Не все браузеры отображают вместо изображения замещающий текст (либо вместе с пиктограммой, либо как строчный), если изображение недоступно*

ПРИМЕЧАНИЕ

Предоставление файлов изображений различного размера для элемента **img** в зависимости от величины экрана устройства осуществляется с помощью сценариев JavaScript или программы, запускаемой на сервере. Это выходит за рамки данной главы, но вы можете прочитать раздел «Адаптивные изображения» в главе 18.

Доступность изображений

Изображения и прочий нетекстовый контент представляют собой проблему для пользователей, получающих доступ ко Всемирной паутине с помощью программ экранного доступа. Замещающий текст позволяет предоставлять краткое описание изображения для тех, кто не может его увидеть. Однако существуют особые типы изображений, такие как таблицы данных и диаграммы, требующие более длинных описаний, чем можно вместить в значении атрибута **alt**.

Для очень длинных описаний продумайте варианты их размещения на странице или в отдельном документе и предоставления сноски или ссылки на него рядом с изображением. В спецификации HTML 4.01 присутствовал атрибут **longdesc** (длинное описание), но он исключен из HTML5 в связи с отсутствием поддержки.

Атрибут **longdesc** указывает на отдельный HTML-документ, содержащий подробное описание изображения, например:

```
<img src="executiveking.jpg" alt="Королевская спальня"
longdesc="executiveking-ld.html">
```

В спецификации HTML5 можно разместить длинное описание изображения с помощью элемента **figcaption**, если таковое заключено в элемент **figure**.

Этой главы не хватит, чтобы рассказать о доступности изображений все, что следовало бы. Поэтому я призываю вас начать самостоятельное исследование этого вопроса со следующих материалов:

- ГОСТ Р 52872-2007 «Интернет-ресурсы. Требования доступности для инвалидов по зрению» (**protect.gost.ru/v.aspx?control=7&id=173349**)
- Технологии WCAG 2.0; Руководство по обеспечению доступности веб-контента (**w3c.org.ru/wp-content/uploads/2011/10/русский-авторизованный-перевод-WCAG.pdf**).
- Разработка политики в области веб-доступности: международный опыт (доклад) (**perspektiva-inva.ru/files/RUS_2010%20G3ict%20Report.pdf**)
- О российском ГОСТе доступности Интернет-ресурсов для инвалидов по зрению (**tinyurl.com/6yn4239**)

Задавание размеров изображения

width="number"

Ширина изображения в пикселах

height="number"

Высота изображения в пикселах

Атрибуты **width** и **height** задают размеры изображения, измеряемые в количестве пикселов. Звучит банально, но эти атрибуты могут ускорить время загрузки страницы на несколько секунд. Браузеры исполь-

зуют заданные размеры, чтобы зарезервировать нужное пространство в макете страницы, пока изображения загружаются, и им не приходится перестраивать страницу каждый раз при появлении нового изображения.

Это замечательно, если вы создаете одну версию страницы с определенными заданными размерами изображений. Однако сегодня, с использованием адаптивного веб-дизайна, принято создавать несколько версий одного и того же изображения и отправлять небольшие изображения на маленькие портативные устройства, а более крупные — на устройства с бо́льшим размером экрана (и масштабировать изображения, чтобы подогнать их под промежуточные размеры). Если вы масштабируете изображение для адаптивного дизайна или предоставляете изображения различных размеров, не используйте в разметке атрибуты **width** и **height**.

Помня об этой проблеме, давайте посмотрим, как атрибуты **width** и **height** применяются в тех случаях, когда их использование уместно.

СОВЕТ

Определение размеров изображения в браузере

Безусловно, можно определить размеры изображения в пикселах, если открыть его в графическом редакторе, но знаете ли вы, что это можно сделать и в браузере? Откройте файл с изображением в браузере Chrome, Firefox или Safari (но только не в Internet Explorer), и его размер в пикселах отобразится в строке заголовка рядом с именем файла.

Сопоставление указываемых значений размеров изображения с фактическими

Убедитесь, что указанные вами размеры соответствуют фактическим размерам изображения. Если они различаются, браузер изменит размеры изображения в соответствии с заданными значениями (рис. 7.4).

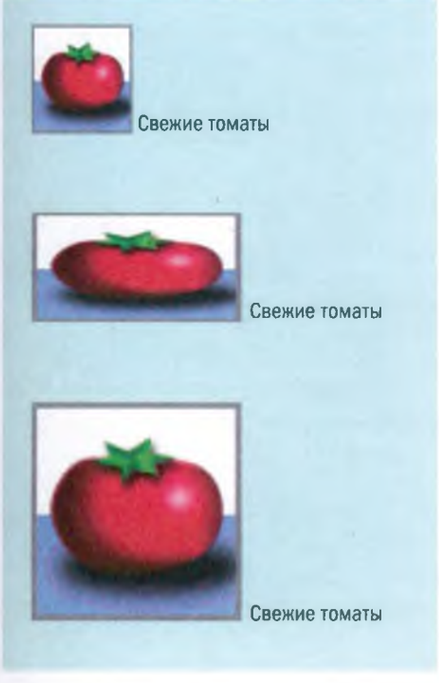


Рис. 7.4. Браузер изменит размеры изображения так, чтобы они совпадали с указанными значениями атрибутов **width** *и* **height**. *Крайне не рекомендуется изменять размеры изображения таким образом*

Несмотря на искушение изменить размеры изображений таким способом, этого делать все же не следует. Хотя изображение на странице будет выглядеть маленьким, исходное большое изображение, которое, соответственно, имеет большой размер файла, все равно станет загружаться. Лучше, если вы потратите немного времени на то, чтобы изменить размеры изображения в графическом редакторе, а затем вставите его на страницу, указав действительные размеры. Кроме того, масштабирование с помощью атрибутов зачастую приводит к размытию изображения. Фактически, если изображения при просмотре выглядят нечетко, первое, что следует сделать — это проверить, совпадают ли в точности их размеры со значениями атрибутов **width** и **height**.

УПРАЖНЕНИЕ 7.1. ВСТАВКА ИЗОБРАЖЕНИЙ И ССЫЛОК НА НИХ

Вы вернулись из Италии, и настало время порадовать друзей и семью, опубликовав кое-какие фотографии, сделанные во время путешествия. В этом упражнении вам предстоит наполнить журнал путешествий миниатюрами изображений и каждую из них сделать ссылкой на изображение более высокого качества.

Все необходимые миниатюры и фотографии я подготовила для вас заранее. Также вам будут предоставлены заготовки HTML-файлов. Все это можно найти в папке *Примеры\глава-07\Упражнение 7-1\Тоскана* на диске, прилагающемся к книге. Скопируйте папку *Тоскана* к себе на жесткий диск, сохранив ее исходную структуру. Как обычно, итоговую разметку вы найдете в приложении А.

Этот небольшой сайт состоит из главной страницы (*index.html*) и трех отдельных документов в формате HTML, каждый из которых содержит изображения с высоким разрешением (рис. 7.5). Прежде всего мы добавим миниатюры изображений, и лишь затем на соответствующих им страницах разместим полноразмерные версии. И, наконец, на основе миниатюр создадим ссылки на эти страницы. Что ж, приступим. Откройте файл *index.html*, и в качестве иллюстрации к тексту вставьте на этой странице миниатюру изображения. Первая уже добавлена:

```
<h2>Поцарелло</h2>

<p><img src="thumbnails/window_thumb.
jpg" alt="вид из окна" width="75"
height="100">Дом, в котором нам довелось
остановиться, носил название Поцарелло.
```

Изображение размещено в начале абзаца сразу после открывающего тега **<p>**.

Поскольку все миниатюры хранятся в каталоге *thumbnails*, в атрибуте **src** я указала путь к файлу в этом каталоге. Кроме того, я добавила описание изображения и размеры по ширине и высоте в пикселах.

Теперь ваша очередь. Вставьте изображение *countryside_thumb.jpg* (100 пикселов в ширину и 75 в высоту) и *sienna_thumb.jpg* (75×100 пикселов) в начала абзацев в соответствующих им разделах. Не забудьте указать путь к файлу и замещающий текст, а также размеры в пикселах.

Когда закончите, сохраните файл и откройте его в браузере, чтобы убедиться, что изображения видны и имеют правильные размеры.

1. Затем вставьте изображения в отдельные HTML-документы.

Файл *window.html* для вас уже создан:

```
<h1>Вид из окна</h1>

<p><img src="photos/window.jpg" alt="вид
из окна" width="375" height="500"></p>
```

Обратите внимание, что полноразмерные изображения находятся в папке с именем *photos*, что должно быть отражено в пути к файлу.

Следуя моему примеру, вставьте изображения на страницы *countryside.html*, *sienna.html*, и *duomo.html*. Подсказка: все изображения имеют размер 500×375 или 375×500 пикселов, то есть ориентация у них разная.

Сохраните все файлы и, открыв их в браузере, оцените свою работу.

2. Вернитесь к редактированию *index.html* и на основе миниатюр изображений создайте ссылки на соответствующие им файлы.

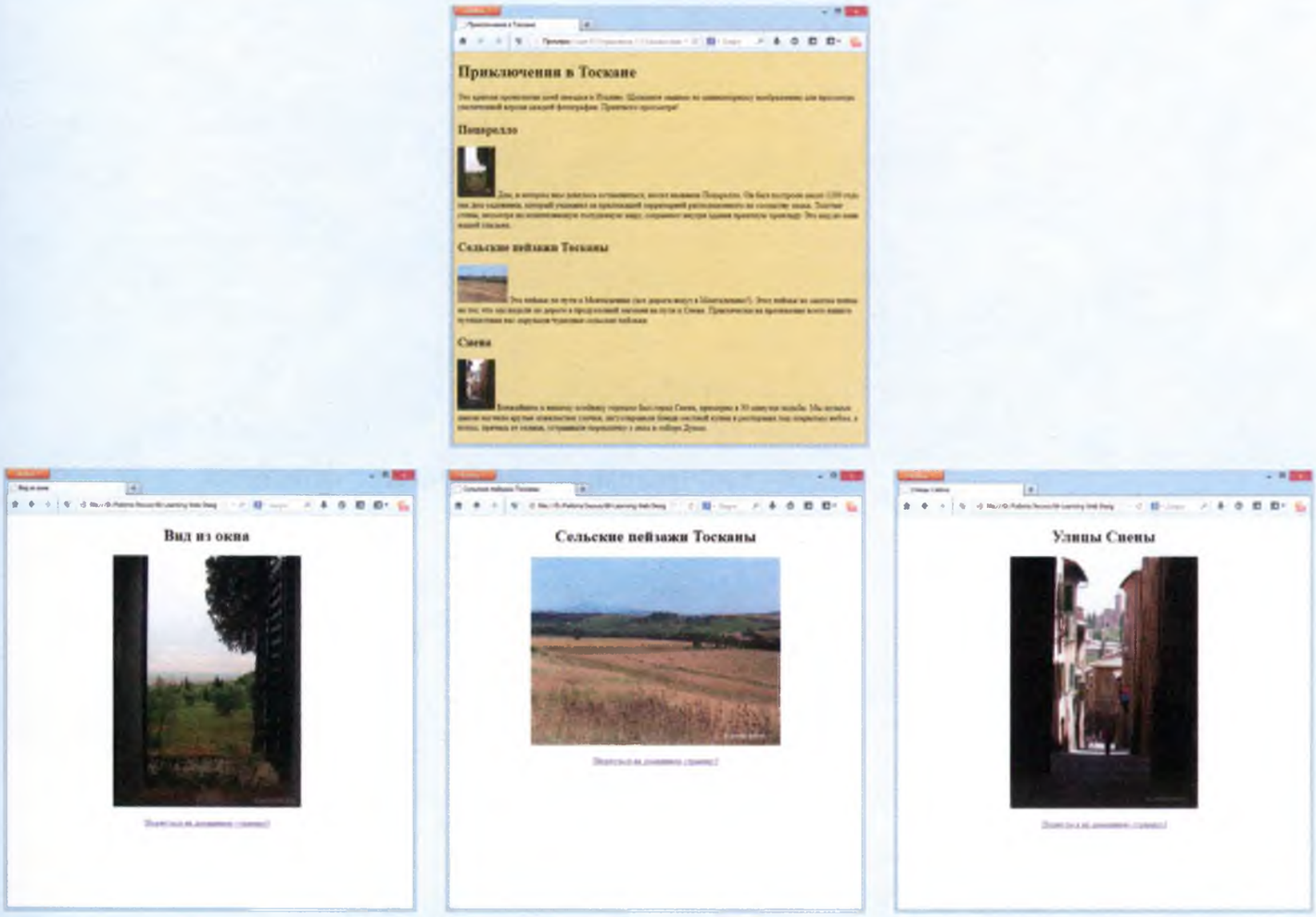


Рис. 7.5. Фотосайт журнала путешествий

Первую ссылку я уже составила.

```
<h2>Поцарелло</h2>

<p><a href="window.html"><img
src="thumbnails/window_thumb.
jpg" alt="вид из окна" width="75"
height="100"></a>Дом, в котором нам до-
велось остановиться...</p>
```

Обратите внимание, URL-путь пишется относительно текущего документа (файла *index.html*), а не относительно расположения изображения (каталога *thumbnails*).

Создайте остальные ссылки с миниатюр изображений на оставшиеся документы.

Если все изображения видны, а ссылки на все страницы и обратно на главную страницу работают, примите мои поздравления — вы отлично справились с этим уроком!

Хотите еще немного потренироваться?

Если у вас есть желание еще немного потренироваться, вы найдете три дополнительных изображения (*sweets.jpg*, *cathedral.jpg* и *lavender.jpg*) и их миниатюрные версии (*sweets_thumb.jpg*, *cathedral_thumb.jpg*, и *lavender_thumb.jpg*) в соответствующих каталогах. На этот раз вам необходимо составить собственные описания изображений на главной странице и с нуля создать HTML-документы для полноразмерных изображений.

В дополнение в каталоге *Тоскана* создайте подкаталог с именем *photopages*. Переместите все документы *html*, кроме файла *index.html*, в этот каталог, а затем обновите ссылки на этих страницах, чтобы изображения снова стали отображаться.

Фреймы

Раз уж мы заговорили о вложениях в страницу, думаю, стоит рассказать вам об элементе **iframe**, который позволяет встроить в документ отдельный HTML-документ или другой ресурс.

На странице вы увидите плавающий или встроенный «фрейм», отображающий вложенный документ с собственным набором полос прокрутки, если он слишком длинный и не помещается (рис. 7.6).

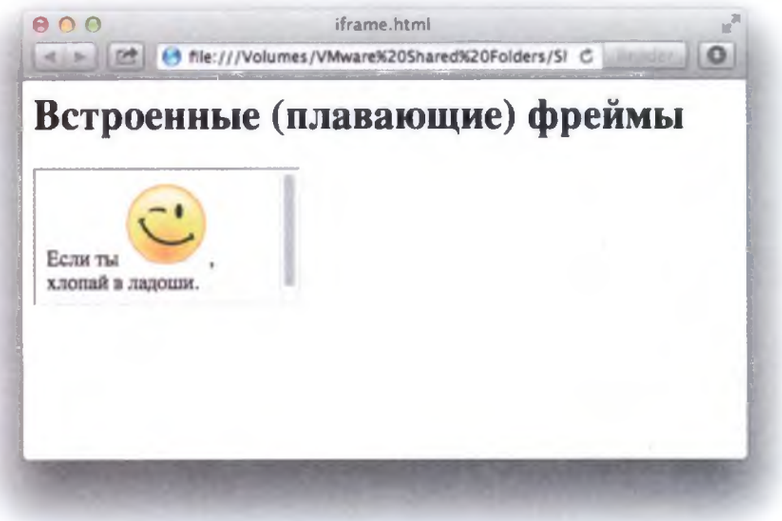


Рис. 7.6. Встроенные фреймы (добавленные с помощью элемента ***iframe****), похожи на окно браузера, открытое в другом окне и отображающее внешние документы HTML и ресурсы*

Фрейм размещается на странице так же, как изображение — нужно указать источник (**src**) его контента, высоту и ширину. Текст **iframe** отображается в браузерах, не поддерживающих данный элемент. В примере показан документ под названием *list.html*, отображаемый во встроенном фрейме.

```
<h1>Встроенные (плавающие) фреймы</h1>

<iframe src="list.html" width="400" height="250">

Ваш браузер не поддерживает встроенные фреймы. Смотрите <a
href="list.

html">список</a>.

</iframe>
```

Встроенные фреймы уже нечасто можно увидеть на просторах Всемирной паутины, но разработчики иногда используют их для размещения стороннего контента, например изолируя интерактивную рекламу или другие виджеты, чтобы они не смешивались со сценариями и контентом остальной части страницы.

Резюме

В этой главе мы рассмотрели два элемента.

Элемент и его атрибуты	Описание
img	Вставляет встроенное изображение
src="*url-адрес***"**	Местонахождение файла с изображением
alt="*текст***"**	Замещающий текст
width="*число***"**	Ширина графического объекта
height="*число***"**	Высота графического объекта
usemap="*usemap***"**	Обозначает клиентскую карту изображений
title="*текст***"**	Выводит всплывающую подсказку при наведении курсора на изображение. Может служить источником дополнительной информации о нем
iframe	Добавляет встроенный фрейм
height="*число***"**	Высота фрейма в пикселах
src="*url-адрес***"**	Ресурс, отображаемый во фрейме
width="*число***"**	Ширина фрейма в пикселах

ГЛАВА 8

РАЗМЕТКА ТАБЛИЦ

Прежде чем перейти к изучению разметки таблиц, подытожим накопленные к настоящему моменту знания. Мы рассмотрели немалый объем материала: как составить основную структуру HTML-документа и придать ему осмысленный и логичный вид с помощью разметки текста, как создавать ссылки и наполнить страницу графическим содержимым.

В этой и следующей главах будут обсуждаться способы разметки специализированного контента, которыми вы, возможно, не станете пользоваться прямо сейчас. Ничего страшного, если вы пропустите эти главы, перейдете сразу к части III и начнете экспериментировать с каскадными таблицами стилей. Главы о таблицах и формах подождут, пока вы не будете готовы к их изучению.

Вы все еще здесь? Отлично. Давайте поговорим о таблицах. Начнем с обсуждения их назначения, а затем перейдем к описанию элементов, на основе которых строятся таблицы с помощью разметки. Имейте в виду, что эта глава посвящена HTML-коду, поэтому мы сосредоточимся на преобразовании контента в таблицы, не заботясь о том, как они будут отображаться. Внешний вид (или, как говорят веб-разработчики, представление) таблицы, как и любого другого контента веб-страниц, следует настраивать посредством CSS, о чем вы узнаете в главе 18.

В этой главе

- Назначение таблиц
- Базовая структура таблиц
- Важность заголовков
- Диапазоны строк и столбцов
- Поля и интервал ячеек
- Обеспечение доступности таблиц

Использование таблиц

HTML-таблицы необходимы в случаях, когда на веб-странице нужно отобразить данные, организованные в строки и столбцы. Таблицы могут понадобиться при создании календарей, расписаний, статистики или других типов информации, как показано на рис. 8.1. Обратите внимание, что данные не обязательно должны быть числовыми. Ячейки таблицы могут содержать любую информацию, включая числа, текстовые элементы, даже изображения и мультимедийные объекты.

Если читатель просматривает в визуальном браузере страницу, на которой данные организованы в строки и столбцы, он сможет моментально сориентироваться, каким образом содержимое ячеек соотносится с соответствующимизаголовками.

Тем не менее при составлении таблицы имейте в виду, что некоторые читатели будут воспринимать ее содержимое, используя дисплей Брай-

Трудности при работе с таблицами

С большими таблицами, такими, как показанные на рис. 8.1, может быть трудно работать на устройствах с небольшим экраном. По умолчанию они уменьшаются до его размеров, отображая слишком мелкий текст, который нельзя прочитать. Пользователи могут увеличить масштаб, но тогда одновременно будут видны только несколько ячеек, и понять организацию заголовков и столбцов станет нелегко.

На момент написания книги дизайнеры только начали выяснять, как лучше обрабатывать табличный материал на маленьких экранах. Один подход заключается в замене таблицы графическим представлением, например, круговой диаграммой. Конечно, это сработает только для определенных типов информации. Простые таблицы в две или три колонки для большей гибкости можно представить в виде списка `dl`. Другой подход заключается в том, чтобы разместить шапку поверх таблицы, которая свяжет отдельный экран со всей таблицей для удобства навигации по таблице.

Крис Койер предлагает умное решение в статье по адресу **css-tricks.com/9096-responsivedata-tables/** Там описывается, как использовать CSS, чтобы переформатировать таблицу в длинный узкий список, который лучше подходит для экрана смартфона. Изучите также способ, предложенный компанией Filament Group (считайте их главными сторонниками адаптивного дизайна) на сайте **filamentgroup.com/lab/responsive_design_approach_for_complex_multicolumn_data_tables/.**

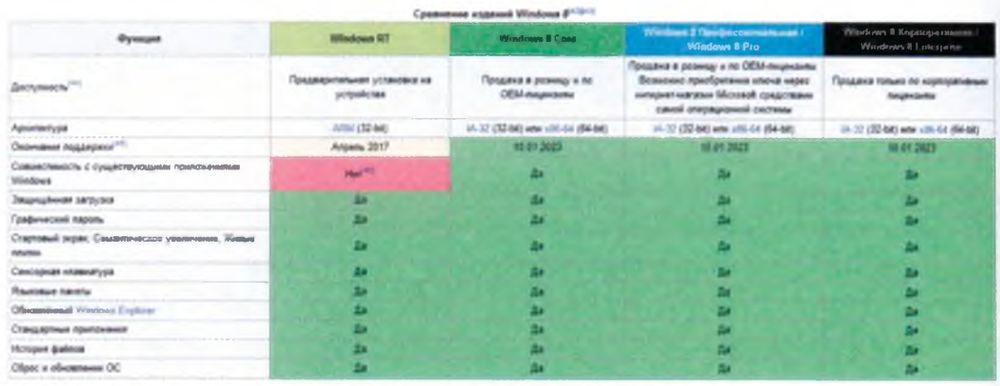
Возможно, к тому времени, как вы будете читать эту книгу, уже появятся новые решения, но в любом случае, создавая веб-контент, важно помнить о пользователях мобильных устройств с маленькими экранами.

ля, или на слух с помощью программы экранного доступа. Далее в этой главе мы рассмотрим, как сделать таблицу доступной для тех пользователей, которые не имеют возможности воспринимать информацию в визуальной форме.

Бинарные операторы SPARQL

Оператор	Тип(A)	Тип(B)	Функция	Тип результата
Логические соединения, определенные в разделе 11.4				
A || B	xsd:boolean (EBV)	xsd:boolean (EBV)	fn:logical-or(A, B)	xsd:boolean
A && B	xsd:boolean (EBV)	xsd:boolean (EBV)	fn:logical-and(A, B)	xsd:boolean
Тесты XPath				
A = B	numeric	numeric	op:numeric-equal(A, B)	xsd:boolean
A = B	simple literal	simple literal	op:numeric-equal(fn:compare(A, B), 0)	xsd:boolean
A = B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), 0)	xsd:boolean
A = B	xsd:boolean	xsd:boolean	op:boolean-equal(A, B)	xsd:boolean
A = B	xsd:dateTime	xsd:dateTime	op:dateTime-equal(A, B)	xsd:boolean
A != B	numeric	numeric	fn:not(op:numeric-equal(A, B))	xsd:boolean
A != B	simple literal	simple literal	fn:not(op:numeric-equal(fn:compare(A, B), 0))	xsd:boolean
A != B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare(STR(A), STR(B)), 0))	xsd:boolean
A != B	xsd:boolean	xsd:boolean	fn:not(op:boolean-equal(A, B))	xsd:boolean
A != B	xsd:dateTime	xsd:dateTime	fn:not(op:dateTime-equal(A, B))	xsd:boolean
A < B	numeric	numeric	op:numeric-less-than(A, B)	xsd:boolean
A < B	simple literal	simple literal	op:numeric-equal(fn:compare(A, B), -1)	xsd:boolean
A < B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), -1)	xsd:boolean
A < B	xsd:boolean	xsd:boolean	op:boolean-less-than(A, B)	xsd:boolean

w3c.org



Сравнение изданий Windows

Функция	Windows RT	Windows 8 Core	Windows 8 Профессиональная / Windows 8 Pro	Windows 8 Корпоративная / Windows 8 Enterprise
Доступность	Предварительная установка на устройства	Продажа в розницу и по OEM-лицензиям	Продажа в розницу и по OEM-лицензиям. Возможно приобретение ключа через интернет-магазин Microsoft средствами самой операционной системы	Продажа только по корпоративным лицензиям
Архитектура	ARM (32-bit)	IA-32 (32-bit) или x86-64 (64-bit)	IA-32 (32-bit) или x86-64 (64-bit)	IA-32 (32-bit) или x86-64 (64-bit)
Окончание поддержки	Апрель 2017	10 01 2023	10 01 2023	10 01 2023
Совместимость с существующими приложениями Windows	Нет	Да	Да	Да
Защищенная загрузка	Да	Да	Да	Да
Графический пароль	Да	Да	Да	Да
Стартовый экран. Семантическое увеличение. Живые плитки	Да	Да	Да	Да
Сенсорная клавиатура	Да	Да	Да	Да
Языковые пакеты	Да	Да	Да	Да
Обновленный Windows Explorer	Да	Да	Да	Да
Стандартные приложения	Да	Да	Да	Да
История файлов	Да	Да	Да	Да
Сброс и обновление ОС	Да	Да	Да	Да

wikipedia.org

Курсы валют на субботу 27 апреля 2013 года

Код	Ник	Наименование	Кол-во	Курс к руб.	Абс. изм.	Отн. изм.
036	AUD	Австралийский доллар	1	33.1562	-0.1195	-0.3587%
944	AZN	Азербайджанский манат	1	39.8311	-0.1282	-0.3206%
051	AMD	Армянских драмов	1000	76.0298	+0.2017	+0.2660%
974	BYR	Белорусский рубль	10000	36.0712	-0.1139	-0.3139%
975	BGN	Болгарский лев	1	20.7995	-0.1038	-0.4968%
986	BRL	Бразильский реал	1	15.3926	+0.2260	+0.1457%
348	HUF	Венгерских форинтов	100	13.4887	-0.0543	-0.4009%
410	KRW	Вон Республики Корея	1000	28.8716	-0.0812	-0.2804%
344	HKD	Гонконгских долларов	10	94.9321	-0.1790	-0.1880%
840	USD	Доллар США	1	31.2196	-0.0972	-0.3107%
978	EUR	ЕВРО	1	40.8418	-0.1865	-0.4561%
356	INR	Индийских рупий	100	57.4894	-0.2908	-0.5031%
398	KZT	Казахский тенге	100	20.4827	-0.0794	-0.3859%
124	CAD	Канадский доллар	1	30.4618	-0.0907	-0.2971%
417	KGS	Киргизских сомов	100	64.4998	-0.0834	-0.1291%
156	CNY	Китайский юань	10	50.8343	-0.1187	-0.2328%
428	LVL	Латвийский лат	1	58.4048	-0.2460	-0.4196%
440	LTL	Литовский лит	1	11.7761	-0.0548	-0.4652%
498	MDL	Молдавских леев	10	19.3412	-0.0790	-0.3100%
946	RON	Новых румынских леев	10	93.6487	-0.1600	-0.1706%
934	TMT	Новых туркменских манатов	1	13.9351	-0.0379	-0.2665%
578	NOK	Норвежских крон	10	53.3440	-0.0879	-0.0987%
985	PLN	Польский злотый	10	97.7996	-0.4135	-0.4210%
960	XDR	СДР (специальные права заимствования)	1	47.8282	-0.0808	-0.0844%
702	SGD	Сингапурский доллар	1	25.2239	-0.0901	-0.1974%
972	TJS	Таджикских сомони	10	65.9455	-0.4067	-0.3541%
949	TRY	Турецкая лира	1	17.3327	-0.0592	-0.3404%
860	UZS	Узбекских сумов	1000	13.1891	-0.0362	-0.2740%
980	UAH	Украинских гривен	10	36.3354	-0.1196	-0.3634%
826	GBP	Фунт стерлингов Соединенного королевства	1	48.2031	+0.2401	+0.5000%

phnet.ru

Рис. 8.1. Примеры организации таких табличных данных как, например, графики, календари или расписания

Во времена, когда стилей CSS не существовало, таблицы были единственной возможностью создавать макеты с несколькими колонками, управлять выравниванием и отступами. Такие макеты, особенно слож-

ные вложенные табличные структуры, некогда считавшиеся стандартом веб-дизайна, сейчас больше не нужны и категорически не приветствуются. Эта глава посвящена рассмотрению HTML-таблиц в том виде, в котором они задумывались.

Минимальная структура таблицы

Чтобы понять, из чего состоит таблица, давайте рассмотрим ее простейший вариант. Ниже представлена небольшая таблица с данными о калориях, состоящая из трех строк и трех столбцов.

Блюда	Калории	Жиры (г)
Куриный бульон	120	2
Салат Цезарь	400	26

На рис. 8.2 показана структура этой таблицы в соответствии с моделью в формате HTML. Все содержимое находится в ячейках, организованных в строки. Каждая ячейка содержит либо информацию о заголовке (названия столбцов, например, «Калории»), либо данные, которые могут представлять собой любой контент.

`<table>...</table>`

Содержимое таблицы (строки и столбцы)

`<tr>...</tr>`

Строка таблицы

`<th>...</th>`

Заголовок таблицы

`<td>...</td>`

Ячейка данных таблицы

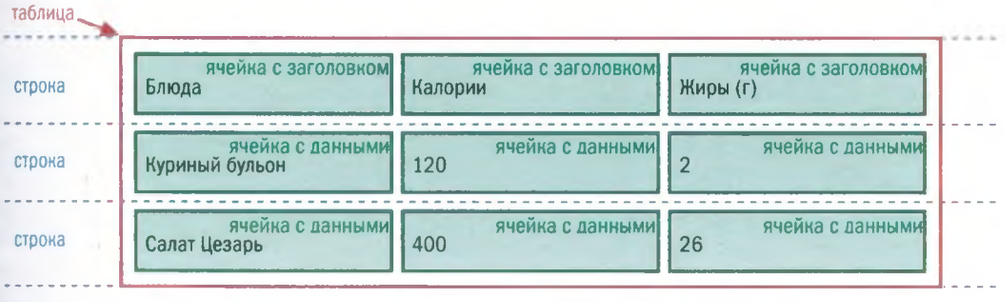


Рис. 8.2. *Таблицы состоят из строк, в которых находятся ячейки — хранилища данных*

Довольно просто, не так ли? Теперь давайте разберемся, что собой представляют эти составляющие с точки зрения элементов языка HTML (рис. 8.3).

Рис. 8.3. *Элементы, составляющие базовую структуру таблицы*

ПРИМЕЧАНИЕ

В спецификации HTML5 используются два особых элемента: **col**, обозначающий столбец, и **colgroup** — для обозначения взаимосвязанных групп столбцов. С их помощью создают информационный слой с данными о таблице, потенциально способный ускорить отображение таблицы на странице. Однако они не входят в табличную модель стандарта HTML, где центральным элементом являются строки. Подробную информацию читайте во врезке «Расширенные элементы таблицы».

На рис. 8.3 показаны элементы, определяющие таблицу (**table**): строки (**tr**, сокращение от table row — строка таблицы), ячейки **th** (сокращение от table headers — заголовки таблицы) и **td** (сокращение от table data — данные таблицы). Ячейки — это сердце таблицы, так как именно в них размещается весь основной контент. Остальные элементы призваны обеспечивать целостность ее структуры.

Единственное, чего мы не увидели, это столбцы (см. примечание). Количество столбцов в таблице определяется числом ячеек в каждой строке. Это один из потенциально сложных моментов в организации HTML-таблиц. Со строками все понятно — если в таблице должны быть три строки, просто вставьте три элемента **tr**. Со столбцами все обстоит иначе. В таблице с четырьмя столбцами каждая строка должна содержать четыре элемента **td** или **th**; столбцы являются следствием организации строк.

В исходном коде страницы разметка таблицы, изображенной на рис. 8.3, скорее выглядела бы как код в приведенном ниже примере. Элементы **td** и **th** принято сдвигать вправо, чтобы легче было ориентироваться в исходном коде. Такое расположение не влияет на отображение страницы в браузере.

```
<table>
<tr>
<th>Блюда</th>
<th>Калории</th>
<th>Жиры (г)</th>
</tr>
<tr>
<td>Куриный бульон</td>
<td>120</td>
<td>2</td>
</tr>
<tr>
<td>Салат Цезарь</td>
<td>400</td>
<td>26</td>
</tr>
</table>
```

Не забудьте, что все содержимое таблицы должно размещаться в ячейках, то есть внутри элементов **td** или **th**. В ячейке можно разместить любые данные: текст, изображения и даже другую таблицу.

Открывающий и закрывающий теги **table** обозначают начало и конец таблицы. Непосредственно в элементе **table** может находиться только

Расширенные элементы таблицы

В этом разделе образец таблицы максимально урезан, чтобы упростить структуру, пока вы изучаете принципы функционирования таблиц. Однако существуют и другие табличные элементы, а также атрибуты с более сложным семантическим описанием, повышающие доступность табличных данных для людей с ограниченными возможностями. Версия этой же таблицы с подробной разметкой выглядит следующим образом:

```
<table>
<caption>Калории и содержание жира
в наиболее популярных продуктах дневного
рациона</caption>
<col span="1" class="itemname">
<colgroup id="data">
<col span="1" class="calories">
<col span="1" class="fat">
</colgroup>
<thead>
<tr>
<th scope="col">Блюда</th>
<th scope="col">Калории</th>
<th scope="column">Жиры (г)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Куриный бульон</td>
<td>120</td>
<td>2</td>
</tr>
<tr>
<td>Салат Цезарь</td>
<td>400</td>
<td>26</td>
</tr>
</tbody>
</table>
```

Элементы группы строк

Строки или группы строк можно охарактеризовать как ячейки, относящиеся к одному заголовку, нижнему колонтитулу или к основной части таблицы, с помощью таких элементов как **thead**, **tfoot** и **tbody**, соответственно. Некоторые пользовательские агенты (синоним устройств просмотра) могут повторять строку заголовка и нижнего колонтитула таблицы на каждой странице документа. С помощью этих элементов разработчики также могут применять стили к различным частям таблицы.

Элементы группы столбцов

Столбцы могут быть определены с помощью элемента **col** или сгруппированы с помощью элемента **colgroup**. Их используют, если к данным в столбцах необходимо добавить определенный семантический контекст, также они могут пригодиться для быстрого подсчета ширины таблицы. Обратите внимание: в этих элементах нет контента, они просто описывают столбцы до того, как начинаются сами данные таблицы.

Специальные возможности

Такие элементы специальных возможностей, как подписи и сводки, предоставляют описание содержимого таблицы. Атрибуты **scope** и **headers**, предназначены для однозначного соотнесения заголовков и соответствующих им данных таблицы. Мы обсудим их позднее в этой главе.

В задачи данной книги не входит исчерпывающее исследование расширенных элементов таблиц, но если вам предстоит работать с объемными таблицами, стоит изучить эти вопросы самостоятельно. Подробную информацию вы найдете на сайте консорциума Всемирной паутины (**www.w3.org/TR/ html5**).

ПРИМЕЧАНИЕ

Согласно спецификации HTML5, таблица должна содержать «в следующем порядке: необязательный элемент **caption**, за которым могут располагаться несколько (или ни одного) элементов **colgroup**, далее необязательный элемент **thead**, затем необязательный элемент **tfoot**, после чего следуют несколько (или ни одного) элементов **tbody** или один или несколько элементов **tr**, далее необязательный элемент **tfoot** (но в общей сложности может быть только один дочерний элемент **tfoot**)».

Стилизация таблиц

После построения структуры таблицы на уровне разметки нет ничего проще, чем настроить ее внешний вид, добавив слой с информацией о стилях.

Визуальный облик таблиц можно и нужно настраивать с помощью каскадных таблиц стилей. Все средства форматирования, которые для этого потребуются, мы рассмотрим далее:

В главе 12:

- Настройка шрифта содержимого ячеек
- Цвет текста в ячейках

В главе 14:

- Размеры таблицы (ширина и высота)
- Границы
- Поля ячеек (пространство вокруг содержимого ячейки)
- Поля таблицы

В главе 13:

- Цвет фона
- Повторяющиеся фоновые изображения

В главе 18:

- Специальные методы управления границами и интервалами между ячейками

определенное количество элементов `tr` (строк). Внутри `tr` — элементы `td` или `th`. Иными словами, внутри элементов `table` и `tr` может содержаться только тот текстовый контент, что заключен между тегами `td` или `th`.

Наконец, на рис. 8.4 показано, как на простейшей веб-странице будет выглядеть таблица при отображении в браузере с настройками по умолчанию. Я знаю, что это не впечатляет. Вдохновение придет вместе с главами о таблицах CSS. Стоит отметить, что по умолчанию таблицы всегда начинаются в браузере с новой строки.

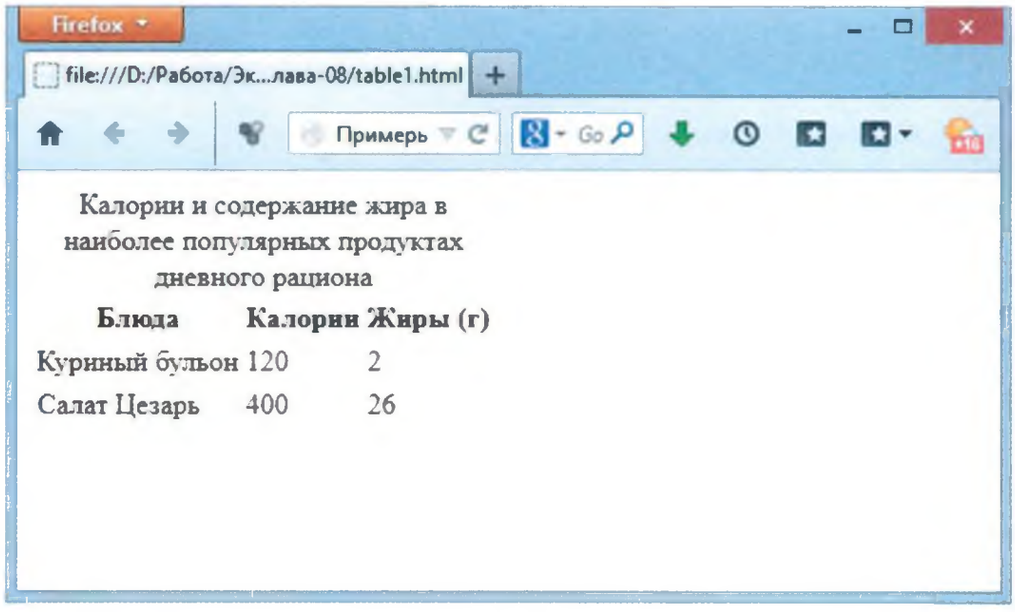


Рис. 8.4. Отображение таблицы в браузере с настройками по умолчанию

Это исходный код другой таблицы. Можете ли вы сказать, сколько строк и столбцов она будет содержать при отображении в браузере?

```
<table>
<tr>
<td>Richard Cube</td>
<td>Trance Nature</td>
<td>ZYX Music</td>
</tr>
<tr>
<td>Spliff-Guru</td>
<td>Scream Of The Future</td>
<td>Spliff-Guru Sound System</td>
</tr>
</table>
```

Если по вашим подсчетам в таблице должно быть две строки и три столбца, то вы правы. Два элемента `tr` образуют две строки, а три элемента `td` в каждой строке образуют три столбца.

Заголовки таблицы

Как видно на рис. 8.4, текст, помеченный как заголовки (элемент `th`), отображается иначе, чем остальные ячейки таблицы (элементы `td`). Разница, однако, не только во внешнем виде. Заголовки таблицы важны, потому что они предоставляют сведения или контекстную информацию о ячейках, которым предшествуют. Элемент `th` может обрабатываться альтернативными устройствами просмотра иначе, чем элементы `td`. Например, программы экранного доступа способны читать вслух заголовок каждой ячейки с данными («Блюда, салат Цезарь, калории, 400, жиры — г, 26»).

Таким образом, они представляют собой ключевые инструменты для обеспечения доступности контента таблицы. Не пытайтесь подменить заголовки, форматируя строку элементов `td` иначе, чем остальные ячейки таблицы. И наоборот, не стоит отказываться от вставки элементов `th` из-за способа отображения, который назначается им по умолчанию (полужирное начертание, выравнивание по центру). Выполняйте семантическую разметку заголовков, а затем изменяйте их внешний вид, создавая правила для стилей.

Мы рассмотрели основы. Прежде чем углубиться в более изощренные методы, попробуйте свои силы в упражнении 8.1.

УПРАЖНЕНИЕ 8.1. СОЗДАНИЕ ПРОСТОЙ ТАБЛИЦЫ.

Попробуйте написать разметку для таблицы, изображенной на рис. 8.5. Откройте текстовый редактор или просто напишите код на бумаге. Готовая разметка приведена в приложении А.

(Обратите внимание, в правилах стилей вокруг ячеек добавлена граница толщиной в 1 пиксел. Это сделано для того, чтобы было четко видно структуру таблицы. Вам не нужно включать эту границу в свою версию кода.)

Обязательно закройте все элементы таблицы. Технически, вам не *обязательно* закрывать элементы `tr`, `th` и `td`. Но желательно, чтобы у вас выработалась привычка писать аккуратный исходный код. Тогда его отображение на любых устройствах будет более предсказуемым. Если вы решите писать документы с использованием синтаксиса XHTML, закрывать элементы таблицы потребуется обязательно, чтобы документ считался валидным.

Альбом	Год
Transmissions	1993
Luciana	1994
Beyond the Infinite	1995
Bible of Dreams	1997
Shango	2000
Labyrinth	2004
Gods & Monsters	2008
Inside The Reactor	2011
The Golden Sun Of The Great East	2013

Рис. 8.5. Создайте разметку для этой таблицы

Объединение ячеек

Одной из основных особенностей структуры таблицы является *объединение* ячеек, которое подразумевает растяжение ячейки и охват ею нескольких строк или столбцов.

Это позволяет создавать сложные табличные структуры, но они имеют и свои недостатки — становится несколько сложнее ориентироваться в разметке. Заголовки или ячейки с данными объединяются посредством добавления атрибута **colspan** или **rowspan**, о чем я расскажу далее.

Объединение столбцов

ПРЕДУПРЕЖДЕНИЕ

Будьте внимательны, указывая значение параметра **colspan**; если оно превысит число столбцов таблицы, большинство браузеров самостоятельно дополнит таблицу, что, как правило, приводит к хаосу.

Объединение столбцов достигается с помощью атрибута **colspan** в элементах **td** или **th** — ячейка растягивается вправо, охватывая последующие столбцы (рис. 8.6). В нашем случае они объединяются для создания заголовка двух столбцов. (Чтобы четко было видно структуру таблицы, вокруг ячеек задана граница.)

```
<table>
<tr>
<th colspan="2">Жиры</th>
</tr>
<tr>
<td>Насыщенные жиры (г)</td>
<td>Ненасыщенные жиры (г)</td>
</tr>
</table>
```

Жиры	
Насыщенные жиры (г)	Ненасыщенные жиры (г)

Рис. 8.6. *При объединении диапазона столбцов с помощью атрибута* **colspan** *ячейка растягивается вправо, охватывая указанное число столбцов*

Обратите внимание, что в первой строке (**tr**) находится только один элемент **th**, в то время как во второй строке — два **td**. Элемент **th**, включенный в объединение, более не отображается в исходном коде. Вместо него появляется ячейка с элементом **colspan**. В каждой строке должно быть одинаковое число ячеек или равнозначный элемент **colspan**. Например есть два элемента **td** и значение параметра **colspan**, равное 2. Таким образом, предполагаемое число столбцов в каждой строке одинаково.

УПРАЖНЕНИЕ 8.2. ОБЪЕДИНЕНИЕ СТОЛБЦОВ

Попробуйте написать разметку для таблицы, изображенной на рис. 8.7. Вы можете открыть текстовый редактор или написать код на бумаге. Для отображения структуры ячеек на рисунке были добавлены границы, но в вашу таблицу их добавлять не обязательно. Готовую структуру вы найдете в приложении А.

7:00	7:30	8:00
Телеканал "Доброе утро"		
Вести.ru	Большой спорт	Рейтинг Баженова
Мультфильмы	Полезное утро	

Рис. 8.7. Потренируйтесь создавать разметку диапазонов столбцов на примере данной таблицы

Несколько советов:

- Для упрощения структуры данная таблица содержит только элементы **td**.
- Вторая строка содержит подсказку о том, что в таблице всего три столбца.
- Когда ячейка включается в объединение, ее элемент **td** не отображается в таблице.

Объединение строк

Диапазоны строк, объединенные при помощи атрибута **rowspan**, ведут себя точно так же, как объединенные диапазоны столбцов, с той лишь разницей, что диапазон ячеек задается сверху вниз и охватывает несколько строк.

В этом примере первая ячейка таблицы растягивается на три строки вниз (рис. 8.8).

```
<table>
<tr>
<th rowspan="3">Порции</th>
<td>Маленькая (150 г.)</td>
</tr>
<tr>
<td>Средняя (250 г.)</td>
</tr>
<tr>
<td>Большая (2400 г.)</td>
</tr>
</table>
```

Вновь обратите внимание, что элементы **td**, обозначающие ячейки диапазона (первые ячейки оставшихся строк), исчезли из кода страницы. Атрибут **rowspan="3"** подразумевает, что в следующих двух строках также есть ячейки, поэтому элементы **td** не нужны.

Порции	Маленькая (150 г.)
	Средняя (250 г.)
	Большая (2400 г.)

*Рис. 8.8. Атрибут **rowspan** растягивает ячейку вниз на указанное число строк*

УПРАЖНЕНИЕ 8.3. ОБЪЕДИНЕНИЕ СТРОК

Несколько советов:

- Строки всегда объединяются сверху вниз, поэтому ячейка с «апельсинами» является частью первой строки.
- Ячейки в составе объединения не отображаются в коде страницы.

Попробуйте написать разметку для таблицы, изображенной на рис. 8.9. Помните, что объединенные ячейки не отображаются в коде. Строки всегда объединяются сверху вниз, поэтому ячейка «апельсины» является частью первой строки, несмотря на то, что ее содержимое выровнено по вертикали.

Если вы работаете в текстовом редакторе, не переживайте, что ваша таблица выглядит не совсем так, как на этом рисунке. Итоговая разметка приведена в приложении А.

Рис. 8.9. *Закрепите навыки работы с диапазонами строк, составив разметку для этой таблицы*

Поля и интервалы ячеек

По умолчанию размер ячеек подстраивается под их содержимое, но зачастую необходимо оставить вокруг табличных данных немного пространства (рис. 8.10). Поскольку интервалы и поля относятся к элементам представления данных, это пространство настраивается с помощью CSS.

Поле ячейки — это расстояние между содержимым ячейки и ее границей. Для его добавления примените свойство **`padding`** к элементу **`td`** или **`th`**.

Интервал ячеек, то есть расстояние между ними, настроить сложнее. Сначала присвойте значение **`separate`** свойству **`border-collapse`** элемента **`table`**, а затем установите расстояние между ячейками, изменив значение параметра **`border-spacing`**. Раньше за поля и интервал ячеек отвечали атрибуты **`cellpadding`** и **`cellspacing`** элемента **`table`**, но в спецификации HTML5 они были признаны устаревшими.

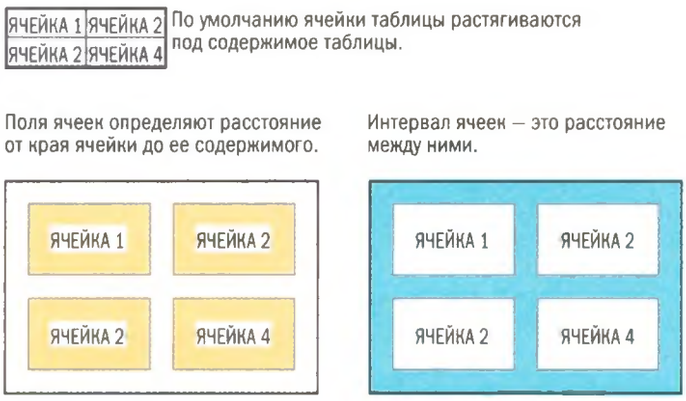


Рис. 8.10. *Поля и интервалы ячеек*

Обеспечение доступности таблиц

Веб-дизайнеру важно учитывать, как контент сайта будут использовать посетители с нарушениями зрения. Особенно трудно разобраться в табличной информации с помощью программы экранного доступа, но вы можете предпринять определенные действия по улучшению опыта взаимодействия пользователей и сделать свой контент более доступным.

Описание содержимого таблицы

Первый шаг — предоставить сведения о контенте таблицы и, возможно, способе ее структурирования, если он нестандартный.

С помощью элемента **caption** создайте краткое пояснение, которое будет отображаться рядом с таблицей. Его можно использовать для описания содержимого таблицы или предоставления подсказок по ее структуре. В элементе **table** элемент **caption** должен указываться прежде всех остальных, как показано в примере, где к таблице пищевой ценности из предыдущих разделов этой главы добавлена подпись.

```
<table>
<caption>Пищевая ценность</caption>
<tr>
<th>Блюда</th>
<th>Калории</th>
<th>Жиры (г)</th>
</tr>
... содержимое таблицы ...
</table>
```

Как видно из рис. 8.11, по умолчанию подпись выводится сверху, но с помощью свойства таблиц стилей (**caption-side**) ее можно переместить *под* таблицу.

Пищевая ценность

Блюда	Калории	Жиры (г)
Куриный бульон	120	2
Салат Цезарь	400	26

Рис. 8.11. По умолчанию подпись выводится над таблицей

Если требуются более длинные описания, рассмотрите возможность заключить таблицу в элемент **figure** и использовать элемент **figcaption**. В спецификации HTML5 содержится ряд советов, как добавить описание к таблице. Их можно найти на сайте **www.w3.org/TR/html5/tabular-data.html#tabledescriptions-techniques**.

ПРИМЕЧАНИЕ

В спецификации HTML 4.01 для элемента **table** существовал атрибут **summary**, который использовался, чтобы предоставлять для вспомогательных устройств длинные описания, не отображая их в визуальных браузерах. Однако этот атрибут не вошел в спецификацию HTML5 и его использование может вызвать ошибки при валидации кода.

Соединение ячеек и заголовков

Мы вкратце рассмотрели заголовки как прямой способ повышения доступности табличных данных, но иногда бывает трудно понять, какой заголовок к каким ячейкам относится. Например, заголовок может располагаться не в верхней части столбца, а в крайней слева или крайней справа ячейке. И хотя зрячие пользователи смогут легко понять структуру таблицы, пользователям, воспринимающим данные на слух в форме текста, общая организация будет не столь ясна. В спецификации HTML 4.01 описаны также несколько дополнительных атрибутов, связанных с обеспечением доступности таблиц.

scope

Атрибут **scope** сопоставляет заголовок таблицы со строкой, столбцом, группой строк (например, **tbody**) или группой столбцов, в которых он прописан, используя значения **row**, **column**, **rowgroup** или **colgroup** соответственно. В этом примере атрибут **scope** объявляет, что ячейка заголовка относится к текущей строке.

```
<tr>
<th scope="row">Марс</th>
<td>.95</td>
<td>.62</td>
<td>0</td>
</tr>
```

headers

В случае действительно сложных таблиц, где атрибута **scope** недостаточно (например, когда таблица содержит несколько диапазонов ячеек), чтобы однозначно сопоставить ячейку с заголовком, в элементе **td** указывают атрибут **headers**. С помощью атрибута **id** элементу заголовка (**th**) присваивают значение идентификатора. В данном примере содержимое ячейки «.38» привязано к заголовку «Диаметр Земли измеряется в километрах».

```
<th id="diameter">Диаметр Земли измеряется в километрах</th>
...много других ячеек...
<td headers="diameter">.38</td>
... много других ячеек...
```

В этом разделе лишь поверхностно затрагиваются упомянутые вопросы. Подробное рассмотрение методов создания таблиц, доступных для людей с ограниченными возможностями, не входит в задачи этой книги. Если вы хотите узнать больше, я рекомендую для начала ознакомиться с публикацией по адресу **www.webaim.org/techniques/tables**.

Резюме

В этой главе вы получили представление о верстке таблиц в HTML-документах. В упражнении 8.4 большая часть пройденного материала собрана в виде практического задания на закрепление навыков создания таблиц.

Выполнив несколько заданий, вы, вероятно, сделали вывод, что верстка кода таблицы вручную — сложное и утомительное занятие. К счастью, инструменты веб-дизайна, такие как программа Adobe Dreamweaver, предоставляют удобный пользовательский интерфейс, благодаря которому справляться с этой задачей становится проще и быстрее. Тем не менее вам будет приятно осознавать, что вы владеете оптимальными методами изменения внешнего вида таблиц, а также всеобъемлющим пониманием их структуры и соответствующей терминологии.

УПРАЖНЕНИЕ 8.4. ВЕРСТКА ТАБЛИЦ

Пришло время обобщить навыки создания таблиц, полученные в этой главе. Ваша задача состоит в том, чтобы написать исходный код страницы, содержащей приведенную на рис. 8.12 таблицу.

Пошаговое описание предстоящей работы.

1. Первое, что следует сделать — создать в текстовом редакторе новый документ и обозначить основную структуру страницы (элементы **html**, **head**, **title** и **body**). Сохраните документ под именем *table.html* в любом каталоге на ваше усмотрение.
2. Далее, чтобы при просмотре работы в браузере границы ячеек и всей таблицы были более четкими, я предлагаю вам указать в документе несколько простых правил таблиц стилей. Не обращайте внимания, если что-то из этого вам будет непонятно, просто вставьте в раздел заголовка документа приведенный ниже код.

```
<head>
<title>Верстка таблиц</title>
<style type="text/css">
td, th { border: 1px solid #CCC; }
table {border: 1px solid black; }
</style>
</head>
```

3. Теперь пора выстраивать таблицу. Я обычно начинаю с составления каркаса и добавляю столько заполнителей в виде пустых строк, сколько их должно быть в итоговой таблице (в данном случае их пять).

```
<body>
<table>
<tr></tr>
<tr></tr>
<tr></tr>
<tr></tr>
<tr></tr>
</table>
</body>
```

4. Начиная с верхней, заполните каждую строку элементами **th** и **td** слева направо, включая все необходимые объединения строк и столбцов. С начальной строкой я вам помогу.

Первая ячейка (в верхнем левом углу) сверху вниз охватывает ячейки двух строк, поэтому должна быть обозначена атрибутом **rowspan**.

Ради соблюдения единообразия во всей строке, я укажу здесь элемент **th**. Эта ячейка пуста.

```
<table>
<tr>
<th rowspan="2"></th>
</tr>
```

Ячейка второго столбца первой строки охватывает ширину двух столбцов, поэтому ее следует обозначить атрибутом **colspan**:

```
<table>
<tr>
<th rowspan="2"> </th>
<th colspan="2">Общий заголовок для двух подзаголовков</th>
</tr>
```

Ячейка в третьем столбце попадает в объединение, которое мы совершили только что, добавив атрибут **colspan**, поэтому ее не нужно включать в разметку. Ячейка четвертого столбца также охватывает две строки сверху вниз.

```
<table>
<tr>
<th rowspan="2"></th>
<th colspan="2">Общий заголовок для двух подзаголовков</th>
<th rowspan="2">Заголовок 3</th>
</tr>
```

5. Теперь ваша очередь. Продолжайте заполнять оставшиеся четыре строки таблицы элементами **td** и **th**. Подсказка: первая и последняя ячейки второй строки объединены в диапазон. И если на рисунке текст в ячейках выделен полужирным шрифтом, их следует размечать как заголовки.
6. При заполнении таблицы укажите над ней описание с помощью тега **caption**.
7. И, наконец, примените атрибут **scope**, чтобы гарантировать, что заголовки Пункт А, Пункт Б и Пункт В связаны с соответствующими строками.
8. Сохраните файл и откройте его в браузере. Таблица должна выглядеть точно так, как показано на рисунке. Если что-то не совпадает, вернитесь к разметке и исправьте ее. Столкнувшись с затруднениями, вы можете посмотреть итоговую разметку для этого задания в приложении А.

Заголовок таблицы

	Общий заголовок для двух подзаголовков		Заголовок 3
	Заголовок 1	Заголовок 2	
Пункт А	данные А1	данные А2	данные А3
Пункт Б	данные Б1	данные Б2	данные Б3
Пункт В	данные В1	данные В2	данные В3

Рис. 8.12. Верстка таблиц

Ниже представлена сводная таблица элементов, изученных в этой главе.

Элемент и его атрибуты	Описание
table	Задает элемент таблицы
td	Задает ячейку внутри таблицы
colspan="*число***"**	Число столбцов в объединенном диапазоне
rowspan="*число***"**	Число строк в объединенном диапазоне
headers="*имя заголовка***"**	Сопоставляет данные в ячейке с заголовком
th	Заголовок таблицы, соответствующий строке или столбцу
colspan="*число***"**	Число столбцов в объединенном диапазоне
rowspan="*число***"**	Число строк в объединенном диапазоне
headers="*имя заголовка***" scope="row|column|rowgroup|colgroup"**	Связывает данный заголовок с другим Сопоставляет заголовок со строкой, группой строк, столбцом, группой столбцов

Элемент и его атрибуты	Описание
`tr`	Задает строку в таблице
`caption`	Присваивает таблице видимое в браузерах описание
`col`	Определяет столбец
`colgroup`	Определяет группу столбцов
`tbody`	Обозначает группу строк в основной части таблицы
`tfoot`	Обозначает группу строк таблицы в области нижнего колонтитула
`thead`	Обозначает группу строк таблицы в области заголовка

ГЛАВА 9

ФОРМЫ

Всемирной паутине потребовалось не так много времени, чтобы превратиться из сети страниц для чтения, в место, куда приходят, чтобы *добиться цели* — купить товары, заказать билеты на самолет, подписать петиции, найти сайт, разместить объявление... список можно продолжать бесконечно!

Все эти взаимодействия обрабатываются с помощью форм.

В самом деле, в ответ на переход от страниц к действиям, спецификация HTML5 представила множество новых элементов формы и атрибутов, облегчающих пользователям процесс заполнения форм, а разработчикам — их создание. Задачи, для решения которых традиционно прибегали к помощи языка JavaScript, можно решить только с применением разметки и естественного поведения браузера. Спецификация HTML5 вводит ряд новинок, среди которых несколько элементов формы, 13 типов ввода и множество атрибутов (они перечислены в табл. 9.1 в конце главы). Некоторые из этих особенностей пока еще не поддерживаются браузерами, здесь они специально отмечены.

В этой главе я рассмотрю принципы работы веб-форм и разметку, необходимую для их создания. Также вкратце мы обсудим важность дизайна веб-форм.

В этой главе

- Принципы функционирования форм
- Элемент `form`
- Сравнение методов POST и GET
- Переменные и их значения
- Элементы формы
- Обеспечение доступности форм

Принцип работы формы

Типичная функционирующая форма состоит из двух частей. Первая — это та, которую вы видите на странице, созданная с помощью разметки HTML. Формы состоят из кнопок, текстовых полей и раскрывающихся списков (известных под общим названием *элементы формы*), предназначенных для сбора пользовательской информации. Кроме того, формы могут содержать текст и другие элементы.

Вторая часть представляет собой приложение или сценарий на стороне сервера, который обрабатывает введенные пользователем данные и возвращает соответствующий результат. Вот что приводит форму в действие. Другими словами, недостаточно просто разместить на HTML-странице компоненты формы. Разработка веб-приложений и сценариев требует определенных знаний в области программирования, что не входит в задачи данной книги. Тем не менее во врезке, которую вы встретите позже в этой главе под заголовком «Как заставить форму работать», рассказывается, как получить необходимые вам сценарии.

От ввода данных до ответа с сервера

Для всех, кто собирается создавать веб-формы, будет не лишним разобраться, что же происходит «за кадром». В этом примере рассмотрены этапы конкретной операции — сбора адресов электронной почты для формирования списка рассылки посредством простой формы. Однако они характерны для работы большинства форм.

1. Пользователь открывает в окне браузера веб-страницу с формой. Браузер видит в коде страницы разметку для элементов формы и заменяет ее соответствующими визуальными объектами — в данном примере текстовым полем ввода и кнопкой «Подписаться» (см. рис. 9.1).
2. Посетитель хотел бы подписаться на эту рассылку, поэтому он вводит свой адрес электронной почты в соответствующее поле ввода и *отправляет* его нажатием кнопки «Подписаться».
3. Браузер собирает введенную информацию, преобразует ее (см. врезку «Несколько слов о преобразовании данных формы») и отправляет на обработку приложению на стороне сервера.
4. Веб-приложение на сервере принимает отправленную информацию и обрабатывает ее (то есть выполняет над ней операции, запрограммированные в коде). В данном случае адрес электронной почты добавляется в базу данных.
5. Веб-приложение возвращает ответ. Тип ответа зависит от содержания и назначения формы. В данном случае это сообщение со словами благодарности за подписку на рассылку. Ответ других приложений может выражаться в обновлении страницы с HTML-формой и отображении свежих данных, переходе на другую страницу, связанную с формой, либо в отображении сообщения об ошибке, если поля формы заполнены неверно. Это лишь некоторые из возможных вариантов.
6. Сервер отсылает ответ веб-приложения обратно браузеру, который его отобразит. Теперь посетитель убедился, что форма работает, и его адрес электронной почты добавили в список рассылки.

Несколько слов о преобразовании данных формы

Данные формы преобразуются тем же способом, что и URL-адреса, где пробелы и другие недопустимые символы, в том числе кириллические, представляются в шестнадцатеричном коде. Например, символ пробела после сбора данных формы представлен в виде значения %20, а символ слеша (/) заменен на значение %2F. Вам в это вникать не нужно, так как браузер выполняет все преобразования автоматически.

Элемент `form`

`<form>...</form>`

Интерактивная форма

Формы вставляются на веб-страницы посредством элемента **`form`**. Он представляет собой контейнер для всего содержимого формы, включая такие элементы, как текстовые поля и кнопки, а также блочные элементы (например, **`h1`**, **`p`** и списки). Однако он не может содержать в себе другой элемент **`form`**.

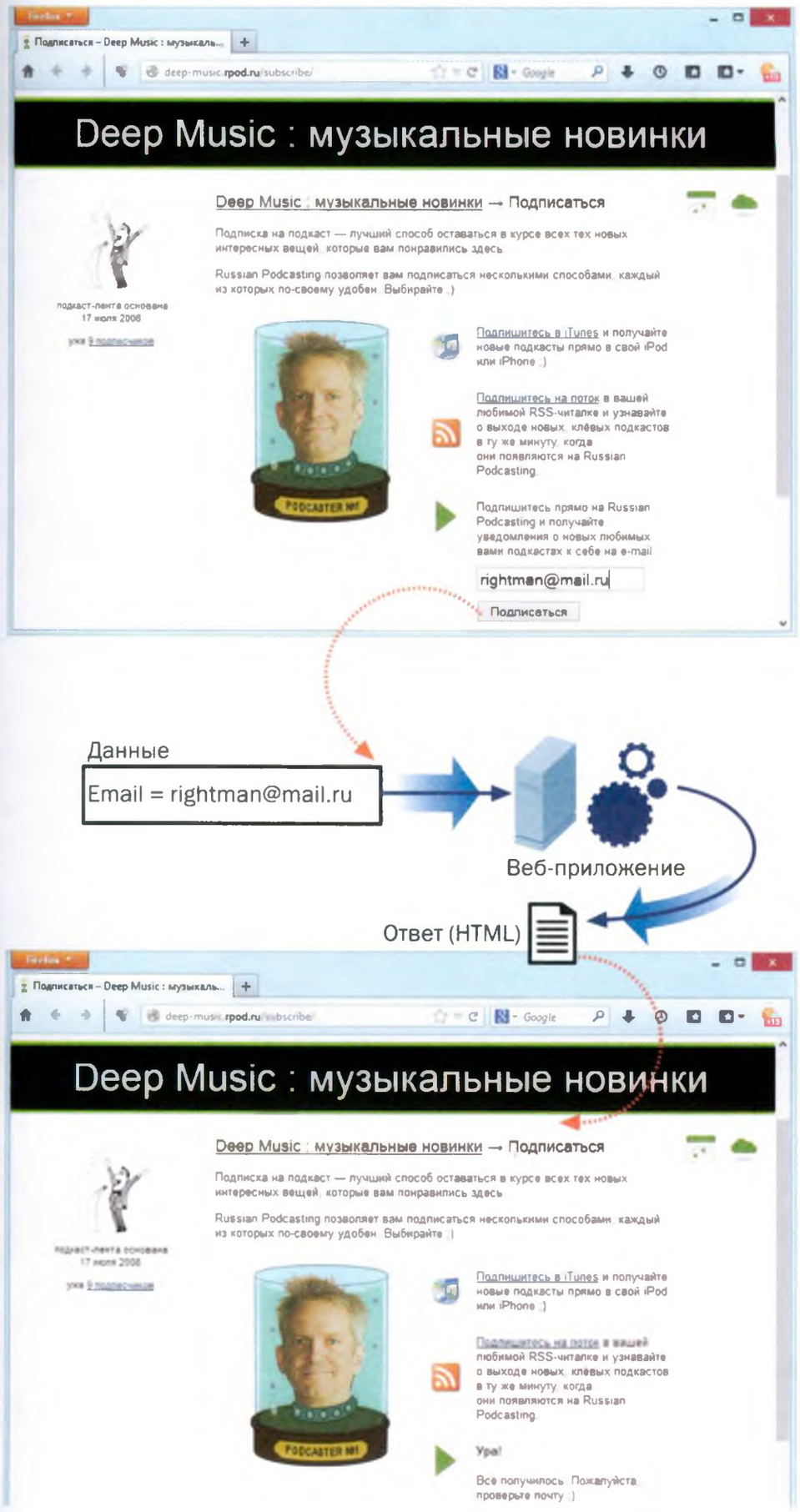


Рис. 9.1. «Закадровый» процесс в момент отправки данных

СОВЕТ

Следите за тем, чтобы элементы **form** не накладывались друг на друга и не оказались вложенными друг в друга. Прежде чем открыть новый элемент **form**, следует указать закрывающий тег для предыдущего.

Ниже приведен исходный код типичной формы, похожей на изображенную на рис. 9.1.

```
<html>
<head>
<title>Подписка на рассылку</title>
<meta charset="utf-8">
</head>
<body>
<h1> Подписка на рассылку </h1>
<form action="/cgi-bin/mailinglist.php" method="post">
<fieldset>
<legend>Подписка на подкаст — лучший способ оставаться в курсе всех тех новых интересных вещей, которые вам понравились здесь.</p>
<ol>
<li><label for="name">Имя:</label>
<input type="text" name="name" id="name"></li>
<li><label for="email">Email:</label>
<input type="text" name="email" id="email"></li>
</ol>
<input type="submit" value="Подписаться">
</fieldset>
</form>
</body>
</html>
```

ПРИМЕЧАНИЕ

В настоящее время принято окружать компоненты формы семантическими HTML-элементами, такими как списки или **div**. Из данного примера видно, что упорядоченные списки — распространенное решение. Но имейте в виду, что часто существуют стили, установленные по умолчанию, которые необходимо убрать прежде, чем применять правила стилей к спискам, особенно для мобильных версий браузеров.

В дополнение к функции контейнера для элементов формы, **form** имеет несколько атрибутов, обязательных для взаимодействия с серверными программами, обрабатывающими данные формы. Далее рассмотрим каждый из них.

Атрибут action

В атрибуте **action** указывается расположение (URL-адрес) приложения или сценария (иногда называемое *страницей сценария*) для обработки формы. В нашем примере атрибут **action** посылает данные на сервер, где они обрабатываются сценарием *mailinglist.php*.

```
<form action="/mailinglist.php" method="post">...</form>
```

Расширение `.php` указывает на то, что эта форма обрабатывается сценарием на языке PHP. Веб-формы могут обрабатываться с использованием одной из следующих технологий:

- PHP (*.php*) — язык сценариев, наиболее часто выполняемых на веб-серверах Apache, с открытым исходным кодом.
- Microsoft ASP.NET (Active Server Pages) (*.asp*) — среда программирования для серверов IIS (Microsoft Internet Information Server).
- Ruby on Rails. Ruby — язык программирования, используемый на платформе Rails. На нем написаны многие популярные веб-приложения.
- JavaServer Pages (*.jsp*) — технология на основе языка Java, похожая на ASP.
- Python — популярный язык сценариев для серверных и веб-приложений.

Существуют и другие варианты, в которых могут использоваться (как в случае с платформой Ruby on Rails) собственные расширения. Уточните у своего программиста, администратора сервера или в веб-документации правильное название и расположение программы, которую требуется указать в значении атрибута **action**.

Иногда код обработки веб-формы, например PHP, оказывается встроенным прямо в текущий HTML-файл. В этом случае оставьте значение атрибута **action** пустым, и форма будет обработана на самой странице.

Атрибут method

Атрибут **method** определяет, каким образом информация должна быть передана на сервер. В качестве примера возьмем следующие данные:

```
name = Anna Sokolova
email = annsokol@example.com
```

После того как браузер преобразует эту информацию в набор символов, пригодных для передачи на сервер, строка будет выглядеть следующим образом (если вам понадобится освежить в памяти принципы преобразования, вернитесь к предыдущей врезке):

```
username=Anna%20Sokolova&email=annsokol%40example.com)
```

Существует два метода отправки этих преобразованных данных на сервер: в элементе **form** в качестве значений атрибута **method** указываются методы POST или GET. Метод устанавливается любой на выбор, и если вы его не указали, по умолчанию будет использоваться GET. Разницу между ними мы рассмотрим в следующих разделах. В нашем примере используется метод POST:

```
<form action="/cgi-bin/mailinglist.php" method="post">...</
form>
```

Метод POST

При выборе метода POST браузер посылает на сервер отдельный запрос, состоящий из нескольких специальных заголовков, после которых

Как заставить форму работать

Чтобы создать на своем сайте работающую форму, необязательно разбираться в программировании.

Существует ряд более простых способов.

Воспользуйтесь преимуществами своего плана хостинга

В тарифные планы хостинга многих компаний включен доступ к простейшим сценариям, таким как списки рассылок. Другие компании могут в счет ежемесячной оплаты услуг хостинга обеспечивать вас всем необходимым для организации полноценного интернет-магазина. Разобраться с тем, как всем этим пользоваться, вам поможет документация на сайте или сотрудники службы поддержки хостинговой компании.

Наймите программиста

Если необходимо индивидуальное решение, можно нанять программиста, владеющего навыками программирования на серверных языках. Объясните ему, чего вы хотите добиться с помощью нужной формы, и он предложит вам подходы к решению данной задачи. Опять же, необходимо знать, имеете ли вы право устанавливать сценарии согласно текущему плану хостинга и поддерживает ли сервер язык, который вы выбрали для создания формы.

следуют данные. Содержимое этого запроса доступно только серверу, в связи с чем этот метод оптимален для передачи конфиденциальных данных, таких как реквизиты банковских карт или другая персональная информация. Метод POST также предпочтителен при передаче большого объема данных, например длинной текстовой записи, так как в отличие от метода GET, у него нет ограничений по количеству передаваемых символов.

Метод GET

При выборе метода GET преобразованные данные формы встраиваются в URL-запрос, отправляемый на сервер. URL-адрес отделяется от остальных данных символом знака вопроса, как показано в этом примере:

```
get http://www.bandname.com/cgi-bin/mailinglist.
php?username=Anna%20Sokolova&email=annsokol%40example.com
```

Метод GET подходит, если пользователям нужно предоставить возможность сохранять в закладках результаты отправки данных формы (например, список результатов поискового запроса). Поскольку данные, введенные в форму, в этом случае оказываются у всех на виду, такой метод не подходит для форм ввода персональной или финансовой информации. Кроме того, метод GET непригоден, чтобы загружать файлы на сервер.

В этой главе мы подробно рассмотрим более популярный метод POST. Теперь, когда мы ознакомились с техническими аспектами элемента **form**, можно приступать к самой сути форм — их элементам.

ПРИМЕЧАНИЕ

Слова POST и GET нечувствительны к регистру и обычно пишутся заглавными буквами. Однако в документах с кодом в спецификации XHTML значение атрибута **method** (**post** или **get**) следует указывать в нижнем регистре.

Переменные и их содержимое

В веб-формах применяются разнообразные элементы, которые позволяют пользователям вводить данные или выбирать параметры. В числе элементов — всевозможные текстовые поля ввода, кнопки, раскрывающиеся списки и некоторые элементы с особыми функциями. Их вставляют в документ при помощи специальной разметки, которую мы рассмотрим в разделе «Обзор элементов формы».

Чтобы научиться создавать простые и интуитивно понятные формы, вам, как веб-дизайнеру, необходимо познакомиться с возможностями элементов форм. Кроме того, полезно иметь представление о том, как они работают изнутри.

Атрибут name

Задача элементов формы — собирать пользовательские данные. В форме, приведенной в качестве примера несколькими страницами ранее, для сбора адреса электронной почты применялось текстовое поле ввода. Выражаясь техническим языком, слово «e-mail» — это *переменная*, по-

лученная через форму. Введенные пользователем данные («rightman@mail.ru») представляют собой *значение*, или *содержимое*, переменной.

Атрибут **name** определяет имя переменной элемента формы. В этом примере текст, который был получен элементом **textarea**, определен в качестве переменной «comment»:

```
<textarea name="comment" rows="4" cols="45" placeholder=
"Оставьте, пожалуйста, свой комментарий."</textarea>
```

После того, как пользователь введет в поле ввода свой комментарий («Это лучшая группа в мире!»), сообщение будет передано на сервер в виде пары имя/значение (переменная/содержимое):

```
comment=%D0%AD%D1%82%D0%BE%20%D0%BB%D1%83%D1%87%D1%88%D0%B
0%D1%8F%20%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0%20%D0%B2%20
%D0%BC%D0%B8%D1%80%D0%B5%21
```

Все элементы формы должны содержать в себе атрибут **name**, чтобы связанное с ней приложение могло распределить данные по типам. Атрибут **name** допустимо добавить и к элементам кнопок **submit** и **reset**, но это не обязательно, так как у них особые функции (отправка данных или очистка полей формы), не связанные со сбором данных.

Именование переменных

Нельзя называть элементы формы как попало. Веб-приложение, обрабатывающее данные формы, запрограммировано на поиск конкретных имен переменных. Создавая форму для готового приложения или сценария, необходимо подобрать конкретные имена переменных и использовать их, чтобы ваша форма и сценарий «общались на одном языке». Имена переменных можно узнать у разработчика, с которым вы сотрудничаете, у системного администратора или найти в руководстве к сценарию, используемому на сервере.

Если сценарий или приложение будет создаваться позднее, обязательно присваивайте переменным простые и описательные имена. К тому же имя каждой переменной должно быть уникальным, то есть две переменные не должны называться одинаково. Также не следует использовать в именах переменных символ пробела и буквы, отличные от латиницы. Заменяйте пробелы символами подчеркивания или точкой.

Мы рассмотрели основные характеристики элемента **form** и правила присвоения имен переменным. Теперь перейдем к важной части разметки формы — ее элементам.

Обзор элементов формы

Мы подошли к увлекательной теме — экспериментам с разметкой, которая позволяет вставлять элементы в форму. В этом разделе вы познакомитесь с элементами для создания:

ПРИМЕЧАНИЕ

Атрибуты, относящиеся к каждому из способов ввода, перечислены в табл. 9.1 в конце главы.

- Текстового поля
- Специализированного текстового поля
- Кнопок отправки данных и сброса
- Переключателей и флажков
- Раскрывающихся и прокручиваемых списков
- Элементов управления выбором и выгрузкой файла
- Скрытых элементов формы
- Элементов обозначения даты и времени (HTML5)
- Элементов числового ввода (HTML5)
- Элементов выбора цвета (HTML5)

На этом временно остановимся и предоставим вам возможность испытать элементы в действии на примере формы заказа, изображенной на рис. 9.2.

Рис. 9.2. *Форма заказа, над которой мы будем работать в упражнениях этой главы*

Большинство элементов добавляется в форму с помощью элемента **input**. Функции и внешний вид элемента **input** меняются в зависимости от значения атрибута **type**. В спецификации HTML5 используется 23 различных элемента формы. Далее мы рассмотрим их все.

Поля ввода текста

Одна из наиболее часто встречающихся задач веб-формы — ввод текстовой информации. Какой элемент для этого использовать, зависит от того, требуется ли пользователям ввести одну строку текста (**input**) или несколько (**textarea**).

ПРИМЕЧАНИЕ

В примерах разметки в данном разделе содержится элемент **label**, используемый для обеспечения доступности контента. Мы подробно обсудим его далее в этой главе в разделе «Обеспечение доступности форм», ну а пока следует *привыкнуть* к тому, как должна выглядеть разметка формы.

Однострочное текстовое поле

Один из самых простых типов элементов формы — это текстовое поле, предназначенное для ввода одного слова или строки текста. Фактически данный тип ввода установлен по умолчанию, а значит, именно это поле отобразится, если вы забудете указать атрибут **type** или введете для него неизвестное значение. Для добавления в форму следует внутри элемента **input** прописать атрибут **type** со значением **text**, как показано в данном коде и на рис. 9.3.

```
<input type="text">
```

Однострочное текстовое поле

```
<li><label>Город <input type="text" name="city" id="form-city"
value="Ваш город" maxlength="50"></label></li>
```

Хотелось бы обратить ваше внимание на некоторые атрибуты:

name

Атрибут **name** необходим для указания имени переменной.

value

Атрибутом **value** задается текст, который по умолчанию отображается в текстовом поле в момент загрузки формы. После очистки ее полей исходное значение восстанавливается.

maxlength

По умолчанию пользователи могут вводить в текстовое поле неограниченное количество символов независимо от размера поля (текст будет прокручиваться вправо, если его длина превышает количество символов, заданное шириной окна). Максимальное число символов текстового поля можно задавать с помощью атрибута **maxlength**, если этого потребует программа обработки формы.

<textarea>...</textarea>

Многострочное текстовое поле

Многострочное текстовое поле

Периодически будет возникать ситуация, когда пользователям потребуется ввести более одной строки текста. Для этого обратимся к элементу **textarea**, который при отображении в браузере заменяется на многострочную прокручиваемую область текста (рис. 9.3).

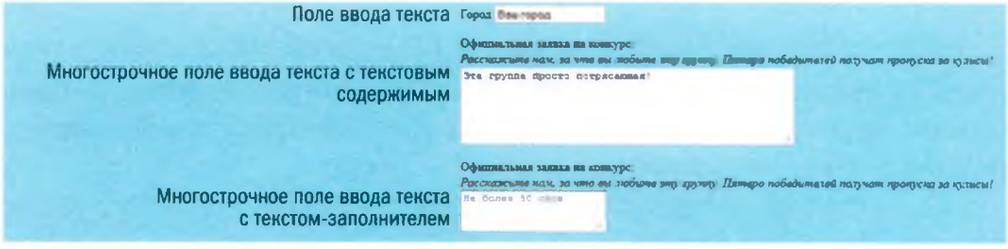


Рис. 9.3. Примеры текстовых полей веб-форм

В отличие от пустого элемента **input**, между открывающим и закрывающим тегами элемента **textarea** заключен текст. Содержимое элемента **textarea** отображается в браузере как исходное значение текстового поля. Кроме того, это содержимое отсылается на сервер при отправке формы, поэтому внимательно следите за тем, что указывается в данном поле. Часто разработчики не указывают ничего между открывающим и закрывающим тегами, а вместо этого оставляют подсказку относительно содержимого поля с помощью атрибутов **title** или **placeholder**. Новый атрибут **placeholder** в спецификации HTML5 может использоваться с элементом **textarea** и другими текстовыми типами ввода и применяется для предоставления небольшой подсказки по заполнению поля. Он не поддерживается в версиях браузера Internet Explorer предшествующих версии 10.

```
<p><label>Официальная заявка на конкурс:<br>

<em>Расскажите нам, за что вы любите эту группу. Пятеро победителей получат пропуск за кулисы!</em><br>

<textarea name="contest_entry" rows="5" cols="50">Эта группа просто потрясающая!</textarea></label></p>

<p><label>Официальная заявка на конкурс:<br>

<em>Расскажите нам, за что вы любите эту группу. Пятеро победителей получат пропуск за кулисы!</em><br>

<textarea name="contest_entry" placeholder="50 words or less">

</textarea>

</p>
```

Атрибуты **rows** и **cols** применяются для определения размера элемента **textarea** с помощью разметки, но чаще всего размер текстового поля задается с помощью CSS. Атрибут **rows** указывает, сколько строк

ПРИМЕЧАНИЕ

Стиль оформления элементов формы зависит от операционной системы и версии браузера.

Атрибуты disabled и readonly

Атрибуты **disabled** и **readonly** можно указывать в любом элементе формы, если нужно сделать так, чтобы пользователи не смогли изменить его значение или выделить его мышью. Отключенные элементы формы выбирать нельзя. Графические браузеры по умолчанию затеняют такой элемент (но, конечно, это можно изменить с помощью CSS). Состояние «отключено» (**disabled**) можно изменить только при помощи сценария. Этот атрибут весьма полезен для ограничения доступа к некоторым полям формы на основании ранее введенных данных.

Атрибут **readonly** не позволяет пользователю изменять значения элементов формы (но при этом их можно выделять). Это позволяет разработчикам устанавливать значения элементов формы в зависимости от ранее введенных данных с помощью сценариев. Данные, к которым применен атрибут **readonly**, должны заметно выделяться внешне, чтобы было понятно, что они отличаются от других введенных данных, иначе они могут смутить пользователей, попытавшихся изменить их значения.

должно отображаться в текстовой области, а атрибут **cols** определяет ширину в количестве символов. Если пользователь вводит больше текста, чем помещается в отведенное пространство, появится ползунок полосы прокрутки.

Существует еще несколько атрибутов, не показанных в примере. Атрибут **wrap** определяет, должен ли текст сохранять переносы строк при отправке формы. Значение **soft** (по умолчанию) не сохраняет переносы строк, а **hard** — сохраняет.

Атрибут **maxlength** (новый в спецификации HTML5) устанавливает ограничение на количество символов, которые могут быть введены в поле.

Специальные поля ввода текста

Помимо общих однострочных текстовых полей ввода существует несколько типов полей для ввода специальной информации, такой как пароли, поисковые запросы, адреса электронной почты, номера телефонов и URL-адреса.

Поле ввода пароля

```
<input type="password">
```

Поле ввода пароля

Поле ввода пароля работает точно так же, как обычное текстовое, с той лишь разницей, что при вводе символы заменяются на звездочки (*), маркеры (•) или какие-либо другие устанавливаемые браузером значки.

Важно отметить, что, несмотря на то, что символы в поле ввода пароля не видны обычному пользователю, сама эта информация формой не шифруется, поэтому такое поведение не должно расцениваться как мера защиты информации.

Ниже следует пример разметки поля ввода пароля. На рис. 9.4 показано, как оно будет выглядеть после заполнения пользователем.

```
<li><label for="form-pswd">Пароль:</label>
<input type="password" name="pswd" maxlength="8" id="form-pswd"></li>
```

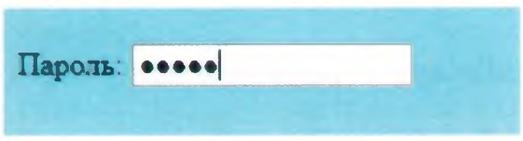


Рис. 9.4. При отображении в браузере символы пароля заменяются маркерами

Поля ввода текста в HTML5

`<input type="search">`

Поле поиска

Новый в HTML5

`<input type="email">`

Адрес электронной почты

Новый в HTML5

`<input type="tel">`

Номер телефона

Новый в HTML5

`<input type="url">`

Расположение (URL-адрес)

Новый в HTML5

До появления HTML5 вводить адреса электронной почты, номера телефонов, URL-адреса или поисковые термины можно было, только используя универсальное поле ввода текста. В HTML5 значения полей ввода **email**, **tel**, **url** и **search** подсказывают браузеру, какого рода информацию ожидать. Эти новые поля используют те же атрибуты, что и универсальное, описанное ранее (**name**, **maxlength**, **size**, и **value**), а также ряд новых атрибутов HTML5.

Каждое из этих полей ввода, как правило, отображается в виде однострочного текстового. Однако поддерживающие их браузеры могут выполнять некоторые интересные действия с дополнительной семантической информацией. Например браузер Safari в операционной системе iOS использует тип поля ввода, чтобы отобразить клавиатуру, подходящую для целей ввода, например кнопку **Найти** (Search) для поля ввода **search** или клавишу **.com**, когда устанавливается поле ввода **url** (рис. 9.5). Браузеры обычно добавляют в поле поиска значок «Очистить поле», действующий в одно касание (как правило, это маленький символ X). Браузер, поддерживающий эти поля, может проверить введенные пользователем данные, чтобы убедиться, что они верны, например, что в тексте, введенном в поле **email**, соблюдена стандартная структура адреса электронной почты (раньше для проверки требовался сценарий JavaScript). Например, браузер Firefox (рис. 9.6) отображает предупреждение, если введенные данные не соответствуют ожидаемому формату.

Не все браузеры поддерживают новые поля ввода спецификации HTML5 или поддерживают их одинаково, но хорошей новостью является то, что если поле ввода не распознано браузером, по умолчанию вместо него отображается универсальное поле ввода текста, которое прекрасно работает. Нет никаких причин, чтобы не начать их исполь-

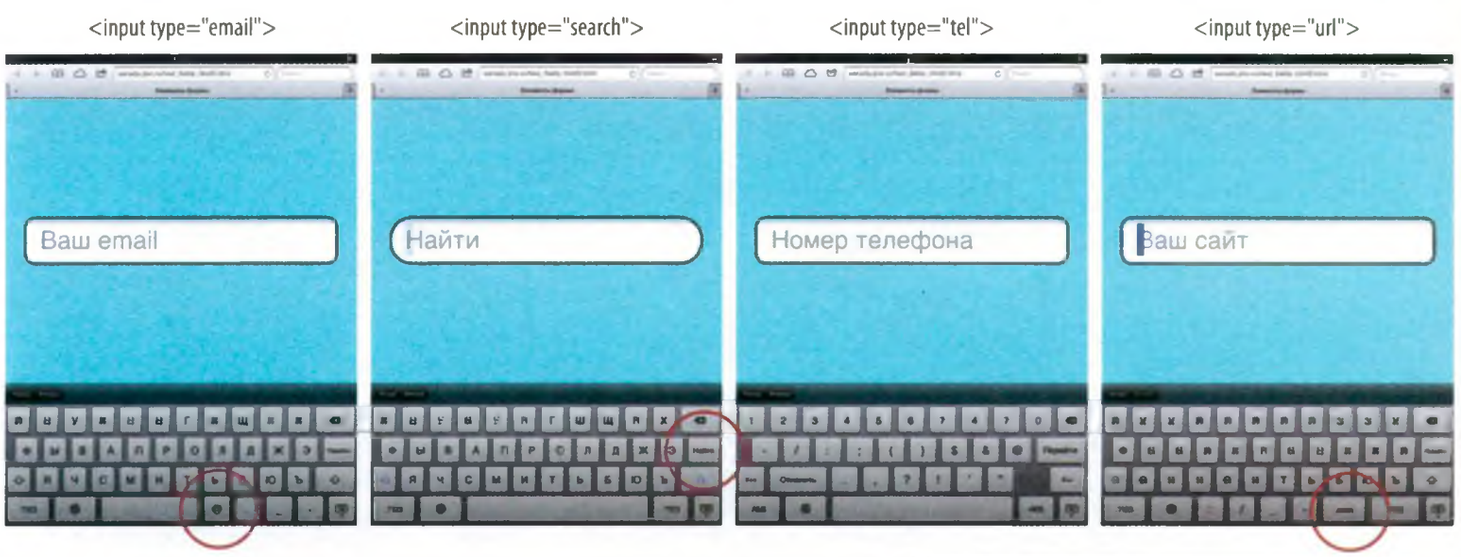


Рис. 9.5. *Браузер Safari в операционной системе iOS предоставляет пользовательские клавиатуры в зависимости от типа поля ввода*

Элемент datalist

Элемент **datalist** (Новый в HTML5) позволяет веб-дизайнеру предоставить раскрывающийся список с предложенными значениями для любого типа поля ввода. Это дает пользователю несколько вариантов быстрого ввода, но если ни один из них не подходит, пользователь может ввести собственный текст.

В элементе **datalist** предложенные значения отмечены как значения вариантов. Используйте атрибут **list** элемента **input**, чтобы связать его с атрибутом **id** соответствующего элемента **datalist**.

В следующем примере (рис. 9.7), **datalist** предлагает несколько вариантов уровней образования для ввода в текстовое поле.

```
<p>Образование полное: <input type="text"
list="edulevel" name="education">
<datalist id="edulevel">
<option value="Среднее">
<option value="Среднее специальное">
<option value="Высшее">
</datalist>
```

На момент написания книги элемент **datalist** полноценно поддерживался браузерами Firefox, Chrome и Opera, а также частично в Internet Explorer версии 10 и выше. В других браузерах, в том числе и Safari, данный элемент не поддерживается — эти браузеры будут игнорировать его и отображать простое текстовое поле ввода, что вполне приемлемо в качестве запасного варианта. Для обеспечения функциональности элемента **datalist** можно также использовать сценарии JavaScript (например, полизаполнение).

Рис. 9.7. Элемент **datalist** *создает раскрывающийся список предлагаемых значений для поля ввода текста*

зовать прямо сейчас как прогрессивное улучшение, даже если вы не можете воспользоваться преимуществами удобного ввода пользователя и проверки браузером на стороне клиента.

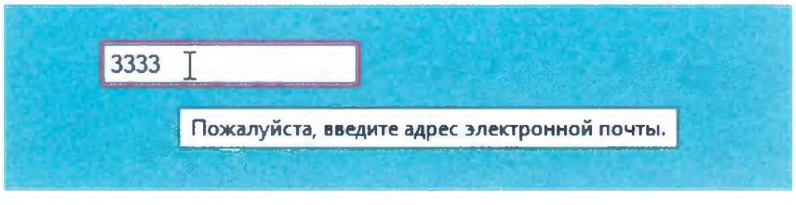


Рис. 9.6. Как элемент поддержки для проверки на стороне клиента, браузер Firefox отображает предупреждение, если введенные данные не соответствуют ожидаемому формату поля ввода **email**

Кнопки отправки данных и сброса

В форму можно добавлять самые разные кнопки, но наиболее важна кнопка отправки данных. Если щелкнуть по ней мышью, кнопка незамедлительно оправит введенные в форму данные на сервер для обработки. Кнопка сброса возвращает элементы форммы в то состояние, в котором они находились в момент загрузки формы. Другими словами, сброс формы не просто очищает ее поля, а обнуляет все установки (переключатели и т. п.).

```
<input type="submit">
```

Отправка данных формы на сервер

```
<input type="reset">
```

Сброс значений элементов формы

Другие типы кнопок

Есть несколько типов кнопок, которые не вполне подходят для изучения новичками, но для полноты картины они вынесены в эту врезку.

Кнопка с изображением

```
<input type="image">
```

Этот тип элемента ввода позволяет заменять кнопку отправки данных любым изображением на ваш выбор. Однако оно будет плоским, не похожим на трехмерную кнопку. К сожалению, с этим типом возникают проблемы, поэтому убедитесь, что добавили тщательно подобранное значение атрибута **alt**.

Кнопка произвольного назначения

```
<input type="button">
```

Если присвоить элементу **input** атрибут **type** со значением **"button"**, то будет создана кнопка, действие которой можно настраивать с помощью какого-либо языка сценариев, например, JavaScript. В самой кнопке нет встроенных функций, в отличие от кнопок отправки данных и сброса.

Элемент button

```
<button>...</button>
```

Элемент **button** — гибкое средство для создания пользовательской кнопки, похожей на те, что создаются с помощью элемента **input**. На ней отображается содержимое элемента **button** (это может быть текст и/или изображение).

Дополнительную информацию о работе с элементом **button**, смотрите на сайте **www.intuit.ru/department/internet/xhtml/11/9.html**.

Обе кнопки вставляются элементом **input**. Как упоминалось ранее, это единственные элементы формы, для которых не нужно указывать атрибут **name**, поскольку они выполняют специальные функции, которые не подразумевают ввода каких-либо данных, однако, его можно добавить при необходимости.

Кнопки отправки данных и сброса просты в использовании. Разместите их в нужной позиции, например в нижней части формы. По умолчанию на кнопке отправки данных отображена надпись «Отправить» или «Отправить запрос», а на кнопке сброса написано «Сброс» или «Очистить». При помощи атрибута **value** вы можете изменить этот текст, например, как показано на рис. 9.8.

```
<p><input type="submit"> <input type="reset" value="Начать заново"></p>
```

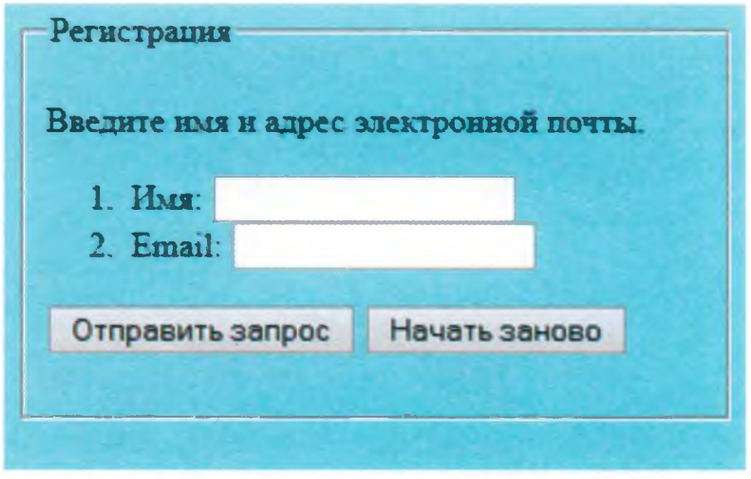


Рис. 9.8. Кнопки отправки данных и сброса

Кнопка сброса не используется в формах так часто, как раньше. Это потому, что в современном развитии форм используются сценарии JavaScript для проверки содержимого полей ввода формы в процессе заполнения, чтобы пользователи получали обратную связь. С продуманным дизайном и поддержкой меньшему числу пользователей придется сбрасывать форму, заполнив ее целиком. Тем не менее это полезная функция, о которой стоит помнить.

Теперь вы обладаете достаточными знаниями для того, чтобы создать форму заказа, изображенную на рис. 9.2. Упражнение 9.1 поможет вам сделать первые шаги.

Переключатели и флажки

Как переключатели, так и флажки позволяют посетителям сделать выбор из числа предложенных вариантов. По своим функциональным возможностям они похожи на небольшие переключатели типа вкл/выкл, управляемые пользователем. Добавить их в форму можно с помощью элемента **input**. Тем не менее они служат разным целям.

УПРАЖНЕНИЕ 9.1. СОЗДАНИЕ ФОРМЫ ЗАКАЗА

Сценарий следующий: вы — веб-дизайнер, которому необходимо создать форму заказа сандалий для сайта обувной фабрики. Представитель заказчика вручил вам схему содержимого формы (рис. 9.9) с пояснениями того, как должны выглядеть и функционировать некоторые ее элементы. К схеме приклеены стикеры, на которых программист написал сведения о сценариях и именах переменных, которые вам понадобятся в работе.

Ваша задача — превратить эту схему в функционирующую онлайн-форму. Для вас уже подготовлен скелет документа, который содержит в себе текст, минимальную разметку и стили. Этот документ, *contest_entry.html*, находится на диске в папке примеров к данной главе. Для самопроверки полный код готовой формы вы найдете в приложении А.

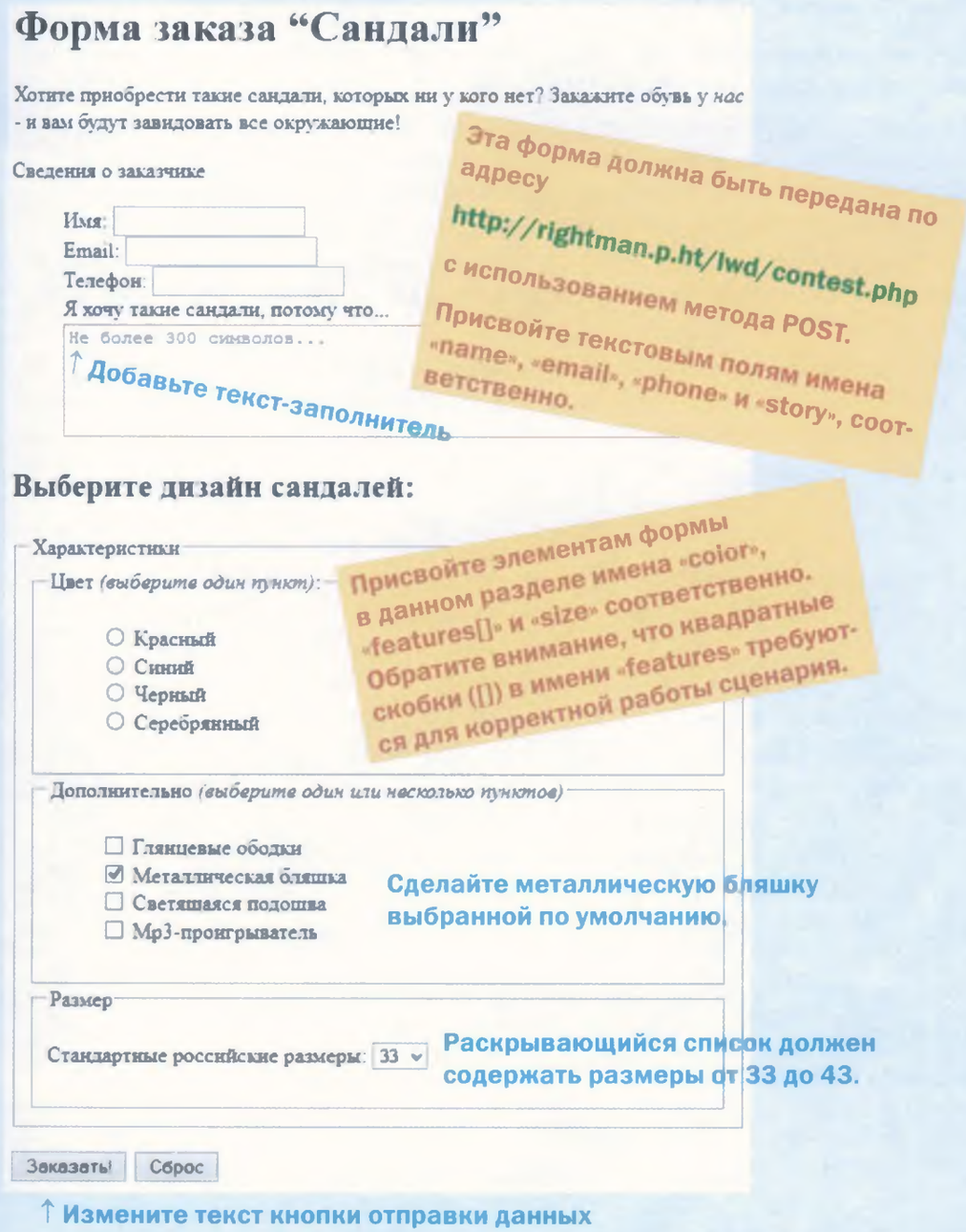


Рис. 9.9. Схема формы заказа

ПРИМЕЧАНИЕ

Если адрес http://rightman.p.ht/lwd/contest.php по каким-то причинам недоступен, используйте http://www.learningwebdesign.com/contest.php.

1. Откройте файл *contest_entry.html* в текстовом редакторе.
2. Первое, что мы сделаем, это заключим все, что идет после вводного абзаца в элемент **form**. Программист оставил нам указания относительно атрибутов **action** и **method** для использования в этой форме. В итоге элемент **form** должен выглядеть следующим образом:

```
<form action="http://rightman.p.ht/lwd/contest.php"
method="post">
...
</form>
```

3. В этом упражнении мы будем работать над разделом «Сведения о заказчике». Начните с создания трех первых текстовых полей, которые размечены как неупорядоченный список. Приведу код одного из них, а остальные два вы напишете самостоятельно.

```
<li>Имя: <input type="text" name="username"></li>
```

Совет: Выберите наиболее подходящий элемент формы для каждого поля ввода. Проверьте, все ли элементы **input** именованы так, как указано в комментариях программиста.

4. Теперь, следуя инструкциям, с новой строки добавьте многострочное текстовое поле для описания обуви. Так как мы не создаем таблицу стилей для этой формы, с помощью разметки задайте ее длину равную четырем строкам и ширину равную 60 символам (в реальной работе предпочтительно использовать CSS потому, что таблицы стилей предоставляют больше контроля при настройке).

```
<li>Я хочу такие сандалии, потому что...<br>
<textarea name="story" rows="4" cols="60" maxlength="300" placeholder="Не более 300
символов..."></textarea></li>
```

5. Мы не будем разбирать создание остальной части формы, пока не изучим еще несколько элементов, однако сейчас в самом конце формы непосредственно перед тегом **</form>** можно вставить кнопки отправки данных и сброса. Не забудьте, что нужно изменить текст кнопки отправки данных.

```
<p><input type="submit" value="Заказать!">
<input type="reset"></p>
</form>
```

6. Теперь сохраните документ и откройте его в браузере. Готовые части формы в основном должны выглядеть, как показано на рис. 9.3. Если что-то не так, значит, нужно еще немного поработать над кодом.

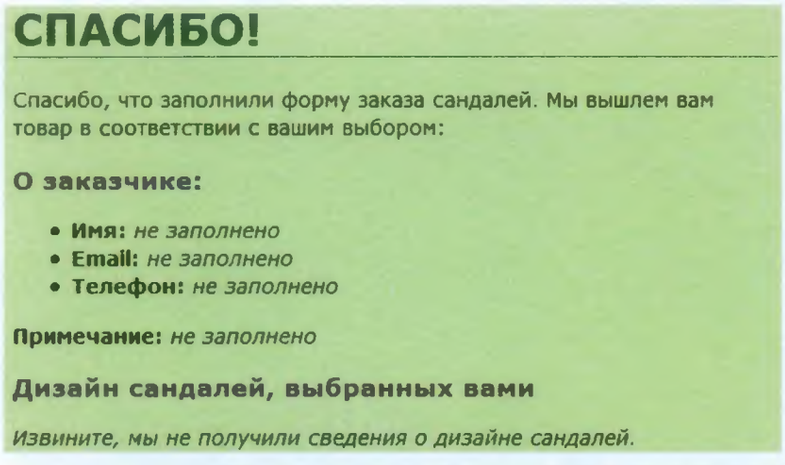


Рис. 9.10. Если форма работает, вы должны увидеть страницу благодарности, показанную на этом рисунке

Ну а если все отображается верно, проверьте форму в действии: заполните поля и отправьте данные в обработку. Вы должны получить ответ, такой как на рис. 9.7 (да, страница *contest.php* действительно существует, но заказ, к сожалению, выдуманный.)

Элемент формы, состоящий из набора переключателей, применим в случае необходимости выбора только *одного* варианта из группы, другими словами, когда варианты выбора являются взаимоисключающими (например, да или нет или мужчина или женщина).

Когда один переключатель «включен», все остальные отключены. Такой принцип работы похож на кнопки старого радио — нажимаешь одну, и все остальные тут же отскакивают*.

В группе флажков, напротив, одновременно можно выбирать столько вариантов, сколько потребуется. Поэтому группа флажков подходит для случаев, когда допустим выбор нескольких вариантов одновременно.

Переключатели

Переключатели размечают в форме с помощью элемента **input**, при этом атрибуту **type** следует присвоить значение **radio**. Атрибут **name** обязателен. Приведем основной синтаксис переключателя:

```
<input type="radio" name="имя _переменной" value="значение">
```

<input type="radio">

Переключатель

Атрибут **name** обязателен и играет важную роль в объединении нескольких переключателей в группу. Когда вы присваиваете группе переключателей одинаковые значения атрибута **name** (**age** в примере ниже), они создают несколько взаимоисключающих вариантов.

В этом примере переключатели являются частью пользовательского интерфейса, посредством которого посетитель указывает, к какой возрастной группе он относится (поскольку человек не может принадлежать более чем к одной возрастной группе, выбор переключателей в качестве элемента формы оправдан). На рис. 9.11 показано, как переключатели отображаются в браузере.

```
<p>Сколько Вам лет?</p>

<ol>

<li><input type="radio" name="age" value="menee24"checked
="checked">младше 24</label></li><li><input type="radio"
name="age" value="25-34">от 25 до 34</li><li><input
type="radio" name="age" value="35-44">от 35 до 44</
li><li><input type="radio" name="age" value="bolee45">старше
45</li>

</ol>
```

ПРИМЕЧАНИЕ

Я исключила элементы **fieldset** и **label** из примеров кода переключателей, флажков и раскрывающихся списков для того, чтобы сохранить структуру разметки как можно более простой. В разделе «Обеспечение доступности форм» вы узнаете, почему важно включить их в разметку для всех элементов формы.

Обратите внимание, что все элементы **input** имеют одно и то же имя переменной («age»), но их значения различны. Так как данные кнопки представляют собой переключатели, только одна из них может быть установлена в активное положение, и соответственно, для обработки на сервер будет отправлено только одно значение.

Наличие атрибута **checked** в элементе **input** указывает на то, что переключатель при загрузке формы будет установлен в это положение.

* Отсюда и название на англ. — radio button (примеч. ред.)

ПРИМЕЧАНИЕ

Как видно из примера ниже, в XHTML-документах атрибуту **checked** следует явным образом присваивать значение **checked**.

```
<input type="radio"
name="foo"
checked="checked" />
```

Однако в спецификации HTML значение атрибута **checked** прописывать не нужно. Его можно сократить, как в примере ниже:

```
<input type="radio"
name="foo" checked>
```

В нашем примере переключатель по умолчанию установлен в положение «менее 24» (см. примечание).

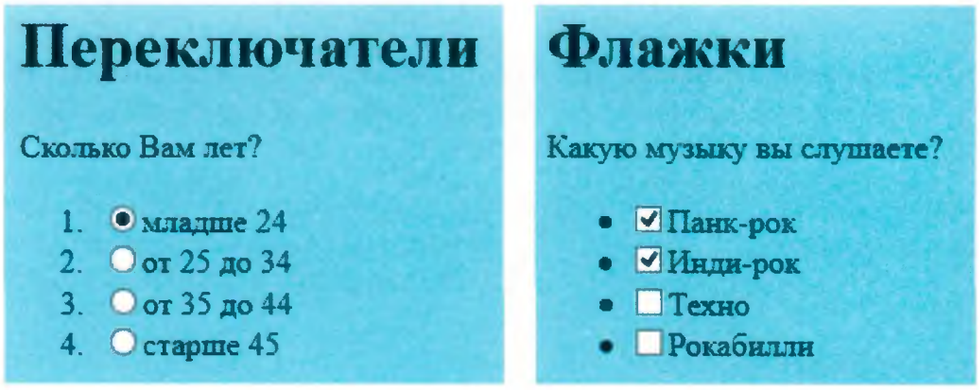


Рис. 9.11. Вставка переключателей на рисунке слева целесообразна, когда допустим выбор только одного значения из списка. Флажки (на рисунке справа) лучше подходят, если необходимо создать список, в котором одновременно может быть выбрано любое количество вариантов, в том числе ни один из них или все

Флажки

<input type="checkbox">

Флажок

Флажки размечают в форме с помощью элемента **input**, при этом атрибуту **type** следует присвоить значение **checkbox**. Как и в случае с переключателями, группа флажков создается путем назначения каждому одного и того же значения элемента **name**. Как отмечалось ранее, разница состоит в том, что одновременно можно устанавливать более одного флажка. При отправке данных формы на сервер будут переданы все значения элементов формы, напротив которых были установлены флажки. Приведем пример группы флажков, предназначенных для выбора музыкальных интересов. Рис. 9.11 иллюстрирует вид группы флажков при просмотре в браузере.

```
<p>Какую музыку вы слушаете?</p>

<ul>

<li><input type="checkbox" name="genre" value="punk"
checked="checked">Панк-рок</li>

<li><input type="checkbox" name="genre" value="indie"
checked="checked">Инди-рок</li>

<li><input type="checkbox" name="genre"
value="techno">Техно</li>

<li><input type="checkbox" name="genre"
value="rockabilly">Рокабилли</li>

</ul>
```

УПРАЖНЕНИЕ 9.2. ДОБАВЛЕНИЕ ПЕРЕКЛЮЧАТЕЛЕЙ И ФЛАЖКОВ

Для выбора ответов на следующие два вопроса формы заказа нужны переключатели и флажки. Откройте документ *contest_entry.html* и следуйте дальнейшим указаниям.

1. В разделе «Характеристики» представлены перечни вариантов цвета и дизайна. Выбор вариантов цвета должен осуществляться средствами переключателей, так как обувь может быть только одного цвета. Вставьте элемент переключателя напротив каждого. На основе данного кода напишите разметку для остальных вариантов цвета.

```
<li><label><input type="radio" name="color"
value="red"> Красный</label></li>
```

2. Составьте разметку для ответов на вопрос о дизайне обуви таким же образом, как и для вопроса о цвете. Однако на этот раз тип (атрибут **type**) элемента **input** установите в значение **checkbox**. Убедитесь, что имя переменной каждого флажка — **features[]** и что, согласно пометкам на схеме, флажок по умолчанию установлен напротив варианта «Металлическая бляшка».
3. Сохраните документ и проверьте правильность отображения формы в браузере, а затем нажмите кнопку «Заказать!» и проверьте ее работоспособность.

Конечно же, флажки не обязательно должны входить в состав группы. В этом примере, чтобы пользователи могли обозначить свое согласие на получение специальных предложений, требуется один-единственный флажок. Значение элемента формы будет передаваться на сервер только в том случае, если пользователь этот флажок установит.

```
<p><input type="checkbox" name="OptIn" value="yes">Да, присылайте мне на электронную почту новости и специальные предложения.</p>
```

В упражнении 9.2 у вас будет возможность научиться вставлять в форму переключатели и флажки.

Списки

Еще один способ представления набора вариантов для выбора — перечисление их в списке. Списки обычно выглядят компактнее, чем группы кнопок или флажков.

Как раскрывающиеся, так и прокручиваемые списки создаются с помощью элемента **select**. Какой именно вы увидите в форме, раскрывающийся или прокручиваемый, зависит от заданного размера элемента и того, разрешен ли одновременный выбор более одного пункта списка. Давайте рассмотрим оба варианта.

<select>...</select>

Список

<option>...</option>

Пункт списка

<optgroup>...</optgroup>

Логическая группа пунктов списка

Раскрывающиеся списки

Элемент **select** отображается как раскрывающийся список (также называемый раскрывающимся или выпадающим меню) по умолчанию, когда размер не задан или атрибуту **size** присвоено значение 1. В раскрывающемся списке возможен выбор только одного из пунктов. Приведем пример (визуализацию данного примера смотрите на рис. 9.12):

```
<p>Какая ваша любимая группа восьмидесятых?
<select name="EightiesFave">
<option>The Cure</option>
<option>Cocteau Twins</option>
<option>Tears for Fears</option>
<option>Thompson Twins</option>
<option value="EBTG">Everything But the Girl</option>
<option>Depeche Mode</option>
<option>The Smiths</option>
<option>New Order</option>
</select>
</p>
```

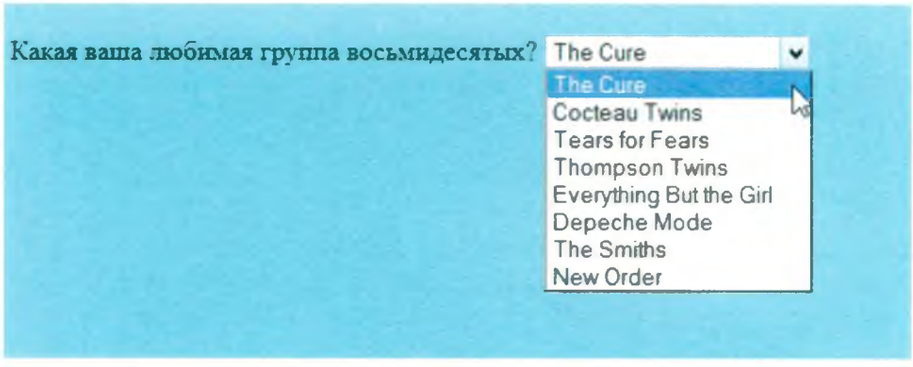


Рис. 9.12. Когда пользователь щелкает мышью по стрелке или по самому списку, тот раскрывается

Как видно из кода примера, элемент **select** представляет собой лишь контейнер для элементов **option**. При отправке данных формы на сервер передается содержимое выбранного элемента **option**. Если по какой-то причине нужно отправить значение, отличное от того, которое отображается в списке, следует заменить одно значение на другое, воспользовавшись атрибутом **value**. Например, если пользователь выберет пункт Everything But the Girl, форма отправит «EBTG» в качестве значения переменной EightiesFave. В остальных случаях в качестве значения будет отправлен текст, находящийся между тегов **option**.

Аналогичный раскрывающийся список для выбора размера обуви вы создадите в упражнении 9.3.

Прокручиваемые списки

Для отображения прокручиваемого списка укажите число видимых строк, введя соответствующее значение атрибута **size**. Этот код ничем не отличается от приведенного в предыдущем примере, за исключением того, что в данном случае формируется список высотой в шесть строк (рис. 9.13).

```
<p>Какие группы 80-х вы слушали?
<select name="EightiesBands" size="6" multiple>
<option>The Cure</option>
<option>Cocteau Twins</option>
<option selected="selected">Tears for Fears</option>
<option selected="selected">Thompson Twins</option>
<option value="EBTG">Everything But the Girl</option>
<option>Depeche Mode</option>
<option>The Smiths</option>
<option>New Order</option>
</select>
</p>
```

Вероятно, вы заметили, что здесь появилось несколько новых атрибутов. Атрибут **multiple** позволяет пользователям выбрать более одного элемента в списке. Обратите внимание, что раскрывающиеся списки не поддерживают атрибут **multiple**; когда браузер его встречает, он автоматически отображает прокручиваемый список.

Указав в элементе **option** атрибут **selected**, вы сделаете его значением по умолчанию для данного списка. При загрузке страницы такие элементы будут выделены. Атрибут **selected** можно указывать для обоих типов списков.

Рис. 9.13. Прокручиваемый список, в котором выбраны сразу несколько элементов

Группировка пунктов раскрывающегося списка

Элемент **optgroup** предназначен для создания смысловых групп пунктов списка. Обязательный атрибут **label**, включенный в элемент **optgroup**, содержит заголовок группы. На рис. 9.14 показано, как группы пунктов выглядят при отображении современными браузерами.

ПРИМЕЧАНИЕ

Следует различать атрибут **label**, включенный в элемент **optgroup**, и элемент **label**, предназначенный для обеспечения доступности (обсуждается далее в этой главе).

```
<select name="icecream" size="7" multiple>
<optgroup label="Традиционные">
<option>Ванильное</option>
<option>Шоколадное</option>
</optgroup>
<optgroup label="Вкусы дня">
<option>Супер пралине</option>
<option>Ореховый сюрприз</option>
<option>Леденцовое</option>
</optgroup>
</select>
```

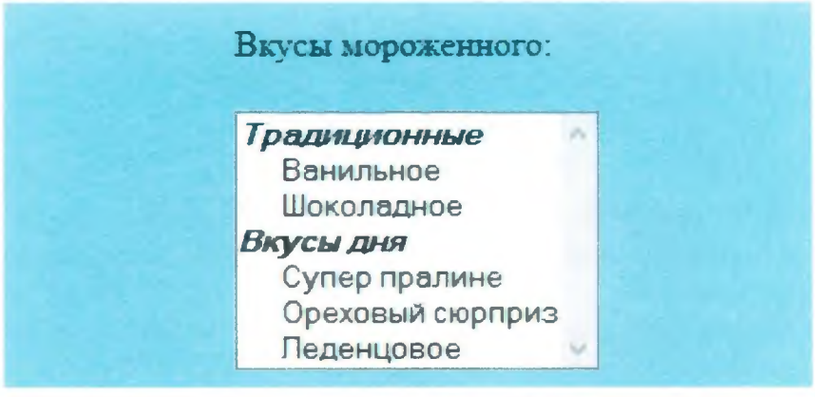


Рис. 9.14. Группа пунктов списка, визуализированная браузером

УПРАЖНЕНИЕ 9.3. СОЗДАНИЕ РАСКРЫВАЮЩЕГОСЯ СПИСКА

Нам осталось добавить на форму заказа всего один элемент — раскрывающийся список для выбора размера обуви.

1. Вставьте элемент **select** списка и задайте размеры обуви (с 33 по 43).

```
<legend>Размер</legend>
<p>Стандартные российские размеры:
<select name="size" size="1">
<option>33</option>
...сюда вставьте остальные размеры...
</select>
</p>
```

2. Сохраните документ и откройте его в браузере. Также нажмите кнопку «Заказать!» и проверьте работоспособность формы.

Поздравляю! Вы создали свою первую функционирующую веб-форму. В упражнении 9.4 мы создадим разметку, обеспечивающую большую доступность формы для вспомогательных устройств. Однако сначала нам предстоит рассмотреть еще несколько ее элементов.

Элемент выбора файла

```
<input type="file">
```

Поле выбора файла

Веб-формы позволяют отправлять на сервер не только данные. С их помощью можно также загружать файлы с компьютера пользователя. Например, типография может принимать через веб-форму графические макеты как приложение к заказам на изготовление визитных карточек. Редакция журнала посредством веб-формы может принимать фотографии на фотоконкурс.

С помощью элемента выбора файла пользователь может выбрать на компьютере документ, который затем будет отправлен на сервер вместе с остальными данными из полей формы. Добавить его можно с помощью хорошо знакомого нам элемента **input**, в котором атрибут **type** принимает значение **"file"**.

Приведенная ниже разметка и рис. 9.15 демонстрируют элемент выбора файла.

```
<form action="/client.php" method="POST" enctype="multipart/
form-data">
<label>Загрузите фотографию для аватара<em>(необязательно)</
em><br>
<input type="file" name="photo" size="28"></label>
</form>
```

В разных браузерах поле выбора файла может отображаться по-разному. Это может быть текстовое поле с кнопкой «Обзор» для выбора файла на жестком диске, как в браузере Internet Explorer (рис. 9.15, вверху) или кнопка с другим названием и сопровождающей надписью, найден ли файл, которые отображаются в браузере Chrome (рис. 9.15, внизу).

Важно отметить, что для формы, которая содержит в себе элемент выбора файла, в качестве типа кодирования (**enctype**) следует указывать значение **multipart/form-data** и отправлять данные методом POST. В этом примере атрибут **size** задает длину поля файла в символах (хотя ее также можно задавать с помощью правила CSS), если в браузере отображается поле.

Поле выбора файла (Internet Explorer)

Поле выбора файла (Chrome)

Рис. 9.15. Поле формы для выбора файла

Скрытые элементы формы

`<input type="hidden">`

Скрытый элемент формы

Бывает, что необходимо отправить программе-обработчику информацию, которая исходит не от пользователя. В этих случаях применяют скрытые элементы формы, которые передают на сервер необходимую информацию вместе с остальными данными в момент их отправки, но при этом невидимы при просмотре формы в браузере.

Чтобы скрыть элемент формы, следует атрибуту **type** элемента **input** присвоить значение **hidden**. Единственная его задача — передать пару имя/значение на сервер в момент отправки данных формы. В нашем примере скрытый элемент формы передает на сервер ссылку на файл, который отображается в браузере в ответ на отправку данных формы.

```
<input type="hidden" name="success-link" value="http://www.
example.com/littlechair_thankyou.html">
```

ПРЕДУПРЕЖДЕНИЕ

Пользователи могут получить доступ и изменить скрытые элементы формы. Если вы хотите стать профессиональным веб-разработчиком, вам нужно научиться устанавливать в программном коде защиту против подобных случаев.

Мне приходилось работать с формами, которые включали в себя десятки скрытых элементов помимо тех, с помощью которых пользователь мог вводить какие-либо данные. Это информация, которую сообщает разработчик приложения, системный администратор или программист формы на стороне сервера. Если вы используете готовый сценарий, внимательно изучите инструкции к нему на предмет необходимости ввода каких-либо скрытых переменных.

Элементы даты и времени (HTML5)

`<input type="date">`

Выбор даты

Новый в HTML5

`<input type="time">`

Выбор времени

Новый в HTML5

`<input type="datetime">`

Выбор даты/времени с учетом часового пояса

Новый в HTML5

`<input type="datetime-local">`

Выбор даты/времени без учета часового пояса

Новый в HTML5

`<input type="month">`

Выбор месяца года

Новый в HTML5

`<input type="week">`

Выбор определенной недели в году

Новый в HTML5

Если вы когда-либо бронировали номер в гостинице или авиабилеты во Всемирной паутине, то, несомненно, пользовались небольшим виджетом календаря для выбора даты. Скорее всего, маленький календарь был создан с помощью JavaScript. Язык HTML5 представляет шесть новых типов ввода данных, которые превращают виджеты для выбора даты и времени в одну из стандартных встроенных возможностей визуализации браузера (такую же, как способность отображать флажки, раскрывающиеся списки и другие современные элементы). Элементы выбора даты и времени на момент написания книги применялись только в нескольких браузерах, например в Opera и Chrome, причем частично (на момент написания книги элемент `<input type="datetime">` не поддерживался) (см. рис. 9.16). В браузерах, не поддерживающих эти элементы, поля ввода даты и времени отображаются в виде рабочего текстового поля.

```
<input type="date">

<input type="time">

<input type="datetime">

<input type="datetime-local">

<input type="month">

<input type="week">
```

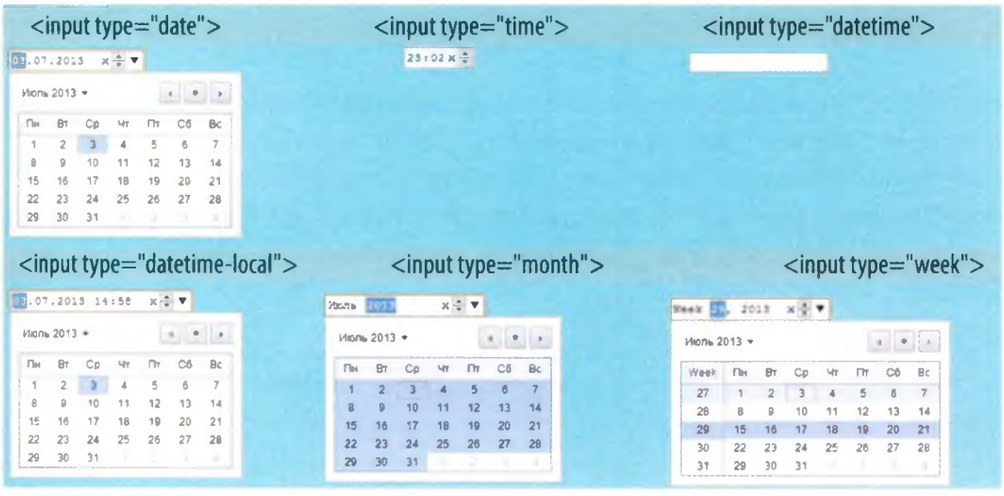


Рис. 9.16. Элементы выбора даты и времени в браузере Opera

Новые элементы ввода, связанные с указанием даты и времени, следующие:

```
<input type="date" name="имя" value="2013-01-14">
```

Создает элемент ввода даты, такой как всплывающий календарь, для указания даты (год, месяц, день). Начальное значение должно быть представлено в формате международной организации по стандартизации, International Organization for Standardization, ISO (**ГГГГ-ММ-ДД**).

```
<input type="time" name="имя" value="03:13:00">
```

Создает элемент ввода для указания времени (часы, минуты, секунды, доли секунды) без часового пояса. Значение указывается следующим образом: **чч:мм:сс**.

```
<input type="datetime" name="имя" value="2004-01-14T03:13:00-5:00">
```

Создает элемент ввода сочетания даты и времени, который содержит информацию о часовом поясе. Значение представляет собой дату и время в формате ISO с часового пояса относительно Универсального астрономического времени, как и для элемента **time** в главе 5 (**ГГГГ-ММ-ДДТчч:мм:ссTZD**). На момент написания книги ни в одном из популярных браузеров не поддерживался.

```
<input type="datetime-local" name="имя" value="2004-01-14T03:13:00">
```

Создает элемент ввода сочетания даты и времени без учета часового пояса

(**ГГГГ-ММ-ДДТчч:мм:сс**)

```
<input type="month" name="имя" value="2004-01">
```

Создает элемент ввода определенного месяца года (**ГГГГ-ММ**).

```
<input type="week" name="имя" value="2004-W2">
```

Создает элемент ввода даты для обозначения определенной недели в году с использованием нумерации формата ISO (**ГГГГ-W#**).

Ввод чисел (HTML5)

`<input type="number">`

Ввод чисел

Новый в HTML5

`<input type="range">`

Ползунковый регулятор

Новый в HTML5

Числовые данные можно вводить с помощью типов ввода **number** и **range**. Для типа ввода **number** браузер может предоставить виджет счетчика, чтобы дать возможность выбрать конкретное числовое значение (браузеры, не поддерживающие этот тип ввода, будут отображать текстовое поле). Тип ввода **range**, как правило, отображается в виде ползункового регулятора (рис. 9.17), который позволяет пользователю выбрать значение в пределах указанного диапазона.

```
<label>Количество гостей<input type="number" name="guests"
min="1" max="6"></label>

<label>Удовлетворенность (0 до 10)<input type="range"
name="satis" min="0" max="10" step="1"></label>
```

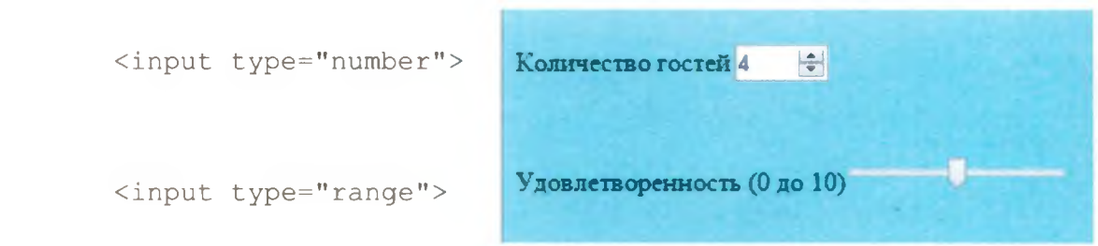


Рис. 9.17. Типы ввода **number** *и* **range** *в браузере Opera*

Обоим типам ввода **number** и **range** присущи атрибуты **min** и **max** для указания минимальных и максимальных допустимых значений ввода (опять же, браузер может проверить, что введенные пользователем значения находятся в пределах допустимых).

Атрибуты **min** и **max** необязательны, и вы можете также установить один из них, опустив другой.

Атрибут **step** позволяет разработчикам указывать приемлемые приращения числового ввода. По умолчанию установлено значение 1. Если установить его равным .5, это позволит использовать значения 1, 1,5, 2, 2,5 и так далее; если 100 — 100, 200, 300, и так далее. Кроме того, можно установить значение **any** атрибута **step**, указав, что в качестве приращения принимаются любые величины.

Опять же, браузеры, которые не поддерживают эти новые типы ввода, отображают вместо них обычное текстовое поле — прекрасный запасной вариант.

Выбор цвета (HTML5)

`<input type="color">`

Палитра цветов

Новый в HTML5

Цель элемента выбора цвета — создание появляющейся палитры цветов, похожей на те, которые используются в графических редакторах для визуального выбора значения цвета. Значения приведены в шестнадцатеричном формате RGB (# RRGGBB). На рис. 9.18 показан результат щелчка мышью по элементу выбора цвета в браузере Chrome.

Еще несколько элементов формы в спецификации HTML5

Для полноты картины рассмотрим остальные элементы формы, появившиеся в HTML5. На момент написания книги, они поддерживались не всеми браузерами и в любом случае были не настолько распространены, что можно немного подождать, прежде чем добавлять их в свои HTML-формы. Вы уже знаете об элементе **datalist**, используемом для задания предполагаемых значений вводимого текста. В HTML5 также вводятся следующие элементы:

progress

<progress> ... </progress >

Обозначение состояния текущего процесса

Новый в HTML5

Элемент **progress** предоставляет пользователям обратную связь, сообщая о состоянии текущих процессов, таких как загрузка файла. У него может быть конкретное конечное значение (обозначаемое с помощью атрибута **max**) или просто указание на то, что процесс выполняется (например, ожидание ответа сервера).

```
Процент загрузки: <progress max="100" name="fave"> 0</progress>
```

meter

<meter> ... </ meter >

Обозначение состояния текущего процесса

Новый в HTML5

Похож на элемент **progress**, но он всегда представляет собой измерение в известном диапазоне значений (также известен как *индикатор*). Ему присущ ряд атрибутов: **min** и **max** указывают минимальное и максимальное значения диапазона; **low** и **high** могут быть использованы для предупреждения о нежелательных пороговых уровнях, а также **optimum**, который указывает на предпочтительное значение.

Значения, вероятно, будут обновляться динамически во время процесса с помощью JavaScript.

```
<meter min="0" max="100" name="download">50%</meter >
```

output

<output> ... </output>

Расчетное значение вывода

Новый в HTML5

Проще говоря, элемент **output** позволяет вывести результаты расчетов с помощью сценария или программы и связать их с вводимыми данными, которые влияют на расчет.

keygen

<keygen>

Генератор ключевых пар

Новый в HTML5

Элемент **keygen** представляет собой элемент формы для создания пары ключей (используется для обеспечения сохранности персональной информации). При отправке формы секретный ключ хранится локально, а публичный упаковывается и отправляется на сервер. Вы можете прочитать об этом подробнее на странице **ru.wikipedia.org/wiki/Криптосистема_с_открытым_ключом**.

Браузеры, не поддерживающие этот тип ввода, по умолчанию отображают текстовое поле.

```
<label>Ваш любимый цвет< input type="color" name="favorite"></
label>
```

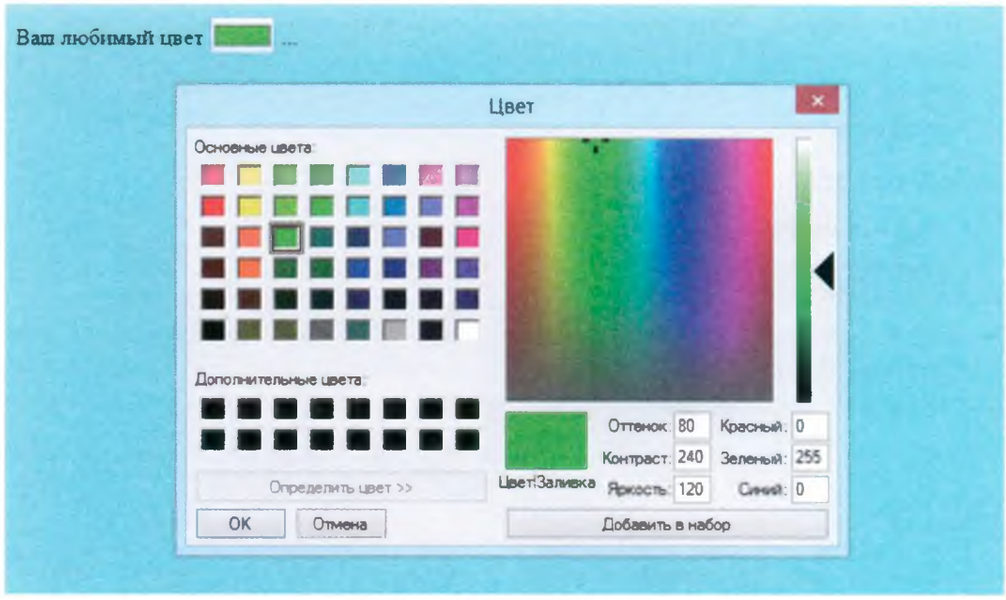


Рис. 9.18. Тип ввода ***color*** *(в браузере Chrome)*

На этом сводка элементов формы завершается. Их вставка — это только первый шаг к созданию формы; любой компетентный веб-разработчик не пожалеет времени, чтобы убедиться, что форма максимально доступна. К счастью, мы можем кое-что сделать в разметке для описания структуры формы.

Обеспечение доступности форм

Чрезвычайно важно продумать возможности восприятия и навигации по формам на вашем сайте для посетителей, которые не могут в полной мере воспользоваться функциями визуальных браузеров. Элементы формы **label**, **fieldset** и **legend** повышают доступность, создавая четкие семантические связи между компонентами формы. Итоговая разметка становится не только многофункциональнее в семантическом смысле, но в ней также появляется больше элементов, которые можно задействовать в качестве «якорей», для привязки правил CSS. Выигрывают все!

Элемент label

Несмотря на то что в браузере надпись «Адрес» визуально находится рядом с текстовым полем, предназначенным для ввода адреса, может оказаться, что в исходном коде надпись и поле будут отделены друг от друга, например ячейками таблицы. Элемент **label** необходим для сопоставления текстового описания с соответствующим полем формы. Таким образом пользователи браузеров, основанных на чтении с экрана, получают важную контекстную информацию.

Каждый элемент **label** может быть соотнесен ровно с одним элементом формы. Работать с ним можно в двух направлениях. Первый метод, называемый ***неявной ассоциацией***, внедряет элемент формы вместе с его описанием в элемент **label**. В приведенном ниже примере элементы **label** присвоены отдельным флажкам и относящимся к ним текстовым описаниям. (Кстати, именно так помечают переключатели и флажки. Элемент **label** нельзя присвоить целой группе).

```
<ul>

<li><label><input type="checkbox" name="genre"
value="punk">Панк-рок</label></li>

<li><label><input type="checkbox" name="genre"
value="indie">Инди-рок</label></li>

<li><label><input type="checkbox" name="genre"
value="hiphop">Хип-хоп</label></li>

<li><label><input type="checkbox" name="genre"
value="rockabilly">Рокабилли</label></li>

</ul>
```

Другой метод, называемый ***явной ассоциацией***, основан на сопоставлении метки с идентификатором (**id**) элемента формы. Атрибут **for** указывает, к какому элементу формы относится метка. Такой подход может оказаться полезным в случае, если в исходном коде страницы элемент формы отдален от текста своего описания. Кроме того, данный метод обладает потенциальным преимуществом представления метки и элемента формы в виде двух самостоятельных элементов, что может пригодиться при их выравнивании средствами таблиц стилей.

```
<label for="form-login-username">Имя пользователя:</label>

<input type="text" name="login" id="form-login-username">

<label for="form-login-password">Пароль:</label>

<input type="password" name="password" id="form-login-
password">
```

Еще одно преимущество использования элементов **label** заключается в том, что пользователь может щелкнуть мышью или коснуться элемента в любой позиции, чтобы выбрать его. Пользователи устройств с сенсорными экранами будут признательны, если область касания окажется более крупной.

ПРЕДУПРЕЖДЕНИЕ

На момент написания книги в устройствах с операционной системой iOS скрытые элементы **label** не откликались на касания, поэтому нужного поведения приходилось добиваться с помощью JavaScript. Я знаю, что мы еще не рассматривали JavaScript, но если вам интересно, исправление выглядит так:

```
document.
getElementsByTagName

('label').setAttribute

('onclick','');
```

СОВЕТ

Чтобы отделить идентификаторы **id**, имеющие отношение к форме, от остальных идентификаторов на странице, заведите себе правило снабжать их приставкой **form-**, как показано в данном примере.

Еще один способ организации элементов формы заключается в следующем: присвойте форме идентификатор **id**, и используйте его как часть идентификаторов всех элементов данной формы. Например:

```
<form id="form-login">
<input id="form-login-username">
<input id="form-login-password">
```

Элементы fieldset и legend

Элемент **fieldset** обозначает логическую группу элементов формы. Элемент **fieldset** может также содержать элемент **legend** с условными обозначениями элементов формы, составляющих логическую группу.

На рис. 9.19 представлена визуализация приведенного ниже кода в браузере Firefox, отображаемая по умолчанию, но внешний вид элементов **fieldset** и **legend** можно настроить с помощью таблиц стилей:

```
<fieldset>
<legend>Подписка на рассылку</legend>
<ul>
<li><label>Включите меня в список рассылки<input type="radio" name="list" value="yes" checked="checked"></label></li>
<li><label>Спасибо, не надо <input type="radio" name="list" value="no"> </label></li>
</ul>
</fieldset>
<fieldset>
<legend>Информация о покупателе</legend>
<ol>
<li><label>Полное имя: <input type="text" name="name"></label></li>
<li><label>Email: <input type="text" name="email"></label></li>
<li><label>Город: <input type="text" name="state"></label></li>
</ol>
</fieldset>
```

ПРЕДУПРЕЖДЕНИЕ

При форматировании элементов **fieldset** и **legend** с помощью стилей, эти элементы могут вести себя непредсказуемо. Например, цвет фона элемента **fieldset** отображается поразному в разных браузерах. Текст элементов **legend** ни в одном браузере не переносится на новую строку. Решение заключается в добавлении к ним элемента **span** или **b** и управлении представлением заключенного в них элемента без снижения доступности. Если вы добавляете к этим элементам стили, обязательно проведите несколько тестов.

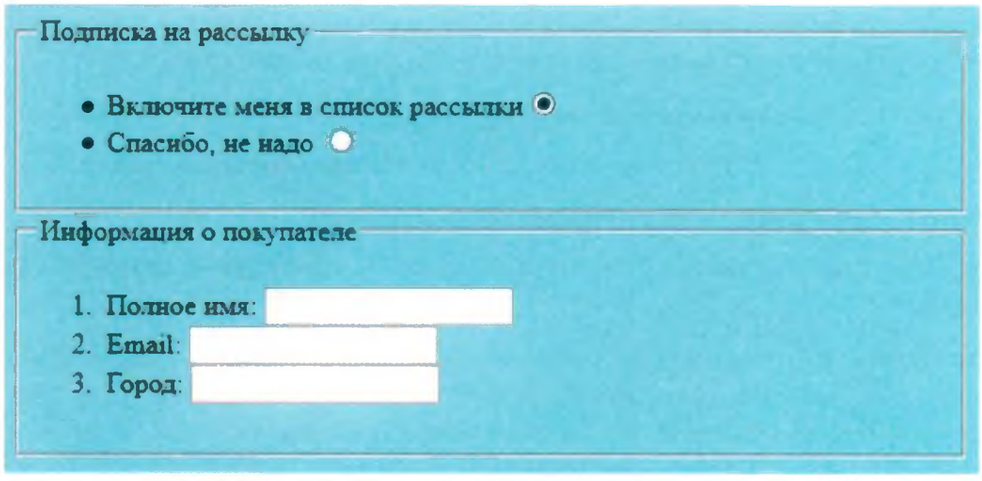


*Рис. 9.19. Визуализация элементов **fieldset** и **legend** по умолчанию*

УПРАЖНЕНИЕ 9.4. ЭЛЕМЕНТЫ LABEL И FIELDSET

Наша форма заказа работает, но, чтобы сделать ее более удобной для использования на вспомогательных устройствах, необходимо добавить к ней соответствующие элементы **label** и создать несколько элементов **fieldset**. Вновь откройте файл *contest_entry.html* и выполните следующие действия.

Я хотела бы начать с основных фрагментов, а детали добавить позже, поэтому мы начнем это упражнение с организации элементов формы в группы с помощью элемента **fieldset**, а затем добавим все метки. Можно сделать и наоборот. В идеале вы просто размечаете элементы **label** и **fieldset** по ходу работы, а не добавляете их позже.

1. Текстовые поля в разделе «Сведения о заказчике» в верхней части формы, безусловно, концептуально связаны между собой, так что давайте заключим их все в элемент **fieldset**. Поменяйте разметку заголовка раздела с абзаца (**p**) на **fieldset**.

```
<fieldset>
<legend>Сведения о заказчике</legend>
<ul>
<li>Имя:<input type="text"
name="username"></li>
...
</ul>
</fieldset>
```

2. Далее, поместим вопросы, касающиеся цвета, характеристик и размеров в группу элементов с условным обозначением «Характеристики» (текст уже есть, просто нужно заменить элемент **p** на **legend**).

```
<h2>Выберите дизайн сандалий:</h2>
<fieldset>
<legend>Характеристики</legend>
Цвет...
Дополнительно...
Размер...
</fieldset>
```

3. Создайте еще одну группу элементов формы только для вариантов цвета, снова заменив элемент **p** на **legend**. Проделайте то же самое в разделах «Дополнительно» и «Размер». В конце концов, у вас будет три группы элементов формы, заключенных в одну более крупную группу «Характеристики». Когда закончите, сохраните документ и откройте его в браузере. Он должен быть похож на окончательный вариант формы, показанный на рис. 9.2 с учетом ожидаемых различий отображения в браузерах.

```
<fieldset>
<legend>Цвет <em>(выберите один пункт)</
em>:</legend>
<ul>...</ul>
</fieldset>
```

4. Теперь добавим несколько меток. К набору полей «Сведения о заказчике» добавьте метку текстового поля, используя метод явной ассоциации. Первую я уже добавила, а вы сделайте остальные три.

```
<li><label for="form-name">Имя:</label>
<input
type="text" name="username" id="form-
name"></li>
```

5. Для создания переключателей и флажков заключите элемент **input** в элемент **label**. Таким образом, кнопка будет выбрана, когда пользователь щелкнет мышью или коснется любой позиции внутри элемента **label**. Ниже представлен образец, сделайте остальные семь.

```
<li><label><input type="radio"
name="color"
value="red">Красный</label></li>
```

Сохраните документ, и все готово! Метки не должны влиять на то, как форма отображается по умолчанию, но вам будет приятно, что вы добавили смысловое значение, и возможно, вы даже используете их для применения стилей в следующий раз.

Макет и дизайн формы

Несмотря на то что эта глава посвящена разметке, а не визуальному представлению, не могу завершить ее главу, не сказав несколько слов о дизайне форм.

Формы и удобство использования

Неудачно спроектированная форма может испортить впечатление пользователя от вашего сайта и отрицательно повлиять на цели вашего бизнеса. Плохо спроектированная форма — это потерянные клиенты, поэтому так важно сделать ее правильно, как для настольных компьютеров и ноутбуков, так и для устройств с небольшими экранами и особыми требованиями. Вам нужно, чтобы путь к покупке или иным действиям был как можно более гладким.

Тема дизайна хороших веб-форм широка, ее хватит на отдельную книгу. Фактически такая книга, написанная экспертом в области веб-форм, Люком Вроблевски, уже существует: «Web Form Design» (Rosenfeld Media, 2008),.

Следующая книга Люка — «Сначала мобильные!» (Манн, Иванов и Фербер, 2012) — содержит советы по форматированию форм для мобильной среды. На его сайте можно просмотреть более ста статей о веб-формах: **www.lukew.com/ff?tag=forms**.

Ниже я приведу вам несколько советов из книги «Web Form Design».

Избегайте ненужных вопросов

Помогите своим пользователям справиться с вашей формой как можно быстрее — не добавляйте в нее вопросы, которые не являются абсолютно необходимыми для выполнения поставленной задачи. Дополнительные вопросы не только замедляют процесс, но и могут вызвать у пользователя подозрения относительно мотивов, по которым вы их задаете. Если у вас есть другой способ получения информации (например, тип банковской карты можно определить по ее первым четырем цифрам), используйте альтернативные средства и не нагружайте пользователя. Если какую-то информацию вы хотели бы знать, но она необязательна, стоит подумать о том, чтобы спросить ее позже, после отправки формы и налаживания отношений с пользователем.

Учитывайте влияние позиций меток

Положение метки относительно элемента формы влияет на время, затрачиваемое на заполнение формы. Чем меньше взгляду пользователя приходится путешествовать по странице, тем быстрее происходит заполнение формы. Размещение меток над соответствующими им элементами создает благоприятную ситуацию для быстрого просмотра и заполнения формы, особенно когда запрашивается знакомая информация (имя пользователя, адрес, и т. д.). Метки, расположенные над элементами, могут быть различной длины и лучше других подходят для отображения форм на устройствах с узким маленьким экраном (например, смартфонах). Однако они приводят к удлинению формы, поэтому, если вас беспокоит занимаемое по вертикали пространство, можно расположить метки слева от поля ввода. Такое расположение меток приведет к более медленному заполнению формы, но, может оказаться целесообразным, если вы хотите, чтобы пользователь не спешил или

имел возможность просмотреть и изучить, какого типа информация запрашивается в форме.

Тщательно выбирайте типы элементов формы

Как вы видели в этой главе, существует довольно много типов элементов формы, из которых можно выбирать, и иногда не так просто решить, какой из них использовать. Например, список вариантов может быть представлен в виде раскрывающегося списка или группы флажков. Тщательно обдумайте плюсы и минусы каждого типа элементов формы, а затем проведите тестирование с участием пользователей.

Группируйте связанные элементы формы

Множество полей, списков и кнопок формы проще проанализировать, если они визуально сгруппированы по темам. Например контактные данные пользователя могут быть представлены в компактной группе так, что пять или шесть полей ввода будут восприниматься как одно целое. Как правило, вам нужен только ненавязчивый указатель, например тонкая горизонтальная линия и немного дополнительного пространства. Не переусердствуйте.

Уточните первостепенные и второстепенные действия

Основное действие в конце формы, как правило, представлено в виде кнопки «Отправить» того или иного вида («Купить», «Зарегистрироваться» и т. д.), которая оповещает о завершении заполнения формы и готовности двигаться дальше. Нужно, чтобы эта кнопка визуально доминировала, и ее было легко найти (например, помогает выравнивание ее по основной оси формы). Второстепенные действия, как правило, это шаг назад, например, сброс формы. Если вам необходимо добавить второстепенное действие, убедитесь, что оно выглядит иначе и менее важно, чем первостепенное действие. Кроме того, рекомендуется предоставить возможность отмены действия.

Стилизация форм

Как вы видели в этой главе, отображение разметки формы, установленное по умолчанию, не идет ни в какое сравнение с тем качеством, какое мы видим сегодня у большинства современных форм. С помощью каскадных таблиц стилей можно создать четкий макет формы, а также изменить внешний вид большинства ее элементов, как и в случае с любыми другими элементами. Такие простые особенности, как хорошее выравнивание и внешний вид, согласующийся с остальным сайтом, могут существенно улучшить впечатление, производимое на пользователя.

Помните, что виджеты форм создаются браузерами и задаются принятыми в операционной системе соглашениями. Однако к таким элементам формы, как текстовые поля, списки, группы элементов формы, метки и условные обозначения, можно применять различное форматирование — изменять размеры, поля, шрифты, цвета, границы и фон.

Только обязательно протестируйте формы в нескольких браузерах на случай появления неприятных сюрпризов. В главе 18 и в части III перечислены некоторые специфические методы, но вы должны приобрести больше опыта работы с CSS. Введите запрос «CSS формы» в поисковой системе и вы получите список рекомендаций в качестве дополнительной помощи.

Резюме

В этой главе мы рассмотрели множество элементов и атрибутов форм. Элементы с пометкой HTML5 являются новыми в спецификации HTML5.

Элемент и его атрибуты	Описание
button name="*текст*" value="*текст*" type="submit|reset|button"	Многофункциональная кнопка Содержит уникальное имя переменной элемента формы Содержит значение, которое будет отправлено на сервер Тип кнопки
datalist HTML5	Предоставляет список вариантов
fieldset	Группирует элементы формы и метки
form action="*url-адрес*" method="get|post" enctype="*тип контента*"	Форма Расположение программы-обработчика данных формы (обязательный атрибут) Метод отправки данных формы Метод кодирования, обычно или **application/x-www-form-urlencoded** (установлен по умолчанию) или **multipart/form-data**
input autofocus type="submit|reset|button|text|password|checkbox|radio|image|file|hidden|email|tel|search|url|date|time|datetime|datetime-local|month|week|number|range|color"	Создает различные элементы формы в зависимости от значения атрибута **type** Определяет, какой элемент формы должен быть готов к вводу при загрузке документа Тип элемента **input**

Элемент и его атрибуты	Описание

`disabled`
 	
`form="значение id формы"`
 	
Полный список атрибутов, относящихся к каждому типу ввода см. в табл.9.1
	Отключает элемент формы, делая его выбор невозможным
	Связывает элемент с определенной формой

`keygen` HTML5
 	
`autofocus`
 	
`challenge="строка шифра"`
 	
`disabled`
 	
`form="значение id формы"`
 	
`keytype="ключевое слово"`
 	
`name="текст"`
	Создает пару ключей для сертификата безопасной транзакции
	Обозначает, что элемент формы должен быть выделен и готов к вводу при загрузке документа
	Создает строку шифра, передаваемую вместе с ключом
	Отключает элемент формы, делая его выбор невозможным
	Связывает элемент с определенной формой
	Определяет тип генерируемого ключа (например, **`rsa`** или **`ec`**)
	Присваивает элементу формы имя для идентификации

`label`
 	
`for="текст"`
 	
`form="значение id формы"`
	Сопровождает элементы формы информацией о них
	Сопоставляет элемент формы по **`id`**-ссылке
	Связывает элемент с определенной формой

`legend`	Указывает подпись для группы элементов формы

`meter` HTML5
 	
`form="значение id формы"`
 	
`high="число"`
 	
`low="число"`
 	
`max="число"`
 	
`min="число"`
 	
`optimum="число"`
 	
`value="число"`
	Указывает дробное значение в заданном диапазоне
	Связывает элемент с определенной формой
	Указывает значение, которое считается «высоким» для данной шкалы
	Указывает значение, которое считается «низким» для данной шкалы
	Определяет максимальное значение диапазона
	Определяет минимальное значение диапазона
	Указывает значение, которое считается «оптимальным»
	Определяет фактическое или измеряемое значение

Элемент и его атрибуты	Описание

`optgroup`
`label="`*текст*`"`
 	
`disabled="disabled"`
	Определяет группу вариантов
	Содержит надпись для группы вариантов
	Отключает элемент формы, делая его выбор невозможным

`option`
`disabled`
 	
`label="`*текст*`"`
 	
`selected`
 	
`value="`*текст*`"`
	Пункт списка
	Отключает элемент формы, делая его выбор невозможным
	Содержит замещающий текст надписи для пункта списка
	Значение данного пункта списка устанавливается в качестве значения по умолчанию
	Содержит замещающее значение надписи для пункта списка

`output` HTML5
 	
`for="`*текст*`"`
 	
`form="`*значение id формы*`"`
 	
`name="`*текст*`"`
	Представляет результаты вычисления
	Создает связь между выводом и элементом формы
	Связывает элемент с определенной формой
	Задает элементу формы уникальное имя

`progress` HTML5
 	
`form="`*значение id формы*`"`
 	
`max="`*число*`"`
 	
`value="`*число*`"`
	Обозначает прогресс выполнения задачи (может применяться, даже если максимальное значение задачи не известно)
	Связывает элемент с определенной формой
	Указывает общее значение или конечный размер задачи
	Указывает, насколько выполнена задача

`select`
 	
`autofocus`
 	
`disabled`
 	
`form="`*значение id формы*`"`
 	
`multiple="multiple"`
	Раскрывающийся или прокручиваемый список
	Обозначает, что элемент формы должен быть выделен и готов к вводу при загрузке документа
	Отключает элемент формы, делая его выбор невозможным
	Связывает элемент с определенной формой
	Включает возможность выбора из прокручиваемого списка нескольких вариантов

Элемент и его атрибуты	Описание
`name="`*`текст`*`"`	Содержит уникальное имя переменной элемента формы
`readonly`	Запрещает изменение элемента формы пользователем
`required`	Указывает, что поле элемента формы обязательно к заполнению пользователем
`size="`*`число`*`"`	Высота прокручиваемого списка в строках
`textarea`	Многострочное текстовое поле
`autofocus`	Обозначает, что элемент формы должен быть выделен и готов к вводу при загрузке документа
`cols="`*`число`*`"`	Ширина области текста в символах
`dirname="`*`текст`*`"`	Позволяет указать направление текста
`disabled`	Отключает элемент формы, делая его выбор невозможным
`form="`*`значение id формы`*`"`	Связывает элемент с определенной формой
`maxlength="`*`текст`*`"`	Указывает максимальное количество символов, допустимое для ввода пользователем
`name="`*`текст`*`"`	Содержит уникальное имя переменной элемента формы
`placeholder="`*`текст`*`"`	Предоставляет краткую подсказку, помогающую пользователю ввести верные данные
`readonly`	Запрещает изменение элемента формы пользователем
`required`	Указывает, что поле элемента формы обязательно к заполнению пользователем
`rows="`*`число`*`"`	Высота области текста в строках
`wrap="hard|soft"`	Контролирует, сохраняются ли переносы строк, созданные при вводе текста в данных. При значении **`hard`** переносы строк сохраняются, а при значении **`soft`** — нет

Табл. 9.1. Доступные атрибуты для каждого типа ввода

	submit	reset	button	text	password	checkbox	radio	image	file	hidden
accept									•	
alt								•		
checked						•	•			
disabled	•	•	•	•	•	•	•	•	•	•
maxlength				•	•				•	
name	•	•	•	•	•	•	•	•	•	•
readonly				•	•	•	•		•	
size				•	•				•	
src								•		
value	•	•	•	•	•	•	•		•	•
Только HTML5										
autocomplete				•	•					
autofocus	•	•	•	•	•	•	•	•	•	
form	•	•	•	•	•	•	•	•	•	•
formaction	•							•		
formenctype	•							•		
formmethod	•							•		
formnovalidate	•							•		
formtarget	•							•		
height								•		
list				•						
max										
min										
multiple										•
pattern				•	•					
placeholder				•	•					
required				•	•	•	•		•	
step										
width								•		

	email	telephone, search, url	number	range	date, time, datetime, datetime-local, month, week	color
accept						
alt						
checked						
disabled	•	•	•	•	•	•
maxlength	•	•				
name	•	•	•	•	•	•
readonly	•	•	•		•	
size	•	•				
src						
value	•	•	•	•	•	•
Только HTML5						
autocomplete	•	•	•	•	•	•
autofocus	•	•	•	•	•	•
form	•	•	•	•	•	•
formaction						
formenctype						
formmethod						
formnovalidate						
formtarget						
height						
list	•	•	•	•	•	•
max			•	•	•	
min			•	•	•	
multiple	•					
pattern	•	•				
placeholder	•	•				
required	•	•	•		•	
step			•	•	•	
width						

ГЛАВА 10

ЗНАКОМСТВО С HTML5

Мы использовали элементы HTML5 в нескольких последних главах, но эта спецификация — не только новые возможности разметки. Фактически HTML5 — это набор новых методов, решающих задачи, для выполнения которых раньше требовались специальное программирование или проприетарные плагины, такие как Flash или Silverlight. HTML5 предлагает стандартизированный способ размещения на странице видео, аудио и интерактивных элементов с использованием открытого исходного кода, а также возможность хранить данные локально, работать в автономном режиме, пользоваться преимуществами информации о географическом местоположении и многое другое. Применяя HTML5 для решения распространенных задач, разработчики могут полагаться на встроенные возможности браузера, и им не нужно «изобретать колесо» для каждого приложения.

HTML5 предлагает так много перспективных возможностей, что этот термин нередко используют и за пределами спецификации.

Когда маркетологи и журналисты говорят «HTML5», они иногда имеют в виду техники CSS3 или любые новые веб-технологии без использования Flash. В этой главе вы узнаете, что на самом деле входит в спецификацию, и, возможно станете одним из тех, кого, как и нас, раздражает, когда термин HTML5 употребляется неверно. Важно, однако, то, что осведомленность пользователей о веб-стандартах в целом, безусловно, является преимуществом и облегчает нам работу при общении с клиентами.

Конечно, любая спецификация в стадии разработки неодинаково поддерживается разными браузерами. Некоторые свойства можно использовать прямо сейчас, а другие — пока еще нет. Но на этот раз, вместо того чтобы ждать, пока спецификация будет полностью готова, браузеры постепенно внедряют поддержку все новых и новых свойств, а разработчикам рекомендуется начинать их использовать (см. врезку «Отслеживание поддержки браузеров»). Также следует упомянуть, что спецификация HTML5 быстро развивается, и к тому времени, как вы будете читать эту книгу, отдельные элементы, вероятно, изменятся. Я сделаю все возможное, чтобы предоставить исчерпывающий обзор, а вы сами решите, какие свойства изучать и использовать.

Многие из нововведений спецификации HTML5 требуют профессиональных навыков веб-разработки. Но независимо от того, будете ли вы применять их, ознакомиться в общих чертах с с предлагаемыми возможностями весьма полезно.

В этой главе

- Что такое HTML5?
- Краткая история стандарта HTML
- Новые элементы и атрибуты HTML5
- Добавление аудио- и видеоконтента на страницы
- Элемент `canvas`

Отслеживание поддержки браузеров

Существует несколько замечательных ресурсов, которые помогут вам узнать, какие из функций HTML5 готовы к использованию. Большинство также отображают поддержку свойств CSS и селекторов.

- Когда я смогу использовать ... (**caniuse.com**)
- Пожалуйста, HTML5 (**html5please.com**)
- Тест HTML5 (**html5test.com**)
- Поддержка HTML5 & CSS3 (**fmbip.com/litmus/**)

Именно «знакомство в общих чертах» является целью данной главы. Для дополнительного углубленного изучения функций HTML5 я рекомендую следующие книги:

- «Самоучитель HTML5» Билла Сандерса (Эксмо, 2012)
- «HTML5 и CSS3 для всех» Алексиса Голдстайна, Луиса Лазариса и Эстель Уэйл (Эксмо, 2012)

Краткая история HTML

История HTML, от первоначального проекта Тима Бернерс-Ли в 1991 году до стандарта HTML5, находящегося сегодня в стадии разработки, увлекательна и насыщена событиями.

Ранние версии HTML

Ранние версии HTML (HTML+ в 1994 году и HTML 2.0 в 1995) были разработаны на основе ранних работ Тима с целью создания жизнеспособной системы управления информацией. Однако когда Всемирная паутина завоевала мир, разработчики браузеров, в первую очередь таких, как Mosaic Netscape и Microsoft Internet Explorer, не стали ждать общих стандартов. Они дали людям то, что те хотели, создав множество элементов, улучшающих внешний вид страниц, но индивидуальных для каждого браузера. Это конкурентное противостояние получило название «Войны браузеров». В результате в конце 1990-х стало обычным делом создавать несколько разных версий сайта — по одной для каждого из популярных браузеров.

В 1996 году только что образованный консорциум Всемирной паутины (W3C) задал ориентир и выпустил первую Рекомендацию — HTML 3.2. Это собрание всех HTML-элементов, использовавшихся в то время. В него вошло множество презентационных расширений HTML, появившихся в результате соперничества браузеров, а также из-за отсутствия альтернативы в виде таблиц стилей. HTML 4.0 (1998) и HTML 4.01 (редакция с небольшими изменениями, которая заменила предшествующий стандарт в 1999) должны были вернуть HTML в нужное русло, подчеркнув разделение структуры и представления, и повысив доступность информации для пользователей с ограниченными возможностями.

Все задачи представления были переложены на новоиспеченные каскадные таблицы стилей (CSS), получавшие полную поддержку браузеров.

ПРИМЕЧАНИЕ

Чтобы узнать подробнее о возникновении Всемирной паутины и HTML, прочтите вторую главу книги «Raggett on HTML4» (Addison-Wesley, 1998) Дэвида Рэггетта, которую можно найти на сайте консорциума Всемирной паутины (**www.w3.org/People/Raggett/book4/ch02.html**).

Появление XHTML

Примерно в то же время, когда разрабатывалась версия HTML 4.01, сотрудники консорциума Всемирной паутины осознали, что один язык разметки с ограниченными возможностями не получится использовать

для описания всех видов информации (химических формул, математических уравнений, мультимедийных презентаций, финансовой информации и т. д.), которые можно распространять во Всемирной паутине. Их решение — *XML* (*Расширяемый язык разметки*), метаязык для создания языков разметки. XML — это упрощенный вариант *SGML* (*стандартного обобщенного языка разметки*), главного метаязыка, который Тим Бернерс-Ли использовал для создания своего оригинального HTML-приложения. Но сам SGML оказался сложнее, чем требовалось для Всемирной паутины.

В консорциуме W3C предполагали, что Всемирная паутина будет развиваться на основе XML, и множество специализированных языков разметки будут использоваться совместно даже в пределах одного документа. Конечно, чтобы претворить это в жизнь, пришлось бы очень внимательно создавать разметку, строго соблюдая синтаксис XML, чтобы исключить потенциальную путаницу.

Их первым шагом было переписать спецификацию HTML в соответствии с правилами XML, чтобы его можно было использовать вместе с другими XML-языками. В результате появился *XHTML* (*Расширяемый HTML*). Первая версия, XHTML 1.0, почти идентичная спецификации HTML 4.01, содержит те же элементы и атрибуты, но имеет более жесткие требования относительно того, как должна выполняться разметка (см. врезку «Требования к разметке XHTML»)

HTML 4.01 и его более строгий коллега XHTML 1.0 на основе XML, стали краеугольным камнем движения по развитию веб-стандартов (см. врезку «Проект по созданию веб-стандартов»). На момент написания книги они по-прежнему оставались наиболее тщательно и согласованно поддерживаемыми стандартами (хотя HTML5 быстро набирает обороты).

Но консорциум Всемирной паутины не останавливается на достигнутом. Не забывая идею создания Всемирной паутины на основе XML, он начал работу над XHTML 2.0 — еще более смелой, чем HTML 4.01, попыткой заставить все работать «правильно». Проблема в том, что этот язык оказался несовместим со старыми стандартами и поведением браузеров. Процесс написания и утверждения затянулся на годы. Без реализации в браузерах создание XHTML 2.0 застопорилось.

Требования к разметке XHTML

- Имена элементов и атрибутов следует указывать в нижнем регистре.

  В стандарте HTML регистр не учитывается.
- Все элементы должны быть закрыты (замкнуты). Пустые элементы закрываются добавлением слеша перед закрывающей скобкой (например, **
**).
- Значения атрибутов необходимо заключать в кавычки. Одинарные кавычки приемлемы наравне с двойными, так как они используются повсеместно. Кроме того, не должно быть никаких лишних пробельных символов (пробелов или переводов строк) до или после значения атрибута внутри кавычек.
- Все атрибуты обязаны иметь явные значения. Формат XML (и, следовательно, XHTML) не поддерживает *минимизацию атрибутов*, которая практиковалась языком SGML и позволяла сводить написание некоторых атрибутов к простому указанию их значений. Таким образом, если в HTML-документе можно просто написать **checked**, что обозначает, что элемент формы будет активен в момент загрузки страницы, то в рамках стандарта XHTML необходимо явно указать **checked="checked"**.
- Правильное вложение документов строго обязательно. У некоторых элементов появились новые ограничения по вложению.
- Для вставки специальных символов следует указывать их символьную ссылку, (например, **&amp** для символа &).
- В качестве идентификатора используется атрибут **id**, а не **name**.
- Сценарии должны располагаться в разделе CDATA, таким образом они будут рассматриваться как обычные текстовые символы и не станут анализироваться как разметка XML. Ниже приведен пример синтаксиса:

```
<script type="type/javascript">
// <![CDATA[
... здесь следует код сценария
JavaScript...
// ]]>
</script>
```

Проект по созданию веб-стандартов

В 1998 году, в разгар войны браузеров, группа, получившая название Web Standards Project (WaSP для краткости) стала призывать создателей браузеров (в то время — компании Netscape и Microsoft) придерживаться открытых стандартов, как они прописаны консорциумом Всемирной паутины. Не останавливаясь на достигнутом, они рассказывали сообществам веб-разработчиков о многочисленных преимуществах разработки в соответствии со стандартами. Их усилия коренным образом изменили способы создания и поддержки сайтов. Сейчас браузеры (даже Internet Explorer) хвастаются тем, что поддерживают стандарты, продолжая при этом вводить инновации. Вы можете прочитать о миссии, истории и текущей работе группы WaSP на сайте (**webstandards.org**).

Создание HTML5

Цель HTML5 — сделать язык HTML более удобным для создания веб-приложений.

В 2004 году сотрудники компаний Apple, Mozilla и Opera сформировали рабочую группу по разработке гипертекстовых приложений для Интернета (Web Hypertext Application Technology Working Group, WHATWG, **whatwg.org**) отдельно от консорциума Всемирной паутины. Целью WHATWG стало дальнейшее развитие HTML в соответствии с новыми требованиями таким образом, чтобы он согласовывался с реальной практикой верстки веб-страниц и поведением браузеров (в отличие от идеала, описываемого XHTML 2.0, где вся работа начиналась с нуля). Их исходные документы, Веб-приложения 1.0 и Веб-формы 1.0 объединились в HTML5, который на момент написания книги находится в разработке под руководством редактора, Йена Хиксона (Ian Hickson) (в настоящее время работает в компании Google).

ПРИМЕЧАНИЕ

Группа WHATWG поддерживает так называемый «живой стандарт» языка HTML (имеется в виду, что номер версии не присваивается) на сайте **www.whatwg.org**. Он почти идентичен HTML5, но включает в себя несколько дополнительных элементов и атрибутов, которые не приняты консорциумом Всемирной паутины, и имеет несколько иную линейку API-интерфейсов.

Консорциум Всемирной паутины в конечном итоге создал собственную рабочую группу по HTML5 (также во главе с Хиксоном), взяв за основу работу, проделанную группой WHATWG. Работа над спецификацией HTML5 велась в обеих организациях совместно, иногда с противоположными результатами. На момент написаниякниги язык HTML5 еще не был формально рекомендован, но это не мешает браузерам постепенно реализовывать его поддержку.

Относительно XHTML 2.0. В конце 2009 года консорциум Всемирной паутины официально прекратил работу над ним, закрыв рабочую группу и направив свои ресурсы и усилия на HTML5.

На этом завершается длинное вступление к главе, которая посвящена новым возможностям спецификации HTML5. Я также рекомендую прочитать врезку «Интересные факты об HTML5», где вы найдете любопытные сведения о данной спецификации. Далее вы узнаете следующее:

- Новая декларация типа документа DOCTYPE
- Новые элементы и атрибуты
- Устаревшие элементы спецификации 4.01
- Интерфейсы прикладного программирования (API-интерфейсы).

Интересные факты об HTML5

Спецификация HTML5 основывается на предыдущих версиях языка HTML и вводит некоторые значительные изменения. Ниже приведено несколько интересных моментов из спецификации HTML5.

- HTML5 основан на HTML 4.01 Строго говоря, на версии HTML, которая не содержит презентационных или иных устаревших элементов и атрибутов. Это означает, что подавляющее большинство элементов в языке HTML5 те же, которые мы использовали в течение многих лет, и браузеры знают, что с ними делать.
- В HTML5 не используется *определение типа документа* (Document Type Definition, DTD), представляющее собой документ, определяющий все элементы и атрибуты в языке разметки. Таким образом документируется язык в SGML, и если вы помните, HTML изначально создавался в соответствии с правилами SGML. HTML 4.01 определяли три разных DTD: *Transitional* (в том числе элементы наследования, которые были помечены как «устаревшие» или выходящие из употребления), *Strict* (из которого исключены устаревшие функции) и *Frameset* (для документов, поделенных на несколько прокручиваемых фреймов; в настоящее время техника считается устаревшей).
- HTML5 — первая спецификация HTML, которая содержит подробные инструкции о том, как браузеры должны обрабатывать неправильную и наследованную разметку. Инструкции основываются на поведении браузера в случаях наследования, но на этот раз есть стандартный протокол, которому должны следовать разработчики, когда их браузеры сталкиваются с неправильной или нестандартной разметкой.
- HTML5 также может быть написан в соответствии с более строгим синтаксисом XML (т.н. *XML-сериализация HTML5*). Некоторые разработчики стали предпочитать характеристики «опрятного» стандарта XHTML (имена элементов указываются в нижнем регистре, значения атрибутов заключены в кавычки, все элементы закрываются и т. д.), так что этот способ верстки по-прежнему остается как вариант, хотя и не обязателен. В крайних случаях может потребоваться, чтобы HTML5-документ использовался как XML для работы с другими приложениями XML. В этом случае он может использовать синтаксис XML и будет готов к работе.
- В дополнение к разметке, HTML5 определяет ряд API-интерфейсов (интерфейсов прикладного программирования). Они облегчают связь с веб-приложениями, а также возлагают некоторые распространенные процессы (такие как аудио- и видеопроигрыватели) на встроенный функционал браузеров.

Особенности разметки

Сначала мы рассмотрим аспекты HTML5, касающиеся разметки, а затем перейдем к API-интерфейсам.

Минимальная декларация DOCTYPE

Как мы видели в главе 4, HTML документы должны начинаться с декларации типа документа (декларация DOCTYPE), которая определяет, какая версия HTML документа представлена ниже. Декларация HTML5 лаконична:

```
<!DOCTYPE HTML>
```

Сравните ее с декларацией типа документа в строгой версии HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4.01/strict.dtd">
```

Определения DOCTYPE языка HTML

Ниже перечислены все определения DOCTYPE, находящиеся в общем пользовании.

HTML5

```
<!DOCTYPE html>
```

HTML 4.01 Transitional

Переходное определение DTD включает в себя устаревшие элементы и атрибуты:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
```

HTML 4.01 Strict

Строгое определение DTD опускает все устаревшие элементы и атрибуты:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4.01/strict.dtd">
```

HTML 4.01 Frameset

Если ваш документ содержит фреймы, то есть вместо элемента **body** используется **frameset**, тогда укажите определение DTD с фреймами:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//
EN"
"http://www.w3.org/TR/HTML4.01/frameset.dtd">
```

XHTML 1.0 Strict

Аналогично HTML 4.01 Strict, но переформулировано в соответствии с правилами синтаксиса XML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1.0 Transitional

Аналогично HTML 4.01 Transitional, но переформулировано в соответствии с правилами синтаксиса XML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
dtd">
```

XHTML 1.0 Frameset

Аналогично HTML 4.01 Frameset, но переформулировано в соответствии с правилами синтаксиса XML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//
EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Почему так сложно? В HTML 4.01 и XHTML 1.0/1.1 декларации должны указывать на публичное определение *DTD* (*определение типа документа*), документ, определяющий все элементы в языке разметки, а также правила их использования. Для языка HTML 4.01 определяли три разных DTD: *Transitional* (в том числе элементы наследования, такие как **font**, и атрибуты, например, **align**, которые были помечены как «устаревшие» или выходящие из употребления), *Strict* (из которого исключены устаревшие функции) и *Frameset* (для документов, поделенных на несколько прокручиваемых фреймов; в настоящее время техника считается устаревшей). В спецификации HTML5 нет определения DTD, поэтому используется простая декларация DOCTYPE.

Определения DTD — это наследие языка SGML и они оказались не так полезны во Всемирной паутине, как предполагалось изначально, поэтому верстальщики на языке HTML5 просто ими не пользуются.

Валидаторы — программы, которые проверяют, чтобы вся разметка в документе была верна (см. примечание) — используют декларации DOCTYPE, чтобы убедиться, что документ соблюдает правила заявленной спецификации. Во врезке «Определения DOCTYPE языка HTML» приведен список всех деклараций, находящихся в общем пользовании, на случай, если вам потребуется верстать документы на языке HTML 4.01 или XHTML 1.0.

ПРИМЕЧАНИЕ

Для проверки правильности разметки вашего HTML-документа воспользуйтесь онлайн-валидатором консорциума Всемирной паутины (**validator.w3.org**).

Специальный валидатор разметки страниц на языке HTML5 также доступен на сайте **html5.validator.nu**. В программе Adobe Dreamweaver реализован встроенный валидатор, который позволяет проверить документ на соответствие различным спецификациям, с которыми вы работаете.

Элементы и атрибуты

В языке HTML5 введен ряд новых элементов. Они разбросаны по всей книге и перечислены в табл. 10.1.

ПРИМЕЧАНИЕ

Подробный перечень отличий HTML5 от HTML 4.01 вы найдете в официальной документации консорциума Всемирной паутины на сайте **www.w3.org/TR/html5-diff/**.

Табл. 10.1. Новые элементы в спецификации HTML5

article	datalist	header	output	source
aside	details	hgroup	progress	summary
audio	embed	keygen	rp	time
bdi	figcaption	mark	rt	track
canvas	figure	meter	ruby	video
command	footer	nav	section	wbr

Новые типы ввода данных

Мы рассмотрели новые типы ввода данных в форму в главе 9, далее они все перечислены: **color**, **date**, **datetime**, **datetime-local**, **email**, **month**, **number**, **range**, **search**, **tel**, **time**, **url** и **week**.

Новые глобальные атрибуты

Глобальными являются атрибуты, которые можно применить к любому элементу. В HTML5 число глобальных атрибутов было увеличено, и многие из них — новые (как указано в табл. 10.2). На момент напи-

сания этой книги, консорциум Всемирной паутины все еще добавлял/удалял атрибуты, поэтому следует свериться со спецификацией, чтобы узнать последние данные (dev.w3.org/html5/spec/global-attributes.html#global-attributes).

Табл. 10.2. Глобальные атрибуты в спецификации HTML5

Атрибут	Значения	Описание
`accesskey`	Один текстовый символ	Назначает ссылке клавишу доступа (сочетание клавиш). Клавиши доступа также применяются для полей формы. Пользователи могут перейти к элементу, нажав сочетание клавиш Alt+<*клавиша*> (Windows) или Ctrl+<*клавиша*> (OS X).
`aria-*`	Одно из стандартизированных ключевых слов, обозначающее состояние или свойство в спецификации WAI-ARIA (www.w3.org/TR/wai-aria/states_and_properties)	Стандарт доступности многофункциональных веб-приложений (*WAI-ARIA*, Accessibile Rich Internet Applications) определяет способ повышения доступности веб-контента и приложений для пользователей со вспомогательными устройствами. Язык HTML5 позволяет добавлять в элементы любые свойства и роли спецификации ARIA. Например, элемент `div`, используемый для создания меню, может содержать атрибут `aria-haspopup`, чтобы это свойство было понятно пользователям, не имеющим визуального браузера. Также изучите близкий ему глобальный атрибут `role`.
`class`	Строка текста.	Присваивает элементу одно или несколько классификационных имен.
`contenteditable`	`true|false`	Новый в HTML5 Указывает, что пользователь может отредактировать элемент. Этот атрибут уже хорошо поддерживается текущими версиями браузеров.
`contextmenu`	Идентификатор элемента `menu`	Новый в HTML5 Определяет контекстное меню, применяемое к элементу. Оно может быть вызвано пользователем, например, щелчком правой кнопкой мыши.
`data-*`	Текстовая строка или числовое значение	Новый в HTML5 Позволяет верстальщикам создавать пользовательские атрибуты к данным (символ «*» означает «любой») например, `data-length`, `data-duration`, `data-speed`, чтобы данные могли использоваться пользовательскими приложениями или сценариями.
`dir`	`ltr | rtl`	Определяет направление элемента («слева направо» или «справа налево»).
`draggable`	`true | false`	Новый в HTML5 Значение `true` указывает, что элемент можно переместить, нажав и удерживая на нем кнопку мыши, в другую позицию в окне.

Атрибут	Значения	Описание
dropzone	copy|link|move|s:text/plain|f:file-type (например, f:image/jpg)	Новый в HTML5 Указывает, что в элемент можно перетащить текст или файловые данные. Значения представляют собой список элементов, разделенных пробелами, содержащий приемлемые типы данных (s:text/plain для текстовых строк; f:file-type для типов файлов) и ключевое слово, сообщающее, какое действие следует совершить с перенесенным контентом: copy — копирование; move — перенос или link — указание ссылки на оригинал.
hidden	В HTML-документах его значение не указывается. В XHTML задайте значение hidden="hidden"	Новый в HTML5 Не позволяет агенту пользователя (браузеру) визуализировать элемент и его потомков. Любые сценарии или элементы формы в скрытых разделах по-прежнему будут работать, но не покажутся пользователю.
id	Текстовая строка (может начинаться не с числа)	Присваивает элементу уникальное имя идентификатора.
lang	Двухбуквенное кодовое обозначение используемого языка (см. www.loc.gov/standards/iso639-2/php/code_list.php)	Определяет язык элемента по его языковому коду.
role	Одно из стандартных ключевых слов в стандарте WAI-ARIA (см. www.w3.org/TR/wai-aria/roles)	Новый в HTML5 Присваивает элементу одну из стандартизированных ролей WAI-ARIA, чтобы его назначение было понятно пользователям с ограниченными возможностями. Например, элемент div, содержащий контент, который отображается в визуальных браузерах в виде меню, может быть помечен атрибутом role="menu", чтобы пояснить его в программах экранного доступа.
spellcheck	true | false	Новый в HTML5 Указывает на необходимость проверки орфографии и грамматики элемента.
style	Перечень правил стилей, разделенных точкой с запятой (пары *свойство: значение*)	Связывает информацию с о стиле с элементом. Например: Heading
tabindex	Числовое значение	Определяет последовательность перехода к данному элементу текущего документа. Значение указывается в диапазоне от 0 до 32 767. Оно используется при перемещении по ссылкам или элементам формы на странице с помощью клавиши Tab и полезно для вспомогательных устройств. Значение –1 допустимо для удаления элементов формы из последовательности переходов и тогда они будут доступны только с помощью JavaScript.
title	Текстовая строка	Добавляет заголовок или рекомендации к элементу. Обычно отображается в виде подсказки.

Устаревшая разметка HTML 4.01

Также в HTML5 некоторые элементы HTML 4.01 были признаны «устаревшими», так как носят презентационный характер, вышли из употребления или плохо поддерживаются браузерами (табл. 10.3). Если вы их используете, браузеры будут поддерживать такие элементы, но я настоятельно рекомендую забыть о них.

Табл. 10.3. Элементы HTML 4, устаревшие в HTML5

acronym	dir	noframes
applet	font	strike
basefont	frame	tt
big	frameset	
center	isindex	

API-интерфейсы в спецификации HTML5

Спецификации HTML, предшествовавшие HTML5, содержали документацию только по элементам, атрибутам и значениям, допустимым в языке. Это хорошо для простых текстовых документов, но разработчики HTML5 планировали облегчить процесс создания веб-приложений, для которых требуются сценарии и программирование. По этой причине, чтобы облегчить взаимодействие с приложением, HTML5 также определяет ряд новых API-интерфейсов.

Интерфейс прикладного програмирования (Application Programming Interface, *API*) — это задокументированный набор команд, имен данных и так далее, который позволяет одному программному приложению общаться с другим. Например, разработчики Twitter указали имена каждого типа данных (пользователи, твиты, метки и т. д.) и методы для доступа к ним в документе API-интерфейса (**dev.twitter.com/docs**), что позволяет другим разработчикам добавлять каналы и элементы Twitter в свои продукты.

Поэтому существует так много программ и виджетов, поддерживающих сеть Twitter.

Интернет-магазин Amazon.com также демонстрирует информацию о своих товарах через API-интерфейс. В самом деле, все издатели признают, что очень хорошо, если ваш контент доступен таким образом. Можно сказать, что API-интерфейсы сейчас очень популярны.

Но вернемся к HTML5, содержащему API-интерфейс для решения задач, для которых обычно требовались собственные плагины (например, Flash) или пользовательское программирование.

Идея состоит в том, что если браузеры изначально предлагают эти функции со стандартизованными наборами методов для доступа к ним, разработчики могут создавать замечательные вещи и рассчитывать на то, что они будут работать во всех браузерах, так же, как мы сегодня привыкли к возможности вставлять изображения на страницы. Конечно, пройдет еще немало времени, пока поддержка этих передовых функций станет повсеместной, но прогресс налицо. Некоторые API-интерфейсы содержат компонент разметки, такой как вложение мультимедийного контента с новыми HTML5-элементами **video** и **audio**. Другие выполняются полностью «за кадром» с применением сценариев JavaScript или серверных компонентов, например, создание веб-приложений, которые работают даже при отсутствии подключения к Интернету (API-интерфейс механизма кэширования данных веб-приложений).

Консорциум Всемирной паутины и сообщество WHATWG трудятся над *множеством* API-интерфейсов, которые можно будет использовать с веб-приложениями, и все они находятся в той или иной стадии завершения и реализации.

Большинство из них имеют свои собственные спецификации, отдельно от HTML5, но они, как правило, включены в общую спецификацию HTML5, которая охватывает все веб-приложения. HTML5 включает в себя спецификации для следующих API-интерфейсов:

API-интерфейс для воспроизведения видео- и аудиоконтента

Применяется для управления аудио- и видеопроигрывателями, встроенными в веб-страницу при помощи новых элементов **video** и **audio**. Мы подробнее рассмотрим использование подобных типов контента далее в этой главе.

API-интерфейс истории посещений

Взаимодействует с журналом посещений браузера для лучшего контроля над кнопкой **Назад** (Back).

API-интерфейс механизма кэширования для поддержки веб-приложений

Позволяет веб-приложениям работать даже при отсутствии подключения к Интернету. Это достигается при помощи документа, где перечислены все файлы и ресурсы, которые должны быть загружены в кэш браузера для того, чтобы приложение работало. Когда соединение доступно, браузер проверяет, были ли изменены какие-либо документы, затем обновляет их.

API-интерфейс редактирования

Предоставляет набор команд, которые можно применить для создания встроенных в браузер текстовых редакторов, позволяя пользователям добавлять и удалять текст, форматировать его и многое другое. Кроме того, существует новый атрибут **contenteditable**, который позволяет редактировать любой элемент контента прямо на странице.

ПРИМЕЧАНИЕ

Полный перечень API-интерфейсов представлен в статье Эрика Уайлда, доступной по адресу **dret.typepad.com/dretblog/html5-api-overview.html**. Все документы, поддерживаемые консорциумом Всемирной паутины, многие из которых описывают API-интерфейсы, указаны на сайте **www.w3.org/TR/tr-title-all**.

API-интерфейс перетаскивания drag-and-drop

Добавляет возможность перетаскивания выделенного текста или файла в целевую область на этой или другой веб-странице. Атрибут **draggable** указывает, что элемент можно выделить и перетащить. Атрибут **dropzone** применяется к целевой области и определяет, какой тип контента она может принять (текст или тип файла), а также какие действия произвести с перемещенным контентом (значения **copy**, **link**, **move**).

Ниже приведены несколько API-интерфейсов с собственными спецификациями, находящихся в разработке консорциума Всемирной паутины (вне HTML5):

API-интерфейс двухмерного рисования

Элемент **canvas** добавляет на страницу динамическое двухмерное пространство для рисования. Мы рассмотрим его в конце этой главы.

API-интерфейс хранилища данных

Позволяет хранить данные в кэш-памяти браузера, чтобы приложение впоследствии могло их использовать. Ранее тот же результат достигался с помощью файлов cookie, но API-интерфейс позволяет сохранить больше данных. Интерфейс также определяет, будут ли они ограничены одним сеансом (**sessionstorage**: когда окно закрывается, данные удаляются) или доменом (**localstorage**: все открытые окна на этом домене имеют доступ к данным).

API-интерфейс геолокации

Позволяет пользователям обмениваться информацией об их географическом положении (широта и долгота), при этом данные доступны для сценариев в веб-приложении. Это позволяет ему обеспечить работу функций, связанных с определением координат устройства, например, предложить ближайший ресторан или найти других пользователей в этом городе.

API-интерфейс Web Worker

Предоставляет возможность в фоновом режиме запускать сценарии со сложными вычислениями. Он позволяет браузеру сохранять быстроту ответа интерфейса веб-страницы на действия пользователя, одновременно с этим выполняя сценарии, требующие больших ресурсов процессора. API-интерфейс Web Workers — часть спецификации HTML5 сообщества WHATWG, но в рекомендациях консорциума Всемирной паутины он выделен в отдельный документ.

API-интерфейс веб-сокетов

Создает «сокет» — открытое соединение между клиентом (браузером) и сервером. Оно позволяет информации передаваться в режиме реального времени, без задержек из-за традиционных HTTP-запросов. Интерфейс удобен для многопользовательских игр, чатов или потоков данных, которые постоянно обновляются, таких как спортивные результаты, котировки акций или потоки сообщений в социальных сетях.

ПРИМЕЧАНИЕ

Веб-сокет можно представить как телефонный сеанс связи между браузером и сервером в отличие от «рации» — поочередного способа общения браузера и сервера.

В некоторых API-интерфейсах имеются взаимодействующие с ними HTML-элементы такие, как **audio** и **video** для встраивания на страницу мультимедийных проигрывателей, а также элемент **canvas** для добавления динамической области рисования. В следующих разделах мы вкратце изучим, как они применяются.

Видео- и аудиоконтент

В первые годы существования Всемирной паутины можно было добавить на страницу MIDI-файл, чтобы получить коротенький примитивный саундтрек (вспомните первые видеоигры). Вскоре появились возможности получше, в том числе технологии RealMedia и Windows Media, что позволило встраивать в веб-страницы контент всевозможных аудио- и видеоформатов. В конце концов технология Flash *фактически* стала встроенным мультимедийным проигрывателем, отчасти благодаря тому, что ее использовали YouTube и аналогичные сервисы видеохостинга.

Общая черта всех этих технологий в том, что для воспроизведения мультимедийных файлов требуется загрузить и установить дополнительные плагины. До недавнего времени в браузерах не было встроенных возможностей для поддержки аудио и видеоконтента, поэтому нишу заполняли плагины. С развитием Всемирной паутины как платформы открытых стандартов, оказалось, что пришло время реализовать поддержку мультимедийных элементов в качестве новейших возможностей браузеров — ввести новые элементы **audio** и **video** и соответствующие API-интерфейсы.

Прощай, Flash?

Объявление корпорации Apple о прекращении поддержки технологии Flash на устройствах под управлением операционной системы iOS, привело к значительному продвижению HTML5 и фактическому прекращению компанией Adobe разработки продуктов Flash для мобильных устройств. Вскоре после этого корпорация Microsoft объявила, что прекращает использование своего проигрывателя Silverlight в пользу альтернативы, предлагаемой HTML5. На момент написания книги языку HTML5 было еще далеко до огромных возможностей и функциональности Flash, но со временем он их достигнет.

Скорее всего, еще несколько лет мы будем использовать проигрыватели Flash и Silverlight, но тенденция перехода от плагинов к веб-технологиям уже очевидна.

Хорошая и плохая новости

Хорошая новость заключается в том, что элементы **audio** и **video** хорошо поддерживаются современными браузерами, включая Internet Explorer, Safari, Chrome, Opera и Firefox для компьютеров и устройств под управлением операционной системы IOS и Android (кроме браузера Opera Mini).

Но отставим идеальный мир, где браузеры поддерживают аудио- и видеоконтент в совершенной гармонии, на деле все не так просто. Хотя браузеры определились с разметкой и сценариями JavaScript для встраивания и управления мультимедийными проигрывателями, к сожалению, они не договорились, какие форматы поддерживать. Далее мы совершим небольшое путешествие в мир форматов мультимедийных файлов. Если вы хотите добавлять аудио или видеоконтент на свою страницу, этот материал важно изучить.

Форматы мультимедийных файлов

При подготовке аудио или видеоконтента для публикации во Всемирной паутине, вам придется принять два решения относительно форматов. Во-первых, как *кодировать* мультимедийный файл (алгоритмы,

Для дополнительного чтения: мультимедийный контент средствами HTML5

Я рекомендую прочитать эти книги, когда вы будете готовы больше узнать о мультимедийном контенте средствами HTML5:

- «Самоучитель HTML5» Билла Сандерса (Эксмо, 2012)
- «HTML5 и CSS3 для всех» Алексиса Голдстайна, Луиса Лазариса и Эстель Уэйл (Эксмо, 2012)

используемые для преобразования одного формата в другой, и способы компрессии).

Метод, используемый для кодирования называется *кодек*, что является сокращением от слов *код*ировать/*дек*одировать. Существует несметное множество кодеков. Какие-то, вероятно, звучат знакомо, например, MP3, а другие могут казаться новыми, например, H.264, Vorbis, Theora, VP8 и AAC. К счастью, лишь немногие из них подходят для Всемирной паутины, и далее мы их рассмотрим. Во-вторых, вам нужно выбрать формат контейнера для мультимедийного файла — можете считать его чем-то вроде ZIP-архива, содержащего файл и его метаданные. Обычно в формате контейнера может храниться несколько типов кодеков, но все это довольно запутанно. Поскольку объем главы ограничен, я не буду распространяться на эту тему, и расскажу о наиболее часто используемых сочетаниях кодеков и контейнеров во Всемирной паутине. Если вы собираетесь добавлять на свой сайт видео или аудиоконтент, я рекомендую подробнее изучить все эти форматы. Для начала замечательно подойдут книги, перечисленные во врезке «Для дополнительного чтения: мультимедийный контент средствами HTML5».

Обзор видеоформатов

Наиболее распространенные форматы для видеоконтента:

- **Контейнер Ogg + видеокодек Theora + аудиокодек Vorbis.** Как правило, формат называется «Ogg Theora», и файл имеет расширение *.ogv*. Все кодеки и контейнер в этом варианте с открытым исходным кодом и без патентных или лицензионных ограничений, что делает их идеальными для распространения во Всемирной паутине. Но говорят, что их качество хуже, чем у других форматов.
- **Контейнер MPEG-4 + видеокодек H.264 + аудиокодек AAC.** Это сочетание обычно называют «MPEG-4», и файл имеет расширение. *mp4* или *.m4v*. H.264 — высококачественный и гибкий видеокодек, но он запатентован и на его использование необходимо получить платную лицензию. Лицензионные требования стали основной причиной, почему браузеры отказываются его поддерживать.
- **Контейнер WebM + видеокодек VP8 + аудиокодек Vorbis.** «WebM» — это новейший формат контейнера, для которого используется расширение файла *.webm*. Он предназначен для работы исключительно с VP8 и Vorbis, и его преимущество заключается в наличии открытого исходного кода и отсутствии лицензионных отчислений.

Конечно, как упоминалось выше, проблема заключается в том, что разработчики браузеров не договорились о поддержке единого формата. Некоторые используют бесплатные варианты с открытым исходным кодом, такие как Ogg Theora или WebM. Другие придерживаются более качественного H.264, несмотря на необходимость получения лицензии. А значит нам, веб-дизайнерам, нужно создавать несколько версий видеофайлов, чтобы обеспечить их поддержку во всех браузе-

рах. В табл. 10.4 показано, какие браузеры поддерживают различные видеоформаты.

Табл. 10.4. Поддержка видеоформатов в популярных браузерах (июль 2013 года)

Формат	Тип	Internet Explorer	Chrome	Firefox	Safari	Opera	iOS Safari	Android Browser
Ogg/Theora	video/ogg	–	4.0+	3.5+	–	10,5+	–	–
MP4/H.264	video/mp4	–/9.0+	4.0+	21.0+	3.2+	–	3.2+	2.1+
WebM	video/webm	–	6.0+	4.0+	–	10,6+	–	2.3+

Обзор аудиоформатов

С аудиоформатами похожая ситуация: доступно несколько на выбор, но ни один не поддерживается всеми браузерами (табл. 10.5).

- **MP3.** Формат MP3 — это кодек и контейнер одновременно, расширение файла *.mp3*. Он используется повсеместно для размещения скачиваемой музыки. MP3 (сокращение от MPEG-1 Audio layer 3) запатентован, и компании, производящие программное обеспечение и комплектующие (не создатели мультимедийных файлов) обязаны приобретать лицензию на его использование.
- **WAV.** Формат WAV (*.wav*) также кодек и контейнер одновременно.
- **Контейнер Ogg + аудиокодек Vorbis.** Его обычно называют «Ogg Vorbis» и файл имеет расширение *.ogg* или *.oga*.
- **Контейнер MPEG-4 + аудиокодек AAC.** Формат «MPEG4 audio» (*.m4a*) используется реже, чем MP3.
- **Контейнер WebM + аудиокодек Vorbis.** Формат WebM (*.webm*) также может содержать только аудиоконтент.

Табл. 10.5. Поддержка аудиоформатов в популярных браузерах (июль 2013 года)

Формат	Тип	Internet Explorer	Chrome	Firefox	Safari	Opera	iOS Safari	Android Browser
MP3	audio/mpeg	9.0+	5.0+	–	4+	–	3.0+	2.0+
WAV	audio/wav или audio/wave	–	5.0+	3.5+	–	10.5+	3.0+	2.0+
Ogg Opus	audio/ogg; codecs=opus	–	–	15.0+	–	–	–	–
Ogg Vorbis	audio/ogg	–	5.0+	3.5+	–	10.5+	–	2.0+
MPEG.4/AAC	audio/mp4	9.0+	5.0+	–	4+	–	3.0+	2.0+
WebM	audio/webm	–	6.0+	4.0+	–	11+	–	2.3+

Инструменты кодирования аудио и видео

Существует множество приложений для редактирования и кодирования видео- и аудиофайлов, ниже я приведу некоторые из них. Все они бесплатны.

Преобразование видеофайлов

- Miro Video Converter (**www.mirovideoconverter.com**) — бесплатный инструмент, преобразующий любой видеофайл в формат H.264, Ogg Theora или WebM, оптимизированный для мобильных устройств или компьютеров, с простым интерфейсом. Он доступен для операционных систем OS X и Windows.
- Handbrake (**handbrake.fr**) — популярный инструмент с открытым исходным кодом с доступом ко множеству настроек формата H.264. Он доступен для операционных систем Windows, OS X и Linux.
- Firefogg (**firefogg.org**) — расширение браузера Firefox для преобразования видеофайлов в формат Ogg Theora. Установите его в браузере Firefox, затем перейдите на сайт Firefogg и преобразуйте видеофайлы с помощью веб-интерфейса.

Преобразование аудиофайлов

- Mp3/Wma/Ogg Converter (**www.freemp3wmaconverter.com**) — бесплатный инструмент, который преобразует аудиоформаты MP3, WAV, WMA, OGG, AAC и многие другие. Доступен только для операционной системы Windows.
- Для операционной системы OS X доступен аудиоконвертер Max с открытым исходным кодом, загрузить который можно с сайта **sbooth.org/Max/**. В программе Audacity (**audacity.sourceforge.net/**) также доступно несколько основных инструментов преобразования, в дополнение к функциям записи аудио.

Добавление видеоконтента на страницу

`<video>...</video>`

Добавляет на страницу видеопроигрыватель

Новый в HTML5

Теперь самое время перейти к разметке для добавления видеоконтента на веб-страницу. Давайте начнем с примера, когда вы точно знаете, каким браузером будут пользоваться посетители. В этом случае, вы можете предоставить видеофайл только в одном формате, используя атрибут **`src`** элемента **`video`** (так же, как и для элемента **`img`**).

На рис. 10.1 показан видеоролик с используемым по умолчанию проигрывателем в браузере Internet Explorer. После примера мы рассмотрим другие атрибуты.

```
<video src="dirt-jumping.mp4" width="640" height="480"
poster="dirt-jumping.jpg" controls autoplay></video>
```

Ниже приведены несколько интересных атрибутов из этого примера, которые следует рассмотреть в подробностях.

`width="`*`ширина в пикселах`***`"`**

`height="`*`высота в пикселах`***`"`**

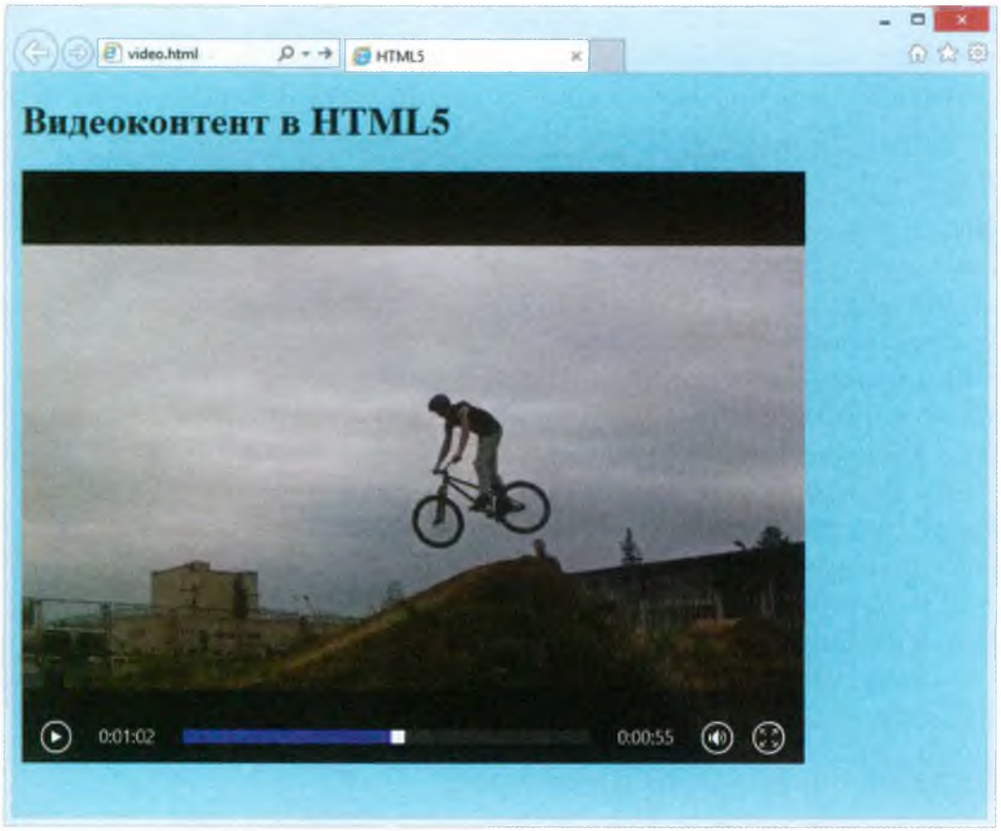


Рис. 10.1. Встроенный видеоролик, использующий элемент **`video`** *(браузер Internet Explorer 10)*

Определяет размер окна, которое встроенный мультимедийный проигрыватель занимает на экране. Рекомендуется установить размеры точно соответствующие в пикселах разрешению изображения видеоролика. Разрешение изображения видеоролика будет изменяться в соответствии с указанными значениями.

`poster="`*`URL-адрес изображения`*`"`

Определяет расположение графического изображения, которое отображается на экране перед воспроизведением видеоролика.

`controls`

Добавление атрибутов **`controls`** подсказывает браузеру отображать встроенные элементы управления мультимедийным файлом — как правило, кнопку воспроизведения/паузы, ползунковый регулятор, позволяющий перейти к определенному фрагменту ролика, а также регулятор уровня громкости. Можно создать собственный пользовательский интерфейс проигрывателя с помощью CSS и JavaScript, если вы хотите добиться единообразия внешнего вида в различных браузерах. Описание, как это сделать, не входит в задачи данной главы, но приводится в книгах, указанных в списке во врезке «Для дополнительного чтения: мультимедийный контент средствами HTML5». Во многих случаях замечательно подходят элементы управления, установленные по умолчанию.

`autoplay`

Задает автоматическое воспроизведение видеоролика, как только он загрузится настолько, чтобы воспроизводиться без остановки. В общем, старайтесь не использовать атрибут **autoplay**, позволяя пользователю самому решать, когда должно начаться воспроизведение видеоконтента.

Кроме того, элемент **video** (как и **audio**) может использовать атрибут **loop**, чтобы зациклить воспроизведение видеоролика (до бесконечности); атрибут **muted** для воспроизведения видеоролика без звука; атрибут **mediagroup**, чтобы сделать элемент **video** частью группы связанных мультимедийных элементов (таких, как видеофайл и файл с субтитрами); и атрибут **preload**, чтобы указать браузеру, следует ли передавать видеоданные сразу после загрузки страницы (**preload="auto"**) или подождать, пока пользователь нажмет кнопку воспроизведения (**preload="none"**). Присвоение значения **preload="metadata"** загружает информацию о мультимедийном файле, но не сам файл. Устройство может решить, как лучше реагировать на значение **auto**, например, браузер смартфона может предотвратить передачу данных автоматически, даже если задано значение **auto**.

Поддержка для всех браузеров

Мы уже знаем, что одного видеоформата в реальном мире недостаточно. По меньшей мере, вам нужно создать две версии видео: Ogg Theora и MPEG-4 (H.264). Некоторые разработчики предпочитают WebM вместо Ogg потому, что он поддерживается браузерами почти так же широко, а размер файлов — меньше. Как запасной вариант для пользователей с браузерами, которые не поддерживают элемент **video**, вы можете вставить на страницу Flash-проигрыватель или воспользоваться таким сервисом, как YouTube или Vimeo, преобразовав файл и скопировав код для вставки.

В разметке на каждый видеофайл указывают элементы **source**, находящиеся внутри элемента **video**. Браузеры просматривают список сверху вниз, пока не найдут тот формат, который они поддерживают и загружают только эту версию. Запасной вариант на основе технологии Flash реализуется с помощью традиционных элементов **object** и **embed**, поэтому, если браузер не может разобраться в элементах **video** и **source**, велика вероятность, что он сумеет воспроизвести файл в проигрывателе Flash. Наконец, в целях обеспечения доступности для всех, настоятельно рекомендуем вам добавить обычные ссылки для загрузки видеофайла, чтобы его можно было воспроизвести в любом установленном на компьютере клиента видеопроигрывателе, если все остальные варианты не будут работать.

Ниже приведен пример кода для встраивания видеоконтента, который должен воспроизводиться на устройствах всех пользователей, в том числе и мобильных устройствах.

Вы можете не предоставлять все или некоторые из этих мультимедийных форматов, поэтому адаптируйте свой код соответствующим образом.

За основу в следующем примере взят код, приведенный в статье Крока Камена (**camendesign.com/code/video_for_everybody**). Я настоятельно рекомендую посмотреть, не обновлялась ли страница, нет ли на ней инструкций по изменению кода и других технических деталей. Мы рассмотрим каждую часть после примера.

```
<video id="yourmovieid" width="640" height="360"
poster="yourmovie_still.jpg" controls preload="auto">
<source src="yourmovie-main.mp4" type='video/mp4;
codecs="avc1.4D401E, mp4a.40.2"'>
<source src="yourmovie.webm" type='video/webm; codecs="VP8,
vorbis"'>
<source src="yourmovie.ogv" type='video/ogg;
codecs="theora,
vorbis"'>
<!—Запасной вариант Flash -->
<object width="640" height="360" type="application/x-
shockwaveflash" data="your_flash_player.swf">
<param name="movie" value="your_flash_player.swf">
<param name="flashvars" value="controlbar=over&amp;image=po
ster.
jpg&amp;file=yourmovie-main.mp4">
<img src="poster.jpg" width="640" height="360" alt=""
title="Не поддерживается функция воспроизведения видео, по-
жалуйста, загрузите видеофайл, приведенный ниже">
</object>
</video>
<p>Загрузить файл Highlights Reel:</p>
<ul>
<li><a href="yourmovie.mp4">Формат MPEG-4</a></li>
<li><a href="yourmovie.ogv">Формат Ogg Theora</a></li>
</ul>
```

В каждом элементе **source** содержатся данные о расположении мультимедийных файлов (**src**) и информация о типе файла (**type**). Помимо перечисления типа MIME файла-контейнера (например, **video/ogg**) было бы полезно также указать список используемых кодеков (см. примечание). Это особенно важно для видеороликов формата MPEG-4, потому что у кодека H.264 несколько различных профилей, таких как *baseline* (используется на мобильных устройствах), *main* (используется браузерами настольных ПК Safari и IE9 +), *extended* и *high* (эти два обычно не используются для размещения видеоконтента во Всемирной паутине). У каждого профиля есть собственный идентификатор, как вы видите в первом элементе **source** кода примера.

Элементы object и embed

Элемент **object** — это универсальный способ внедрения в веб-страницу мультимедийного контента — видеофайлов, Flash-роликов, апплетов и даже изображений. Он содержит несколько элементов **param** (служащих для определения параметров), сообщающих инструкции или ресурсы, которые объект должен отобразить. Внутрь элемента **object** также можно поместить резервный контент, который отображается, если не поддерживается мультимедийный файл. Атрибуты и параметры меняются в зависимости от типа объекта и иногда уникальны для сторонних плагинов, отображающих мультимедийный контент.

Родственник элемента **object** — элемент **embed** также позволяет добавлять на страницы мультимедийные файлы. Он не входил в стандарт, но широко поддерживался браузерами, пока в конце концов не обрел официальный статус в спецификации HTML5.

Для некоторых мультимедийных файлов требуется использовать элемент **embed**, который часто применяется в качестве запасного варианта в элементе **object**, чтобы угодить всем браузерам.

Пример использования элементов **object** и **param** можно увидеть в примере кода далее.

ПРИМЕЧАНИЕ

Если вы посмотрите внимательно, то увидите, что длинная строка значений атрибута **type** в элементе **source** заключена в одинарные кавычки (**'**). Так как в двойные кавычки должны быть заключены кодеки, для всего атрибута необходимо использовать другой тип кавычек.

После элементов **source** используется элемент **object** для встраивания Flash-проигрывателя, который будет воспроизводить видеофайл в формате MPEG-4 в браузерах с Flash-плагином. Доступно множество Flash-проигрывателей, но Крок Камен рекомендует JW Player, который легко установить (просто выгрузить на сервер файлы сценария JavaScript с расширением *.js* и Flash с расширением *.swf*). Найти дистрибутив проигрывателя JW Player и инструкции по его установке и настройке можно на странице **www.longtailvideo.com/players/jw-flv-player/**. Если вы используете проигрыватель JW Player, замените в примере код *your_flash_player.swf* на **player.swf**.

Важно отметить, что запасной Flash-файл предназначен для браузеров, которые не распознают элемент **video**. Если браузер поддерживает элемент **video**, и попросту не поддерживает ни один из форматов мультимедийных файлов, он не отобразит вариант в формате Flash, появится лишь пустая область (страница). Поэтому рекомендуется с целью обеспечения максимальной доступности иметь прямые ссылки (**a**) для прямой загрузки видеофайлов помимо элемента **video**.

ПРЕДУПРЕЖДЕНИЕ

Если ваш сервер не настроен для правильной передачи типа видеоконтента (MIME-типа) ваших видеофайлов, некоторые браузеры не будут их отображать. MIME-типы для каждого формата перечислены в столбце «Тип» табл. 10.4 и 10.5. Не забудьте проконсультироваться у администратора вашего сервера или службы техподдержки хостинговой компании, если вы собираетесь загружать мультимедийные файлы, чтобы MIME-типы были установлены правильно.

Наконец, если вы хотите, чтобы видеоролик запускался автоматически, добавьте к элементу **video** атрибут **autoplay** и атрибут **autostart=true** к элементу **param** Flash, как показано ниже:

```
<video src="movie.mp4" width="640" height="480" autoplay>

<param name="flashvars" value="autostart=true&amp;controlbar
=over&amp;

image=poster.jpg&amp; file=yourmovie-main.mp4">
```

Имейте в виду, что видеоконтент не будет воспроизводиться автоматически на устройствах под управлением операционной системы iOS, даже если вы установите такой параметр в коде. Корпорация Apple отключает автозапуск на своих мобильных устройствах в целях предупреждения нежелательной передачи данных.

Добавление аудиоконтента на страницу

<audio>...</audio>

Добавляет на страницу аудиопроигрыватель

Новый в HTML5

Если вы сумели разобраться в примере разметки страницы для внедрения видеоконтента, вы уже знаете, как добавить аудиоконтент на страницу. Элемент **audio** использует те же атрибуты, что и элемент **video**, за исключением **width**, **height** и **poster** (ввиду отсутствия изображения). Так же, как и с элементом **video**, вы можете предоставить несколько вариантов аудиоформатов с помощью элемента **source**, как показано в примере ниже.

```
<audio id="soundtrack" controls preload="auto">

<source src="soundtrack.mp3" type="audio/mp3">
```

```
<source src="soundtrack.ogg" type="audio/ogg">
<source src="soundtrack.webm" type="audio/webm">
</audio>
<p>Загрузить аудиофайл:</p>
<ul>
<li><a href="soundtrack.mp3">MP3</a></li>
<li><a href="soundtrack.ogg">Ogg Vorbis</a></li>
</ul>
```

Вы можете установить автоматический запуск воспроизведения аудиофайла, а затем настроить циклическое воспроизведение:

```
<audio src="soundtrack.mp3" autoplay loop></audio>
```

Рисование средствами HTML5

Еще одним прекрасным дополнением в спецификации HTML5 является элемент **canvas** и связанный с ним API-интерфейс двухмерного рисования. Элемент **canvas** создает область на веб-странице, в которой вы можете рисовать, используя набор функций JavaScript для создания линий, фигур, заливок, текста, анимации и т. п. Вы можете использовать эту область для отображения иллюстрации, но источник большого потенциала элемента **canvas** (и восхищения, вызванного им в мире веб-разработчиков) в том, что все это создается с помощью сценариев.

Это значит, что элемент **canvas** динамичен, позволяет рисовать предметы на лету и реагировать на действия пользователя. Так он становится отличной платформой для создания анимаций, игр и даже целых приложений с помощью собственного функционала браузера и без плагинов таких, как Flash.

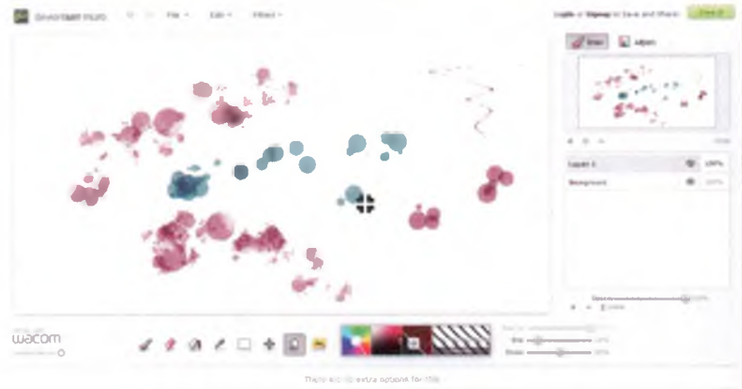
Хорошая новость в том, что элемент **canvas** на момент написания книги поддерживался всеми текущими версиями браузеров, за исключением Internet Explorer 8 и более ранних версий.

На рис. 10.2 приведено несколько примеров использования элемента **canvas** для создания игр, рисования программ, интерактивного инструмента воссоздания молекулярной структуры и анимации астероида. Дополнительные примеры можно найти на сайте **canvasdemos.com**.

Приемы использования элемента **canvas** нельзя полностью изложить в этой книге, особенно без опыта работы с JavaScript, но далее я попытаюсь вкратце объяснить, как этот элемент работает, а также приведу несколько примеров.

ie.microsoft.com/testdrive/Performance/AsteroidBelt/#

www.relfind.com/game/magician.html



muro.deviantart.com

alteredqualia.com/canvasmol/

Рис. 10.2. Несколько примеров использования элемента canvas для создания игр, анимаций и приложений

Элемент canvas

`<canvas>...</canvas>`

Двухмерная динамическая область для рисования

Новый в HTML5

С помощью элемента **canvas** можно добавить на страницу область холста и указать его размеры, используя атрибуты **width** и **height**. И на этом фактически разметка заканчивается. Для браузеров, которые не поддерживают элемент **canvas**, можно обеспечить внутри тегов резервный контент (сообщение, изображение или любой подходящий материал).

```
<canvas width="600" height="400" id="my_first_canvas">
```

Ваш браузер не поддерживает холст HTML5. Попробуйте открыть страницу в браузере Chrome, Firefox, Safari или Internet Explorer 9.

```
</canvas>
```

Разметка только освобождает пространство для рисования.

Рисование с помощью JavaScript

API-интерфейс двухмерного рисования включает в себя функции для создания базовых фигур (такие, как **strokeRect()** для рисования прямоугольного контура и **BeginPath()** для рисования линии) и их пере-

мещения (например, **rotate()** и **scale()**), а также атрибуты для применения стилей (например, **lineWidth**, **strokeStyle**, **fillStyle**, and **font**).

Следующий пример был создан моим коллегой в издательстве O'Reilly Media Сандерсом Кляйнфельдом для его книги «HTML5 for Publishers». Он был достаточно любезен и позволил мне использовать его в этой книге.

На рис. 10.3 изображен простой смайлик, который мы создадим с помощью API-интерфейса двухмерного рисования.

Рис. 10.3. *Готовый вариант нашего примера использования холста. Оригинал можно увидеть на сайте* ***examples.oreilly.com/0636920022473/my_first_canvas/my_first_canvas.html***

Ниже представлен сценарий, который его создал. Не переживайте, что вы пока не знакомы с языком JavaScript. Бегло просмотрите сценарий и обратите внимание на комментарии. Затем я опишу некоторые из используемых функций.

```
<script type="text/javascript">
window.addEventListener('load', eventWindowLoaded, false);
function eventWindowLoaded() {
canvasApp();
}
function canvasApp(){
var theCanvas = document.getElementById('my_first_canvas');
var my_canvas = theCanvas.getContext('2d');
my_canvas.strokeRect(0,0,200,225)
// для начала нарисуйте границу холста
//нарисуйте лицо
my_canvas.beginPath();
my_canvas.arc(100, 100, 75, (Math.PI/180)*0, (Math.
PI/180)*360, false);
// размеры круга
my_canvas.strokeStyle = "black"; // контур круга черного
цвета
my_canvas.lineWidth = 3; // ширина обводки три пиксела
my_canvas.fillStyle = "yellow"; // используйте желтый цвет
для заливки круга
my_canvas.stroke(); // нарисуйте круг
my_canvas.fill(); // залейте круг
// теперь, нарисуйте левый глаз
my_canvas.fillStyle = "black"; // переключитесь на черный
цвет заливки
my_canvas.beginPath();
```

```
my_canvas.arc(65, 70, 10, (Math.PI/180)*0, (Math.
PI/180)*360, false);
// размеры круга
my_canvas.stroke(); // нарисуйте круг
my_canvas.fill(); // добавьте заливку круга
my_canvas.closePath();
// теперь нарисуйте правый глаз
my_canvas.beginPath();
my_canvas.arc(135, 70, 10, (Math.PI/180)*0, (Math.
PI/180)*360, false);
// размеры круга
my_canvas.stroke(); // нарисуйте круг
my_canvas.fill(); // добавьте заливку круга
my_canvas.closePath();
// нарисуйте улыбку
my_canvas.lineWidth = 6; // задайте ширину контура равную
шести пикселам
my_canvas.beginPath();
my_canvas.arc(99, 120, 35, (Math.PI/180)*0, (Math.PI/180)*-
180, false);
// размеры полукруга
my_canvas.stroke();
my_canvas.closePath();

// Смайлик говорит!
my_canvas.fillStyle = "black"; // задайте черный цвет залив-
ки текста
my_canvas.font = '20px _sans'; // используйте шрифт без за-
сечек величиной 20 пикселов
my_canvas.fillText ("Приветики!", 45, 200); // напишите
текст
}
</script>
```

Наконец, еще немного информации о функциях API-интерфейса двухмерного рисования, используемых в примере:

strokeRect(x1, y1, x2, y2)

Рисует прямоугольный контур из точки (x1, y1) в точку (x2, y2). По умолчанию, начальная точка холста (0,0) находится в верхнем левом углу, а координаты *x* и *y* измеряются по направлению вправо и вниз.

`beginPath()`

Начало рисования линии.

`closePath()`

Окончание рисования линии, которая была начата функцией **`beginPath()`**.

`arc(`*`x, y, arc_radius, angle_radians_beg, angle_radians_end`***`)`**

Рисует дугу, где координаты (x, y) являются центром окружности, **`arc_radius`** — это длина радиуса окружности, а **`angle_radians_beg`** и **`_end`** указывают начало и конец угла дуги.

`stroke()`

Рисует линии заданной траектории. Если не добавить эту функцию, линия не появится на холсте.

`fill()`

Заполняет цветом контур, обозначенный функциями **`beginPath()`** и **`endPath()`**.

`fillText(`*ваш_текст, x1, y1*`)`

Добавляет на холст текст, начиная с указанных координат (x, y).

Кроме того, следующие атрибуты используются для определения цвета и стилей:

`lineWidth`

Ширина границ контура.

`strokeStyle`

Цвет границ.

`fillStyle`

Цвет заливки (внутренней части) фигуры, созданной с помощью контура.

`font`

Шрифт и размер шрифта текста.

Конечно, API-интерфейс двухмерного рисования включает в себя гораздо больше функций и атрибутов, чем использовано в примере. Полный список см. в спецификации консорциума Всемирной паутины на сайте **dev.w3.org/html5/2dcontext/**. Кроме того, во Всемирной паутине найдется множество учебников, посвященных двухмерному рисованию средствами HTML5.

Резюме

К этому моменту вы должны иметь четкое представление о HTML5. Мы рассмотрели новые элементы для добавления в документы улучшенной семантики, изучили различные API-интерфейсы, находящиеся в разработке, которые добавят в браузеры полезный функционал. Вы узнали, как использовать элементы **`video`** и **`audio`** для добавления на страницу мультимедийных файлов (и получили основные сведения о мультимедийных форматах), и, наконец, рассмотрели элемент **`canvas`**.

В следующей части этой книги, «Правила CSS для представления», вы узнаете, как верстать таблицы стилей для настройки внешнего вида страницы, в том числе начертания шрифта текста, цвета, фона и даже макета страницы.

ЧАСТЬ III
ПРАВИЛА CSS ДЛЯ ПРЕДСТАВЛЕНИЯ

В этой части

ГЛАВА 11

КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ

Вы уже не раз слышали упоминания о таблицах стилей, теперь мы наконец, заставим их работать и начнем придавать нашим страницам нужный стиль. Каскадные таблицы стилей (CSS, Cascading Style Sheets) — это стандарт W3C, определяющий *представление* документов, написанных на HTML, и вообще любом языке XML. Представление, опять же, относится к способу отображения документа или методу его вывода на экран компьютера, дисплей сотового телефона, печати на бумаге или при чтении программами экранного доступа. Управляя представлением при помощи таблиц стилей, язык HTML может позволить себе определять только структуру документа и его содержание, как и задумывалось изначально.

CSS — это отдельный язык со своим собственным синтаксисом. Эта глава охватывает CSS-терминологию и фундаментальные принципы, которые помогут вам разобраться что к чему в следующих главах, где вы будете изучать, как изменять текст и стили шрифтов, добавлять цвета и фон, и даже создавать базовый макет страницы при помощи CSS. К концу части III я надеюсь дать вам хорошую основу для дальнейшего самостоятельного изучения, а также предложить множество практических заданий.

В этой главе

- Преимущества и возможности каскадных таблиц стилей (CSS)
- Как разметка HTML формирует структуру документа
- Написание правил CSS
- Подключение стилей к HTML-документу
- Важные принципы наследования, каскадирования, специфичности, порядка правил и блочной модели CSS

Преимущества CSS

Не то чтобы вы нуждались в дальнейшем убеждении, что таблицы стилей — это то что надо, но на всякий случай перечислю преимущества использования таблиц стилей:

- **Четкое управление представлением и макетом при печати.** С помощью CSS можно добиться точности при печати. Предлагается даже ряд свойств специально для печатного варианта страницы (но мы не будем говорить о них в этой книге).
- **Меньший объем работы.** Вы можете менять внешний вид всего сайта редактированием одной таблицы стилей.
- **Более доступные сайты.** Когда все вопросы представления регулируются CSS, вы можете размечать контент семантически, делая его более доступным для невизуальных или мобильных устройств.

- **Надежная поддержка браузерами.** Практически каждый современный браузер поддерживает все таблицы уровней CSS2, а также многие замечательные характеристики CSS3 (см. врезку «Краткая история CSS» в конце этой главы, чтобы узнать, что имеется в виду под «уровнями» CSS).

Если подумать, то действительно нет никаких преград в использовании таблиц стилей. Есть некоторое запаздывание в устранении проблем совместимости с различными браузерами, но их можно избежать либо обойти, если знать, в чем они заключаются.

Возможности CSS

Я не говорю здесь о второстепенных визуальных настройках наподобие смены цвета заголовков или задавания отступа абзаца. При использовании всех возможностей CSS, таблицы стилей являются надежным и мощным инструментом дизайна. Мои глаза открылись благодаря многообразию и богатству дизайнов проекта CSS Zen Garden (**www.csszengarden.com**). В далеком прошлом, когда разработчики колебались, стоит ли отказываться от макетов, созданных с помощью таблиц, сайт CSS Zen Garden Дэвида Ши показал, чего именно можно добиться, используя только CSS. Дэвид опубликовал HTML-документ и пригласил дизайнеров добавить их собственные таблицы стилей для создания визуального дизайна документа. Рис. 11.1 демонстрирует только небольшую часть моих любимых дизайнов. Все они используют *один и тот же* исходный HTML-документ.

Более того, документ не содержит ни одного элемента `img` (все эти изображения используются как фон). Но посмотрите, как замысловато и отлично от других выглядит каждая страница — и все при помощи таблиц стилей. Это стало доказательством возможностей, кроющихся в разделении CSS и HTML, в отделении представления от структуры. Сайт CSS Zen Garden больше не обновляется и теперь считается историческим документом поворотного момента в принятии веб-стандартов. Несмотря на его возраст, я все еще считаю, что это прекрасный небольшой урок, четко демонстрирующий, на что способны CSS.

Возможно, для создания CSS макетов, наподобие тех, что показаны на рис. 11.1, требуется много опыта. Потрясающие навыки в области графического дизайна также будут кстати (к сожалению, в комплекте с книгой они не продаются). Я показываю этот пример наперед, потому что хочу, чтобы вы осознавали весь потенциал дизайна, основанного на CSS, в особенности потому, что примеры в книгах для начинающих обычно весьма незамысловатые. Уделяйте время обучению, но держите цель перед собой.

CSS Zen Dragen
by Matthew Buchanan

Shaolin Yokobue
by Javier Cabrera

By the Pier
by Peter OngKelmscott

Organica Creativa
by Eduardo Cesario

Рис. 11.1. Эти страницы из проекта CSS Zen Garden используют один и тот же исходный HTML-документ, дизайн изменен исключительно при помощи CSS (материалы размещены с разрешения CSS Zen Garden и дизайнеров)

Как работают таблицы стилей

Это просто как раз-два-три!

1. Начните с документа, размеченного на языке HTML.
2. Напишите правила стилей, чтобы показать, какими вы хотели бы видеть определенные элементы.
3. Присоедините правила стилей к документу. Когда браузер отображает документ, он следует вашим правилам представления элементов (пока пользователь не применит какие-нибудь принудительные стили, но к данной теме мы обратимся позже).

С этим разобрались, хотя, конечно же, есть еще что-то. Давайте рассмотрим каждый из шагов немного подробнее.

1. Разметка документа

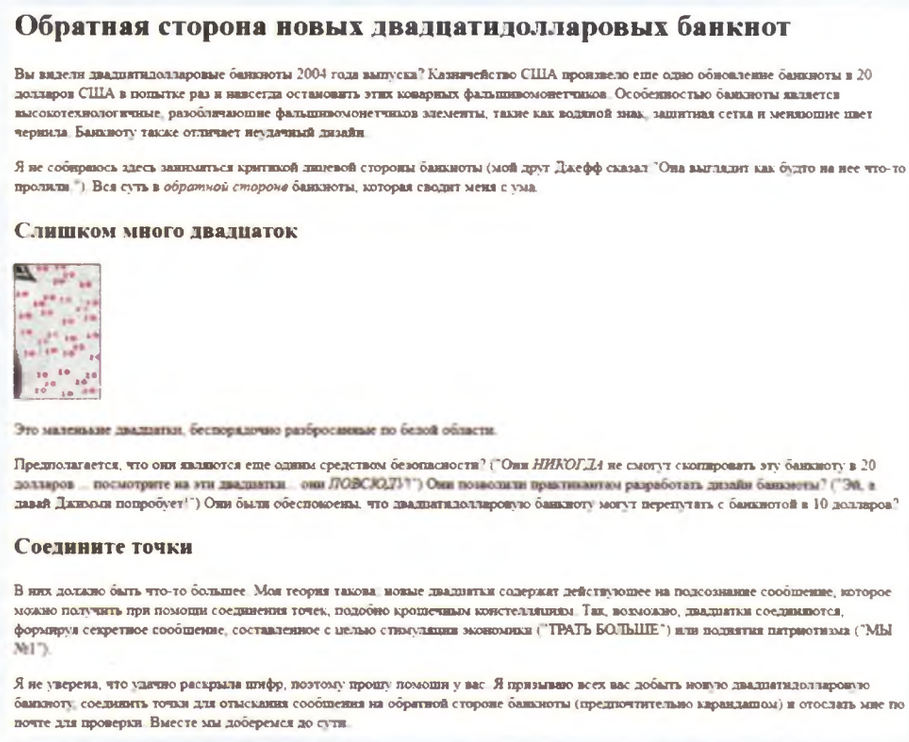
Вы многое знаете о разметке контента из предыдущих глав. Например, что важно выбрать HTML-элементы, которые точно описывают содержание контента. Также я говорила, что разметка создает структуру документа, иногда называемую *структурным слоем*, на который может быть наложен *слой представления*.

В этой и последующих главах вы увидите, что понимание структуры документа и отношений между элементами является главным в вашей работе как автора таблиц стилей.

Чтобы понять, как просто менять внешний вид документа при помощи таблиц стилей, попробуйте выполнить упражнение 11.1. Хорошая новость — я подготовила небольшой HTML-документ, чтобы вы могли поупражняться.

УПРАЖНЕНИЕ 11.1. ВАША ПЕРВАЯ ТАБЛИЦА СТИЛЕЙ

В этом упражнении мы добавим несколько простых стилей в короткую статью. HTML-документ *twenties.html* и связанное с ним изображение *twenty_20s.jpg* вы найдете на диске, прилагающемся к книге. Во-первых, откройте документ в браузере, чтобы увидеть, как он выглядит по умолчанию (приблизительно так, как показано на рис. 11.2). Вы также можете открыть документ в текстовом редакторе и следовать указаниям, когда это упражнение продолжится в следующем разделе.



Обратная сторона новых двадцатидолларовых банкнот

Вы видели двадцатидолларовые банкноты 2004 года выпуска? Казначейство США произвело еще одно обновление банкноты в 20 долларов США в попытке раз и навсегда остановить этих коварных фальшивомонетчиков. Особенностью банкноты является высокотехнологичные, разоблачающие фальшивомонетчиков элементы, такие как водяной знак, защитная сетка и меняющие цвет чернила. Банкноту также отличает неудачный дизайн.

Я не собираюсь здесь заниматься критикой лицевой стороны банкноты (мой друг Джефф сказал "Она выглядит как будто на нее что-то пролили."). Вся суть в *обратной стороне* банкноты, которая сводит меня с ума.

Слишком много двадцаток

Это маленькие двадцатки, беспорядочно разбросанные по белой области.

Предполагается, что они являются еще одним средством безопасности? ("Они *НИКОГДА* не смогут скопировать эту банкноту в 20 долларов ... посмотрите на эти двадцатки... они *ПОВСЮДУ*!") Они позволили практикантам разработать дизайн банкноты? ("Эй, а давай Джимми попробует!") Они были обеспокоены, что двадцатидолларовую банкноту могут перепутать с банкнотой в 10 долларов?

Соедините точки

В них должно быть что-то большее. Моя теория такова: новые двадцатки содержат действующее на подсознание сообщение, которое можно получить при помощи соединения точек, подобно крошечным коистелляциям. Так, возможно, двадцатки соединяются, формируя секретное сообщение, составленное с целью стимуляции экономики ("ТРАТЬ БОЛЬШЕ") или поднятия патриотизма ("МЫ №1").

Я не уверена, что удачно раскрыла шифр, поэтому прошу помощи у вас. Я призываю всех вас добыть новую двадцатидолларовую банкноту, соединить точки для отыскания сообщения на обратной стороне банкноты (предпочтительно карандашом) и отослать мне по почте для проверки. Вместе мы доберемся до сути.

Рис. 11.2. *Так выглядит статья без каких-либо инструкций таблицы стилей. И хотя мы не будем делать ее привлекательной, вы поймете, как работают стили*

2. Написание правил

Таблица стилей составлена из одной и более инструкций (называемых *правилами* или *наборами правил*), которые описывают, как элемент или группа элементов должны отображаться. Первый шаг в изучении CSS — это знакомство с частями правил. Как вы увидите, они понятны интуитивно. Каждое правило *выбирает* элемент и *объявляет*, как он должен выглядеть.

Следующий пример содержит два правила. Первое из них задает отображение всех элементов **h1** в документе зеленым цветом; второе — определяет, что шрифт абзацев должен быть мелким без засечек.

```
h1 { color: green; }

p { font-size: small; font-family: sans-serif; }
```

Согласно терминологии CSS существует две главные части правила — это *селектор*, устанавливающий элемент или элементы, на которые надо воздействовать, и *определение*, предоставляющее инструкции представления. Определение, в свою очередь, составлено из *свойства* (например, **color**) и его *значения* (**green**), разделенных двоеточием и пробелом. Одно или более определений размещаются внутри фигурных скобок, как показано на рис. 11.3.

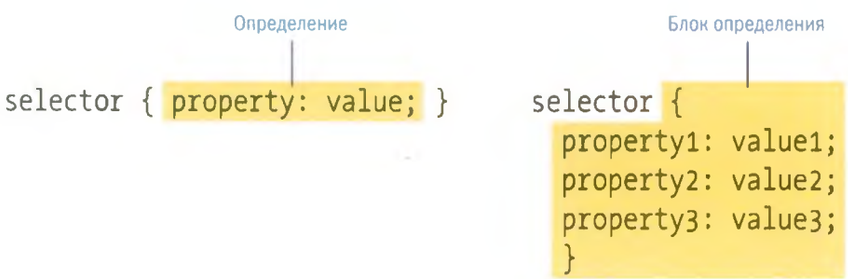


Рис. 11.3. Части правила таблицы стилей

Селекторы

В предыдущем небольшом примере элементы **h1** и **p** выступают как селекторы. Этот наиболее часто используемый тип селектора называется *селектор элемента*. Свойства, определенные для каждого селектора, будут соответственно применяться ко всем элементам **h1** и **p** в документе.

В следующих главах я представлю вам более сложные селекторы, которые вы можете использовать для целенаправленного выбора элементов, включая способы выбора групп элементов и элементов, которые появляются в отдельных контекстах.

Владение селекторами — то есть способностью выбрать самый подходящий тип селектора и искусно его использовать — это важный шаг в становлении профессионала CSS.

ПРИМЕЧАНИЕ

Шрифты *без засечек* (Sans-serif) не имеют ни одного мелкого росчерка (*засечки* (Serif)) на концах штрихов, и им свойственно смотреться гладкими и по-современному. Мы поговорим о шрифтах более подробно в главе 12.

Определения

Определение составлено из пары *свойство/значение*. Допустимо использовать несколько определений в одном правиле, например указанное выше правило для элемента **p** имеет свойства как **font-size**, так и **font-family**.

ПРИМЕЧАНИЕ

Технически, точка с запятой после последнего определения в блоке не требуется, но я рекомендую всегда ее ставить там. Это упростит более позднее добавление определений к правилу.

Каждое определение должно оканчиваться точкой с запятой для отделения его от следующего (см. примечание). Если вы пропустите этот знак препинания, то следующее за ним определение будет игнорироваться. Фигурные скобки и заключенные в них определения часто называются *блоком определения* (см. рис. 11.3).

Из-за того, что таблицы CSS игнорируют пробельные символы и переходы на новую строку внутри блока определения, верстальщики обычно пишут каждое определение в блоке на отдельной строке, как показано в следующем примере. Благодаря этому легче найти свойства, применяемые к селектору, и сказать, где заканчивается правило описания стиля.

```
p {
font-size: small;
font-family: sans-serif;
}
```

Обратите внимание, что на самом деле здесь ничего не поменялось — по-прежнему один набор фигурных скобок, точек с запятой после каждого определения и т. д. Единственное отличие — вставка переходов на новую строку и несколько пробелов для выравнивания.

Основой таблиц стилей является совокупность стандартных свойств, которые могут быть применены к выбранным элементам. Полная спецификация CSS определяет множество свойств для всего, начиная от отступа текста и заканчивая тем, как заголовки таблиц должны читаться вслух синтезатором речи.

Эта книга охватывает наиболее распространенные и лучше всего поддерживаемые свойства, которые вы можете начать применять прямо сейчас.

Значения зависят от свойства. Некоторые свойства преобразовывают размеры, некоторые — цвета, другие имеют предопределенный список ключевых слов.

При использовании свойства важно знать, какие значения оно принимает, однако во многих случаях достаточно и просто здравого смысла.

Перед тем как мы перейдем дальше, предлагаю немного попрактиковаться в написании правил стилей самостоятельно, продолжив упражнение 11.1.

Задавание измеряемых значений

При задавании измеряемых значений, единица измерения должна следовать сразу за числом, как, например, здесь:

```
{ margin: 2em; }
```

Вставка пробела перед единицей измерения станет причиной нефункционирующего свойства. Так писать **НЕПРАВИЛЬНО**:

```
{ margin: 2 em; }
```

Допускается пропускать указание единицы измерения для нулевых значений:

```
{ margin: 0; }
```

УПРАЖНЕНИЕ 11.1. ВАША ПЕРВАЯ ТАБЛИЦА СТИЛЕЙ (ПРОДОЛЖЕНИЕ)

Откройте файл *twenties.html* в текстовом редакторе. Вы обнаружите, что я установила элемент **style** в разделе **head** документа, чтобы вы указали там правила. Элемент **style** используется для вставки таблицы стилей в раздел заголовка HTML-документа.

Для начала мы добавим небольшую таблицу стилей, рассмотренную в этом разделе. Укажите следующие правила в документе так, как показано ниже.

```
<style type="text/css">
h1 {
color: green;
}
p {
font-size: small;
font-family: sans-serif;
}
</style>
```

Сохраните файл и взгляните на него в браузере. Вы должны заметить некоторые изменения (но если ваш браузер уже использует шрифт без засечек, видно будет только изменение размера шрифта). Если вы ничего не заметили, вернитесь и проверьте, на месте ли открывающая и закрывающая фигурные скобки, а также точки с запятой. Случайно пропустить эти символы легко, а это является причиной того, что таблица стилей не работает.

Теперь мы будем вносить изменения и добавления в таблицу стилей, чтобы увидеть, как легко писать правила, и наблюдать результаты изменений. Попытайтесь выполнить следующее (помните, что вам надо сохранять документ после каждого изменения, чтобы они были видны, когда вы обновляете документ в браузере).

- Задайте элементу **h1** серый цвет (**grey**) и взгляните на него в браузере. Затем сделайте его синим (**blue**), и наконец, красным (**red**). (Полный список доступных оттенков приведен в главе 13.)
- Добавьте новое правило, которое сделает красным также элемент **h2**.
- Добавьте левое поле шириной 100 пикселов к элементам абзаца (**p**), используя следующее определение:

```
margin-left: 100px;
```

- Помните, что вы можете добавить это новое определение к существующему правилу для элементов **p**.
- Добавьте также левое поле шириной 100 пикселов к заголовкам **h2**.
- Добавьте красную нижнюю границу шириной 1 пиксел элементу **h1**, используя определение:

```
border-bottom: 1px solid red;
```

- Переместите изображение к правому краю и сделайте так, чтобы текст обтекал изображение при помощи свойства **float**. Сокращенная запись свойства **margin**, показанная в этом правиле, добавляет пространство в 0 пикселов сверху и снизу изображения и пространство в 12 пикселов слева и справа от изображения (значения заданы способом, рассмотренным в главе 14).

```
img {
float: right;
margin: 0 12px;
}
```

Когда вы все выполните, документ должен выглядеть приблизительно как показано на рис. 11.4.

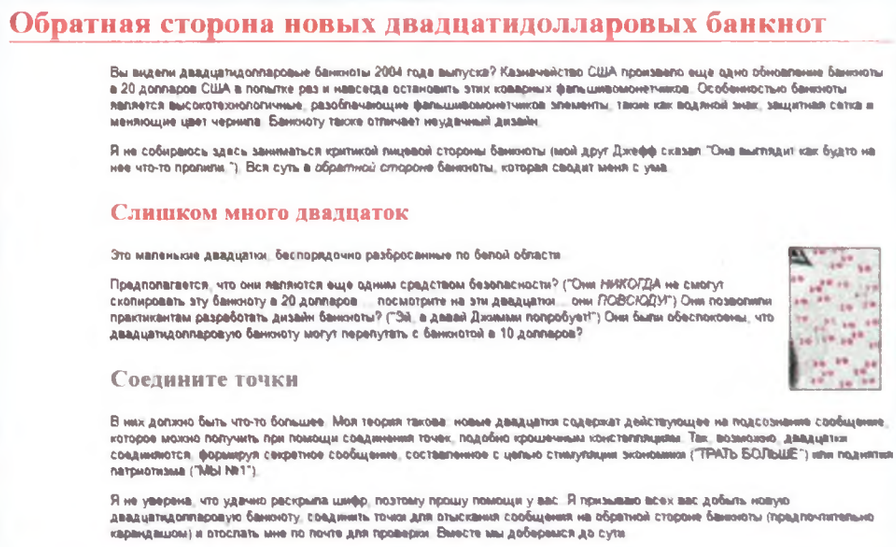


Рис. 11.4. *Страница после добавления небольшой таблицы стилей*

3. Присоединение таблиц к документу

В предыдущем упражнении мы встроили таблицу стилей напрямую в HTML-документ, используя элемент стилей **style**. Это только один из трех способов применить информацию о стиле к HTML-документу. Скоро вы сможете опробовать каждый из них, но полезно иметь общее представление о методах и терминологии уже сейчас.

Внешние таблицы стилей. Внешняя таблица стилей — это отдельный текстовый (и только текстовый) документ, который содержит ряд правил стилей. Он должен иметь имя с расширением *.css*. Документ с расширением *.css* связывается или импортируется в один или более HTML-документ (мы рассмотрим, как это делать в главе 13). Таким образом, все файлы веб-сайта могут совместно использовать одну и ту

же таблицу. Это наиболее действенный и предпочтительный метод для присоединения таблиц стилей к содержимому.

Глобальные таблицы стилей. С этим типом мы работали в упражнении. Такая таблица размещается в документе при помощи элемента **style** и ее правила применяются только в этом документе. Элемент **style** *должен* быть расположен в разделе **head** документа. Этот пример также включает комментарий (см. врезку «Комментарии в таблицах стилей»).

```
<head>
<title>Здесь требуется написать заголовок документа</title>
<style type="text/css">
/* здесь находятся правила */
</style>
</head>
```

Встроенные таблицы стилей. Вы можете применять свойства и значения к одному элементу, используя в нем атрибут **style**, как показано ниже:

```
<h1 style="color: red">Введение</h1>
```

Чтобы добавить несколько свойств, разделите их точкой с запятой:

```
<h1 style="color: red; margin-top: 2em">Введение</h1>
```

Встроенные стили применяются только к отдельному элементу, в котором они появляются. Следует избегать применения таких стилей, пока не станет совершенно необходимо переписать стили из глобальной или внешней таблицы стилей. Встроенные стили вызывают проблемы, по-

Комментарии в таблицах стилей

Иногда полезно оставлять себе или коллегам комментарии в таблице стилей. Язык CSS имеет свой собственный синтаксис комментариев, показанный здесь:

```
/* здесь находятся комментарии */
```

Текст, который содержится между символами **/*** и ***/**, будет игнорироваться при анализе таблицы стилей, а значит, вы можете оставлять комментарии в любом месте таблицы стилей, даже внутри правила.

```
body { font-size: small;
/* font-size:large; */ }
```

Одно из применений комментариев — для пометки разделов таблицы стилей, чтобы их было проще найти в дальнейшем, например:

```
/* Стили макета */
```

или

```
/* Стили нижнего колонтитула */
```

ПРИМЕЧАНИЕ

В HTML 4.01 и XHTML 1.0/1.1 элемент **style** должен был содержать атрибут **type**, идентифицирующий контент данного элемента.

```
<style type="text/css">
```

В HTML5 атрибут **type** больше не требуется.

Элемент **style** может также включать атрибут **media**, используемый для определения конкретного устройства вывода, например, экран, печатное или портативное устройство. См. также главу 13.

УПРАЖНЕНИЕ 11.2. ПРИМЕНЕНИЕ ВСТРОЕННЫХ СТИЛЕЙ

Откройте файл *twenties.html* независимо от того, в каком состоянии вы его оставили в упражнении 11.1. Если вы выполнили упражнение до конца, то получили правило, применяющее красный цвет к элементам **h2**.

Теперь напишите встроенный стиль, который делает второй элемент **h2** серым. Мы сделаем это прямо в открывающем теге, используя атрибут **style**, как показано ниже:

```
<h2 style="color: purple">Соедините точки</h2>
```

Сохраните файл и откройте его в браузере. Теперь тот заголовок становится серым, заменяя любой установленный ранее цвет. Другой заголовок второго уровня не изменился.

тому что они вставляют информацию о представлении в структурную разметку. Также они затрудняют внесение изменений, потому что надо выискивать каждый атрибут **style** в исходном коде.

Упражнение 11.2 позволяет написать встроенный стиль и увидеть, как он работает. Мы не будем больше применять встроенные стили в силу перечисленных выше причин.

Важные концепции

Существует несколько важных концепций, которые необходимо понять, чтобы разобраться, как работают каскадные таблицы стилей. Я собираюсь бегло ознакомить вас с этими принципами сейчас, чтобы не пришлось потом отступать от обзора свойств стилей. Безусловно, каждая из этих концепций будет рассматриваться и поясняться более подробно в следующих главах.

Наследование

У вас глаза такого же цвета, как и у ваших родителей? Вы получили их цвет волос? Их уникальную улыбку? Подобно тому, как родители передают по наследству характерные черты своим детям, элементы HTML передают определенные свойства стилей содержащимся в них элементам. Обратите внимание, что в упражнении 11.1, когда мы придали стиль элементам **p** в виде мелкого шрифта без засечек, элемент **em** во втором абзаце также стал отображаться мелким шрифтом без засечек, даже несмотря на то, что мы не создавали правила специально для него (рис. 11.5). Это потому, что элемент *унаследовал* стили от абзаца, в котором он находится.

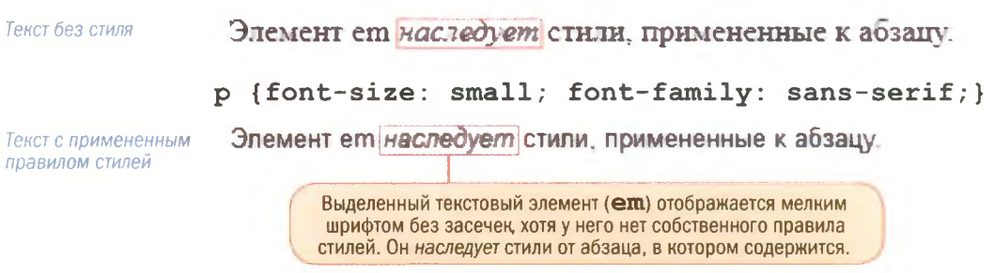


Рис. 11.5. Элемент **em** *наследует стили, примененные к абзацу*

Структура документа

Здесь важно понимать структуру вашего документа. Как я уже ранее отмечала, HTML-документы имеют неявную структуру или иерархию. Например образец статьи, с которой мы упражнялись, имеет корневой элемент **html**, который содержит раздел заголовка (**head**) и тело (**body**) страницы, а **body** содержит элементы заголовка и абзаца. Несколько абзацев, в свою очередь, содержат встроенные элементы, такие как изо-

бражения (`img`) и выделенный текст (`em`). Вы можете наглядно представить структуру перевернутого дерева, разветвляющегося от корня, как показано на рис. 11.6.

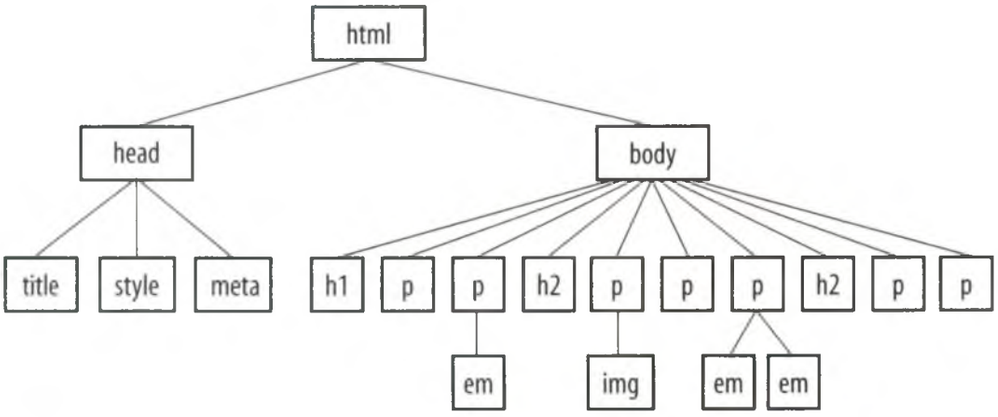


Рис. 11.6. *Древовидная структура документа twenties.html*

Родительские и дочерние элементы

Дерево документа становится генеалогическим, когда появляются направления связей между элементами. Все элементы, содержащиеся внутри исходного, называются *потомками*. Например все элементы `h1`, `h2`, `p`, `em` и `img` документа на рис. 11.6 — потомки элемента `body`.

Элемент, который полностью содержится внутри другого (без промежуточных уровней иерархии), называют *дочерним элементом (ребенком)*. И наоборот, содержащий его элемент называется *родительским*. Например, элемент `em` является дочерним по отношению к `p`, а элемент `p` — родительским по отношению к `em`.

Все элементы, лежащие выше конкретного элемента по иерархии — его *предшествующие элементы (предки)*. Два элемента с одним и тем же «родителем» — *элементы одного уровня (сестры)*. Все это может казаться оторванным от практики, но пригодится при написании CSS-селекторов.

Перемещение по дереву сверху вниз

Когда вы пишете стиль, основанный на шрифтах, используя элемент `p` в качестве селектора, правило применяется ко всем абзацам документа так же, как и ко встроенным текстовым элементам, которые они содержат. Ранее на рис. 11.5 мы видели доказательство наследования элементом `em` свойства стилей, примененного к его родителю (`p`). Рис. 11.7 демонстрирует, что происходит, на основе схемы структуры документа. Обратите внимание, что элемент `img` исключен, так как свойства, связанные со шрифтом, не применяются к изображениям.

Обратите внимание, я сказала *определенные* свойства наследуются. Это важно отметить, потому что некоторые свойства таблицы стилей наследуются, а некоторые нет. Свойства, связанные с форматированием текста, — размер шрифта, цвета, стилей и т. д. — передаются по наследству.

СОВЕТ ПО CSS

Когда вы изучаете новое свойство, хорошо бы отмечать, является ли оно наследником. Наследование указывается для каждого свойства, перечисленного в этой книге. Как правило, наследование отвечает вашим ожиданиям.

А вот такие, как границы, поля, фон и т. д., которые оказывают воздействие на очерченную границей область вокруг элемента, по наследству не передаются. Если подумать, то это не лишено смысла. Например, если вы помещаете границу вокруг абзаца, то вряд ли захотите, чтобы граница была также вокруг каждого встроенного элемента (такого как **em**, **strong** или **a**), содержащегося в нем.

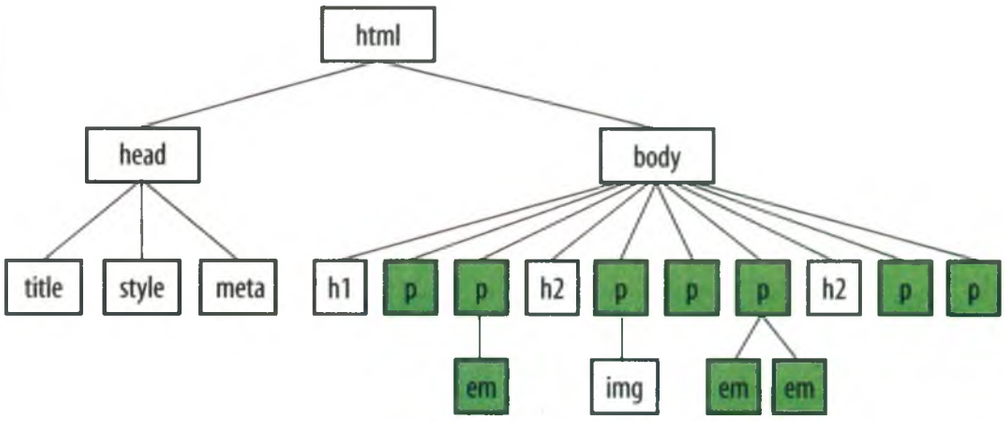


Рис. 11.7. *Определенные свойства, примененные к элементам* **p**, *наследуются их дочерними элементами*

ПРЕДУПРЕЖДЕНИЕ

Таблица стилей браузера может замещать стили, заданные в элементе **body**, так что будьте готовы к появлению неожиданных стилей.

Можно выгодно использовать наследование при написании таблиц стилей. Например, если вы хотите, чтобы все текстовые элементы внешне отображались шрифтом Verdana, можете написать отдельные правила стилей для каждого элемента в документе и установить свойство **font-face** в значение Verdana. *Лучшим* способом будет написать одно правило стилей, которое применяет свойство **font-face** к содержимому элемента **body**, и позволить всем текстовым элементам, входящим в **body**, унаследовать этот стиль (рис. 11.8).

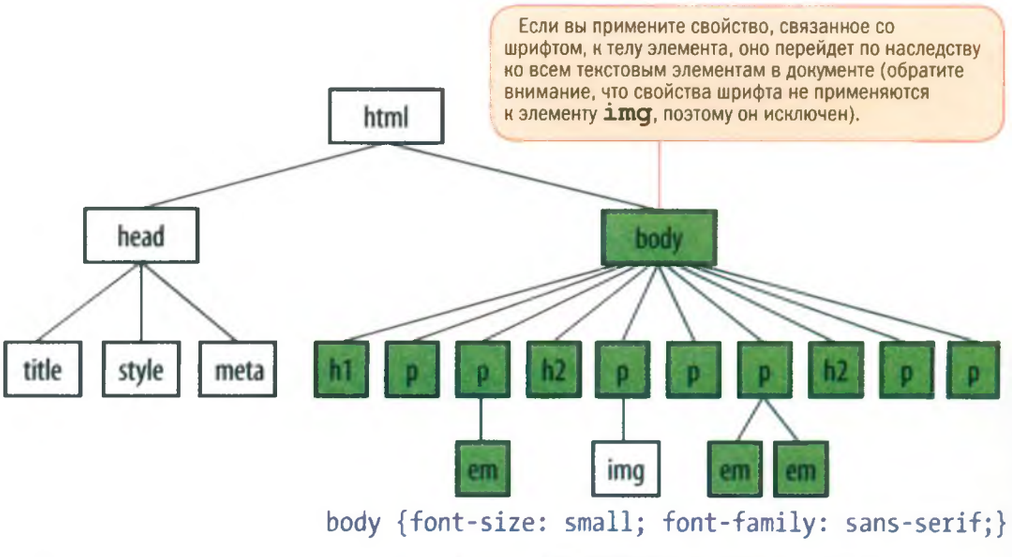


Рис. 11.8. *Все элементы в документе наследуют определенные свойства, примененные к элементу* **body**

Любое свойство, примененное к конкретному элементу, заменит у него значения, унаследованные для этого свойства. Возвращаясь к примеру со статьей, мы могли определить, что элемент **em** должен отображаться шрифтом с засечками, и это отменило бы унаследованный параметр шрифта без засечек.

Конфликтующие стили: каскад

Никогда не задумывались, почему таблицы стилей называются «каскадными»? CSS позволяет вам применять несколько таблиц стилей к одному и тому же документу, что означает неизбежное возникновение конфликтов. Например, что должен делать браузер, если импортированная в документ таблица стилей устанавливает красный цвет для элемента **h1**, а глобальная таблица стилей имеет правило, делающее элементы **h1** фиолетовыми?

Люди, которые разработали спецификацию таблиц стилей, предвидели эту проблему и разработали иерархическую систему, присваивающую разный вес разным источникам информации о стиле. *Каскад* имеет отношение к тому, что происходит, когда несколько источников информации о стиле конкурируют за управление элементами на странице: информация о стиле передается вниз по дереву до тех пор, пока не заменится командой стилей с большим весом.

Например, если вы не применяете никакой информации о стиле к веб-странице, она будет отображена в соответствии с внутренней таблицей стилей браузера (это называется отображением по умолчанию, или, согласно консорциуму Всемирной паутины W3C — *таблицей стилей пользовательского агента*). Однако если верстальщик веб-страницы задал для документа таблицу стилей (*таблицу стилей верстальщика*), то она имеет больший вес и отменяет стили браузера. Единственное исключение — если пользователь пометил стиль как «важный», в этом случае данный стиль перевешивает все остальные (см. врезку «Назначение приоритета»).

Источник таблицы стилей — это иерархия, определяющая, какой стиль является приоритетным. Как вы уже уяснили, существуют три способа присоединения информации о стиле к исходному документу, и они также имеют каскадный порядок. Вообще, чем ближе таблица стилей к содержимому, тем больший вес ей присваивается. Глобальные таблицы стилей, которые появляются напрямую в документе в элементе `style`, имеют больший вес, чем внешние таблицы стилей. В примере, начинающем этот раздел, элементы `h1` в конечном счете окрашивались фиолетовым цветом, согласно глобальной таблице стилей, а не красным, как определялось во внешнем файле *.css*, имеющем меньший вес. Встроенные стили имеют больший вес, чем глобальные таблицы стилей, потому что они ближе к контенту. Именно этот эффект мы наблюдали в упражнении 11.2.

Для предотвращения замены конкретного правила, вы можете назначить ему «важность» при помощи индикатора **!important**, как объясняется во врезке «Назначение приоритета».

Врезка «Иерархия таблиц стилей» предоставляет обзор каскадного порядка от общего к частному.

Специфичность

Когда два правила в одной таблице стилей вступают в конфликт, для определения приоритетного используется селектор соответствующего типа.

Как только подходящая таблица стилей выбрана, в ней все еще могут быть конфликты; поэтому каскадирование продолжается на уровне правил. Когда два правила в одной таблице стилей конфликтуют, для определения победителя используется селектор соответствующего типа. Чем специфичнее селектор, тем больший вес ему присваивается для замены конфликтующих определений.

Пока еще рано рассматривать специфичность, потому что мы просмотрели только один тип селектора (к тому же наименее специфичный). Сейчас просто возьмите на заметку термин *специфичность* и принцип замены одних селекторов другими. Мы вернемся к ним в главе 12, когда в вашем арсенале будет больше типов селекторов.

Назначение приоритета

Если вы хотите, чтобы правило не было заменено более поздним конфликтующим, вставьте индикатор **!important** сразу после значения свойства перед точкой с запятой. Например, чтобы абзац отображался всегда синим цветом, используйте правило:

```
p {color: blue !important;}
```

Даже если позднее браузер встретит в документе встроенный стиль (который должен заменить действующую по всему документу таблицу стилей) вроде этого:

```
<p style="color: red">
```

тот абзац по-прежнему будет отображаться синим цветом, потому что правило с индикатором **!important** не может быть заменено в таблице стилей верстальщика.

Единственный вариант, при котором состоится замена: если конфликтующее правило в пользовательской таблице стилей также было помечено как **!important**. Этого следует избегать, чтобы специальные требования пользователей, такие как крупный шрифт для слабовидящих, никогда не заменялись.

Основываясь на предыдущих примерах, если таблица стилей читателя включает в себя правило:

```
p {color: black;}
```

текст останется синим, потому что все стили верстальщика (даже те, которые не помечены индикатором **!important**) имеют преимущество над стилями читателя. Однако если конфликтующий пользовательский стиль помечен индикатором **!important**, вроде этого:

```
p {color: black !important;}
```

абзацы станут отображаться черным цветом, и их нельзя будет заменить никаким стилем, предоставленным верстальщиком.

Иерархия таблиц стилей

Информация о стиле может поступать от разных источников, перечисленных ниже, от общего к частному. Пункты, находящиеся ниже по списку, отменяют пункты, находящиеся выше:

- Настройки по умолчанию браузера
- Пользовательские настройки стилей (установленные в браузере как «таблица стилей клиента»)
- Связанная внешняя таблица стилей (добавленная при помощи элемента **link**)
- Импортируемые таблицы стилей (добавленные при помощи функции **@import**)
- Глобальные таблицы стилей (добавляемые при помощи элемента **style**)
- Информация о встроенном стиле (добавленная в открывающий тег при помощи атрибута **style**)
- Любое правило стилей, помеченное верстальщиком индикатором **!important**
- Любое правило, помеченное пользователем индикатором **!important**

Очередность правил

Наконец, если возникают конфликты внутри правил стилей, имеющих одинаковый вес, побеждает то, которое указано последним. Например возьмем три правила:

```
<style type="text/css">
p { color: red; }
p { color: blue; }
p { color: green; }
</style>
```

В данном сценарии текст абзаца будет отображаться зеленым цветом из-за последнего правила в этой таблице стилей. Другими словами, то правило, которое располагается ближе всего к контенту документа, заменяет те, что определены до него. То же самое происходит, когда конфликтующие стили встречаются в одном наборе определений:

```
<style>
p {    color: red;
color: blue;
color: green; }
</style>
```

В результате получится зеленый цвет, потому что последнее определение отменяет два предыдущих. Когда вы имеете дело со сложными свойствами, легко случайно отменить предыдущие определения правила, поэтому важно помнить о таком их поведении.

ПРИМЕЧАНИЕ

Правило «выигрывает перечисленный последним» также применяется и в других контекстах в CSS. Например внешние таблицы стилей, перечисленные позднее в исходном коде, имеют преимущество над теми, что перечислены до них.

Блочная модель

Поскольку мы говорим о «важных принципах CSS», уместно ввести понятие *блочной модели* — краеугольного камня модели визуального форматирования CSS. Самый простой способ понять блочную модель — это представить, что браузеры видят каждый элемент на странице (как блочный, так и встроенный) заключенным в небольшой прямоугольный *блок*. Вы можете применять к этим блокам свойства, такие как границы, поля, отступы и фон, и даже перемещать на странице.

Мы изучим детали блочной модели в главе 14, но сейчас для вас будет полезно получить о ней общее представление, так как в следующих двух главах мы будем рассматривать текст и фон.

Чтобы увидеть элементы приблизительно так, как их видит браузер, я написала правила стилей, которые добавляют границы вокруг каждого элемента контента в статье-примере.

```
h1 { border: 1px solid blue; }
h2 { border: 1px solid blue; }
p { border: 1px solid blue; }
em { border: 1px solid blue; }
img { border: 1px solid blue; }
```

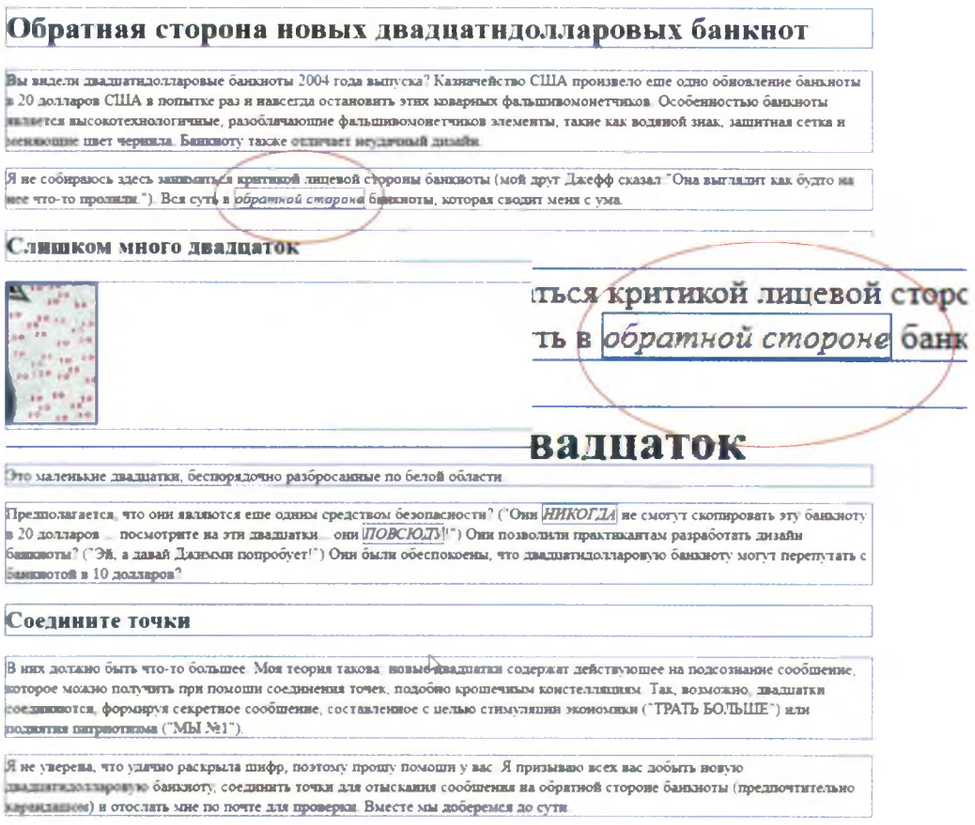


Рис. 11.9. *Правила всех элементов воспроизводят блоки этих элементов*

Результаты показаны на рис. 11.9. Границы воспроизводят форму каждого блочного элемента. Также блоки есть и вокруг встроенных элементов

(`em` и `img`). Обратите внимание, что границы блочных элементов растягиваются по доступной ширине окна браузера, что является их основным свойством при нормальном потоке документа. Встроенные блоки окружают только те символы или изображение, которые они содержат.

Сгруппированные селекторы

Это хорошая возможность показать, как удобнее сокращать правила стилей. Если вам когда-нибудь понадобится применить одно и то же свойство к нескольким элементам, можете сгруппировать селекторы в одно правило, разделив их запятыми. Ниже приведено одно такое правило, которое выполняет то же действие, что и пять правил, перечисленных ранее. Подобное группирование делает будущий процесс редактирования более эффективным и сокращает размер файла.

```
h1, h2, p, em, img { border: 1px solid blue; }
```

Теперь в вашем инструментарии есть два типа селекторов: простой селектор элементов и сгруппированные селекторы.

Краткая история CSS

Первая официальная версия CSS (*CSS Level 1 Recommendation*, также известная под названием *CSS1*) была официально выпущена в 1996 году и включала в себя свойства для добавления шрифта, цвета и инструкции разрядки элементов страницы. К сожалению, неполная поддержка браузерами препятствовала широкому признанию CSS в течение нескольких лет.

Версия *CSS Level 2* (*CSS2*) вышла в 1998 году. В ней в значительной мере были добавлены свойства позиционирования, что позволило использовать CSS для разметки страниц. Стандарт также вводил стили для других типов устройств (таких как устройства печати, портативные устройства или синтезаторы речи) и более сложные методы выбора элементов стилизации. *CSS Level 2, Revision 1* (*CSS2.1*) внесла незначительные поправки в версию CSS2 и стала полновесным стандартом в 2011 году.

Версия *CSS Level 3* (*CSS3*), в отличие от предыдущих, делится на множество отдельных модулей, каждый из которых посвящен какой-либо функции, например анимации, многоколончатым макетам или границам. В то время как некоторые модули стандартизированы, другие остаются экспериментальными. Таким образом, разработчики браузеров могут приступить к реализации одной из функций, не дожидаясь, пока будет «готова» вся спецификация. На самом деле многие разработчики используют улучшенные функции CSS3 (даже если они не повсеместно поддерживаются), когда можно применить запасной вариант и контент при этом не теряется. Их удобно использовать для «украшения» стабильного дизайна (другими словами, как улучшение).

Будьте в курсе последних новостей о разработках консорциума Всемирной паутины в области CSS, посещая сайт **www.w3.org/Style/CSS/current-work**.

Контрольный вопрос

Вы можете догадаться, почему я просто не добавила элементу `body` свойство `border` и не позволила ему перейти по наследству ко всем элементам в сгруппированном селекторе?

Ответ

Потому что `border` является одним из тех свойств, которые не наследуются, как отмечалось ранее.

Дальнейшее изучение CSS

Эта глава охватила все основы каскадных таблиц стилей, включая синтаксис правил, способы применения стилей к документу и важные принципы наследования, каскадирование и блочную модель. Таблицы стилей больше не должны быть загадкой, и в дальнейшем мы станем работать, добавляя свойства и селекторы в ваш арсенал, а также рассмотрим подробнее принципы, введенные в этой главе.

CSS — обширная тема, далеко выходящая за рамки данного издания. Книжные магазины и Всемирная паутина перегружены информацией о таблицах стилей для пользователей с навыками всех уровней. Я собрала материал из перечня источников, которые сочла наиболее полезными в процессе моего обучения. Кроме того, в разделе «Инструменты CSS» я предоставила список широко распространенных инструментов, которые помогают при написании таблиц стилей.

Список литературы

Дефицита в хороших книгах по CSS нет, но ниже указаны те, по которым училась я сама, и поэтому могу их рекомендовать.

- Дакетт Дж. «Основы веб-программирования с использованием HTML, XHTML и CSS» (Эксмо, 2010)
- Макдональд М. «Создание Web-сайтов. Основное руководство» (Эксмо, 2010)
- Макфарланд Д. «Большая книга CSS» (Питер, 2012)
- Фельке-Моррис Т. «Большая книга веб-дизайна» (Эксмо, 2012)

Веб-ресурсы

Указанные сайты будут хорошим подспорьем при изучении таблиц стилей.

- Консорциум W3C (**www.w3.org/Style/CSS**). Консорциум Всемирной паутины следит за развитием технологий Всемирной паутины, включая CSS.
- Справочник по CSS (**css.manual.ru**). Справочник-руководство по CSS, базируется на спецификации CSS 2.1.
- Учебник CSS (**ru.html.net/tutorials/css/**). Учебник CSS на русском языке, рекомендованный консорциумом Всемирной паутины.

Инструменты CSS

Я привела пару инструментов, которые могу рекомендовать лично.

Расширение Web Developer

Веб-дизайнеры в восторге от расширения Web Developer, написанного Крисом Педериком. Оно добавляет в браузер панель с инструментами, позволяющими манипулировать любой страницей в окне и анализировать ее. Вы можете редактировать таблицу стилей для страницы, которую просматриваете, а также получать информацию о HTML и используемых графических объектах. Кроме того, расширение проверяет валидность CSS, HTML и доступность страницы. Загрузить его можно по адресу **chrispederick.com/work/web-developer/** или **addons.mozilla.org/ru/firefox/addon/web-developer/**.

Обратите внимание, что в браузере Safari предлагается похожий встроенный инспектор (выберите команду меню **Разработка** ⇒ **Показать веб-инспектор** (Develop ⇒ Show Web Inspector).

Программы для верстки веб-страниц

Современные WYSIWYG-программы для верстки веб-страниц, такие как Adobe Dreamweaver и Microsoft Expression Web, могут быть настроены на автоматическое написание таблиц стилей, позволяя вам заниматься дизайном страницы. Но этот способ не всегда рациональный (например, программы зачастую злоупотребляют атрибутом `class` для создания правил стилей). Однако они могут ускорить процесс верстки на начальном этапе, предоставив каркас таблицы стилей, который вы потом можете изменить или наполнить вручную.

ГЛАВА 12

ФОРМАТИРОВАНИЕ ТЕКСТА (ВКЛЮЧАЯ СЕЛЕКТОРЫ)

По прочтении этой главы вы узнаете *пятнадцать* новых свойств CSS, используемых для форматирования текста. Попутно вы также изучите, как с большими возможностями использовать селекторы, имеющие конкретное имя идентификатора или класса, для целенаправленного выбора элементов из определенного контекста.

Природа Всемирной паутины специфична. Никак нельзя узнать наверняка, будет ли доступен шрифт, который вы используете, или насколько крупным или мелким окажется кегль шрифта в браузере пользователя. Далее мы рассмотрим лучшие решения данных проблем.

На протяжении всей этой главы мы будем совершенствовать меню бистро «Черный гусь», аналогичное тому, которое мы разметили ранее в главе 5. Я призываю вас выполнять упражнения, чтобы получить глубокое понимание того, как работают свойства. На рис. 12.1 показано, как выглядело меню, когда мы его видели в последний раз, и как оно будет выглядеть, когда мы закончим. Это не шедевр, поскольку мы изучаем CSS только поверхностно, но, по крайней мере, текст смотрится более изящно.

В этой главе

- Свойства, связанные со шрифтом
- Веб-шрифты и их стеки
- Настройки форматирования текста, такие как высота строки, отступы и выравнивание
- Форматирование шрифта: подчеркивание, изменение регистра, добавление тени и т. п.
- Интервалы между буквами и словами
- Селекторы потомков, идентификаторов и классов
- Основы специфичности

Свойства шрифта

Когда я работаю над дизайном текстового документа (особенно для вывода на печать, но также и для Всемирной паутины), первое, что я делаю, это задаю шрифт. В CSS шрифт задается при помощи набора связанных с ним свойств для указания вида шрифта, размера, насыщенности и начертания. Также есть сокращенное свойство, которое позволит вам задавать все атрибуты за один раз.

НА ЗАМЕТКУ

Связанные со шрифтом свойства:

```
font-family
font-size
font-weight
font-style
font-variant
font
```

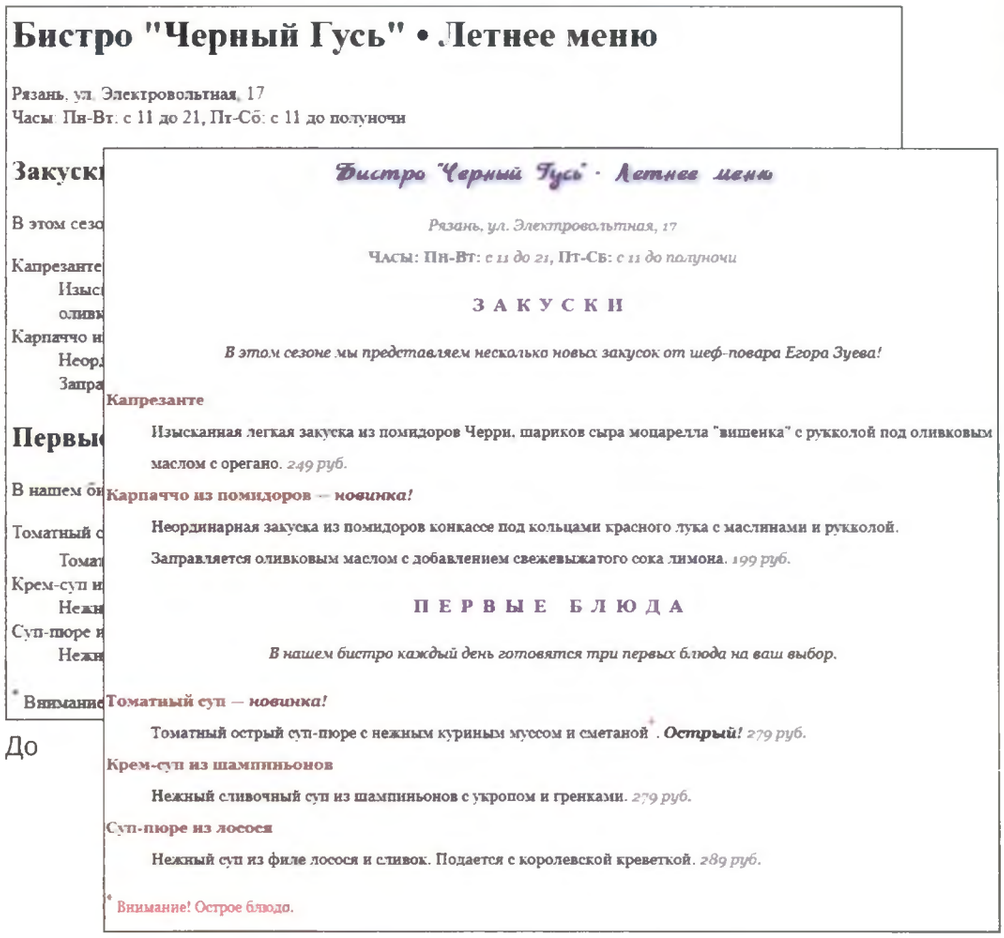


Рис. 12.1. *Вид «до» и «после» меню бистро «Черный Гусь», над которым мы будем работать в этой главе*

Указание имени шрифта

Приступая к работе, хорошо первым делом выбрать гарнитуру шрифта, или, как это называется в CSS, *семейство шрифтов*. Давайте начнем с использования свойства **`font-family`** и его значений.

`font-family`

Принимаемые значения: *один или более шрифт или имена семейств типовых шрифтов, разделенных запятыми* | **`inherit`**

Значение по умолчанию: *зависит от браузера*

Применение: *ко всем элементам*

Наследование: *да*

Используйте свойство **`font-family`** для задания шрифта или списка шрифтов (*стека шрифтов*) по имени, как показано в примерах ниже.

```
body { font-family: Arial; }
var { font-family: Courier, monospace; }
p { font-family: "Trebuchet MS", Verdana, sans-serif; }
```

Пара слов о перечислениях свойств

Каждое новое свойство, перечисляемое в этой книге, сопровождается информацией о том, как оно себя ведет и как его использовать. Здесь приведено краткое объяснение каждой части описания, указываемой при перечислении свойств.

Принимаемые значения

Под принимаемыми значениями понимаются допустимые значения свойства. Зарезервированные значения выделяются моноширинным шрифтом (например, **`small`**, **`italic`** или **`small-caps`**) и должны печататься в точности как указано.

Значение по умолчанию

Понимается значение, которое будет использоваться для свойства по умолчанию, если не задано никакое другое значение. Обратите внимание, что браузер использует таблицу стилей со значениями, которые могут отличаться от значений по умолчанию в CSS.

Применение

Некоторые свойства применяются только к определенным типам элементов, таким как блочные или табличные элементы.

Наследование

Показывает, будет ли свойство передаваться потомкам выбранного элемента. См. главу 11 для объяснения принципов наследования.

Вы, должно быть, спрашиваете: «Зачем задавать больше одного шрифта?» Хороший вопрос, и он приводит нас к одной из проблем при задавании шрифтов для веб-страниц.

Далее приведены некоторые важные требования синтаксиса:

- Все имена шрифтов, за исключением семейств типовых шрифтов, должны быть написаны с большой буквы. Например «Arial» вместо «arial».
- Используйте запятые для разделения нескольких имен шрифтов, как показано во втором и третьем примерах.
- Обратите внимание, что имена шрифтов, которые содержат символ пробела (такие как Trebuchet MS в третьем примере), должны быть заключены в кавычки.

Ограничения шрифтов

Браузеры ограничены использованием тех шрифтов, к которым у них есть доступ. Обычно это означает, что данные шрифты уже установлены на компьютерах пользователей. Однако в 2012 году браузеры начали активно поддерживать встроенные веб-шрифты с помощью правила CSS **`@font-face`** так, что веб-дизайнеры смогли предлагать собственные шрифты. Подробнее об этом читайте во врезке «Веб-шрифты».

Веб-шрифты

Возможность предоставлять собственные шрифты для использования на веб-странице существовала примерно с 1998 года, но никогда не была реально осуществимой из-за несоответствий браузеров. К счастью, ситуация изменилась и в настоящее время веб-шрифты — вполне жизнеспособный вариант. Всемирная паутина никогда не выглядела лучше!

О веб-шрифтах можно говорить долго, поэтому данная врезка представляет собой лишь введение с указанием основных моментов, начиная с трудностей.

Почему ситуация изменилась только сейчас?

На пути добавления шрифтов на веб-страницы раньше стояли два основных препятствия. Первое заключалось в том, что разные браузеры поддерживают разные форматы шрифтов. Большинство шрифтов представлены в формате OpenType (OTF) илиTrueType (TTF) формате, но Internet Explorer принимает только собственные шрифты Embedded Open Type (CPB).

Теперь появился иной стандарт упаковки шрифтов для доставки на веб-страницы, который применяют производители всех браузеров, даже Internet Explorer. Новый открытый формат веб-шрифтов (*Web Open Font Format, WOFF*) — это контейнер, упаковывающий их для доставки во Всемирную паутину.

Теперь, когда браузер Internet Explorer начиная с версии 9 поддерживает WOFF, в один прекрасный день, возможно, нам больше ничего не понадобится. Тем не менее на момент написания книги, все еще необходимо было предоставлять один и тот же шрифт в нескольких различных форматах (подробнее об этом чуть ниже).

Другая проблема предоставления шрифтов на веб-страницы по-прежнему актуальна и заключается в том, что компании, создающие шрифты, обеспокоены, что те будут уязвимы на сервере и доступны для неконтролируемой загрузки. Создание шрифтов требует больших усилий, и они очень ценны. К большинству прилагаются лицензии, оговаривающие очень узкое использование их на ограниченном числе компьютеров и пункта «доступен для свободного использования» в этих лицензиях обычно нет.

Таким образом, чтобы ссылаться на веб-шрифт, вы должны использовать его на законных основаниях и предоставлять таким способом, который поддерживают все браузеры. Существует два основных способа предоставления шрифтов: разместить их на хостинге самостоятельно или прибегнуть к помощи службы веб-шрифтов. Мы рассмотрим оба варианта.

Самостоятельное размещение

При самостоятельном размещении вы находите нужный шрифт, выгружаете на сервер во всех необходимых форматах, и связываете его с вашей веб-страницей с помощью правила CSS `@font-face`.

Шаг 1: Поиск шрифта. Это может быть довольно сложно, поскольку лицензионное соглашение с конечным пользователем (End User License Agreement, EULA) практически для всех коммерческих шрифтов не распространяется на использование во Всемирной паутине. Обязательно приобретите дополнительную лицензию, если таковая имеется. Однако благодаря спросу некоторые компании позволяют использовать шрифты во Всемирной паутине, кроме того, постоянно растет количество шрифтов с открытым исходным кодом, которые можно использовать бесплатно. Сайт Fontspring (fontspring.com) Итана Данхэма — замечательное место, где несложно купить шрифты с веб-лицензией, которые затем вы сможете использовать на своем сайте или компьютере. Другой сайт Итана Данхэма, FontSquirrel (Fontsquirrel.com) — отличный источник шрифтов с открытым исходным кодом, которые можно бесплатно использовать в коммерческих целях.

Шаг 2: Сохранение шрифта в различных форматах. На момент написания книги, предоставление нескольких форматов — это норма. Существуют инструменты, преобразующие исходный шрифт в другие форматы, а также сервис, который выполнит за вас все необходимые действия со шрифтами — генератор шрифтов на сайте **www.fontsquirrel.com/fontface/generator**. Перейдите на эту страницу, выгрузите свой шрифт, и программа предоставит версии шрифта в форматах TTF, CPB, WOFF и SVG, а также код CSS, который необходим для обеспечения поддержки шрифта. Имейте в виду, что использовать сервис следует только для шрифта, использование которого во Всемирной паутине допускается (неважно, бесплатный это шрифт, шрифт с открытым исходным кодом или коммерческий). Кроме того, следует учитывать, что непосредственно от разработчика шрифтов вы получите более качественные версии шрифта, чем при использовании генератора.

Шаг 3: Выгрузка на сервер. Разработчики обычно хранят файлы шрифтов в одном каталоге с файлами CSS, но это дело вкуса. Если вы загружаете пакет с сайта FontSquirrel, не забудьте сохранить все файлы вместе, как они были расположены изначально.

Шаг 4: Верстка кода. Свяжите шрифт со своим сайтом с помощью правила **@font-face** в документе .css. Оно задает шрифту имя для свойства **font-family**, которое вы затем сможете указать в таблице стилей. В нем также перечислены расположения файлов шрифтов в различных форматах. Приведенный ниже кроссбраузерный пример кода был разработан Итаном Данхэмом для устранения ошибки в браузере Internet Explorer. Я рекомендую прочитать полный текст его статьи на странице **www.fontspring.com/blog/the-new-bulletproof-font-face-syntax.** См. также статью Пола Айриша на сайте **paulirish.com/2009/bulletproof-font-face-implementation-syntax/.**

```
@font-face {
font-family: 'Font_name';
src: url('myfont-webfont.eot?#iefix')
format('embedded-opentype'),
url('myfont-webfont.woff') format('woff'),
url('myfont-webfont.ttf')
format('truetype'),
url('myfont-webfont.svg#svgFontName')
format('svg');
}
```

Затем укажите принятое имя шрифта в правилах, касающихся шрифтов, например:

```
p {font-family: Имя_шрифта; }
```

Использование сервиса веб-шрифтов

Если все это кажется вам трудоемким, можете зарегистрироваться на одном из сервисов веб-шрифтов, который выполнит за вас всю грязную работу. За определенную плату вы получаете доступ к высококачественным шрифтам, а сервис решает вопросы лицензирования и защиты с компаниями-разработчиками шрифтов. Также сервисы обычно предоставляют интерфейс и инструменты, превращающие процесс вложения шрифтов в простое копирование и вставку.

У этих сервисов разные схемы оплаты. Некоторые взимают ежемесячные платежи, другие берут деньги за каждый шрифт. Иногда требуется оплачивать и пропускную способность канала. Обычно это многослойные тарифы, от бесплатных до нескольких сотен долларов в месяц.

Ниже приведены некоторые сервисы веб-шрифтов, популярные на момент написания книги, но рекомендуется выполнить поиск во Всемирной паутине, чтобы найти актуальные и выгодные предложения.

Google Web Fonts (www.google.com/webfonts)

Google Web Fonts — бесплатный сервис, предоставляющий доступ к сотням шрифтов с открытым исходным кодом, которые можно бесплатно использовать в коммерческих целях. Все, что вам нужно сделать — это выбрать шрифт, а затем скопировать и вставить код, который будет сгенерирован. Если ваш бюджет не распространяется на шрифты, а вы не слишком щепетильны в этом вопросе — данный вариант замечательно подойдет. Мы применим его в первом упражнении этой главы.

Adobe Typekit (www.typekit.com)

Typekit был первым сервисом веб-шрифтов, а теперь является частью корпорации Adobe.

Данный сервис использует JavaScript, чтобы связать шрифты с вашим сайтом, улучшая таким образом производительность и качество во всех браузерах.

Fonts.com (fonts.com)

Сервис Fonts.com может похвастаться самой объемной коллекцией шрифтов крупнейших компаний-разработчиков. Если вам нужен конкретный шрифт, скорее всего он здесь представлен.

К другим сервисам относятся WebINK (**www.extensis.com/en/WebINK**), Typotheque (**www.typotheque.com/webfonts**), Fonts Live (**www.fontslive.com**) и Fontdeck (**fontdeck.com**). Они отличаются количеством предлагаемых шрифтов и схемами оплаты, так что, возможно, вы решите посетить их все, чтобы приценяться.

Резюме

Вам решать, каким способом добавлять добавить шрифт на сайт. Если вам нравится полный контроль, хорошим решением будет разместить на хостинге собственный шрифт (законно, разумеется). Если нужен особенный шрифт, потому что на нем строится бренд вашего клиента, вы, вероятно, найдете его на одном из веб-сервисов, предлагающих шрифты за определенную цену. Если вы хотите поэкспериментировать с веб-шрифтами, сервис Google Web Fonts как раз подойдет.

Теперь у вас есть базовые знания по использованию веб-шрифтов. Ситуация может быстро меняться, поэтому не забудьте провести собственное исследование сервисов веб-шрифтов, когда будете готовы начать работу.

Но вернемся к правилу **font-family**. Даже если вы укажете в правиле стиля, что необходимо использовать шрифт Futura, в случае, когда браузер его не найдет (например, если шрифт Futura не установлен на компьютере пользователя или при загрузке веб-шрифта происходит сбой), вместо него будет использоваться шрифт браузера по умолчанию.

К счастью, CSS позволяет вам задавать список резервных шрифтов (стек шрифтов, рассмотренный нами ранее), которые должны применяться, если первый выбранный шрифт не поддерживается. Если первый из указанных шрифтов не найден, браузер пробует применить следующий и далее по списку, пока не найдет работающий шрифт. В указанном выше третьем примере, если браузер не найдет шрифт Trebuchet MS, он будет использовать шрифт Verdana, а если и шрифт Verdana не поддерживается, он заменит его каким-либо другим шрифтом без засечек.

Семейства типовых шрифтов

Последний вариант «какой-либо другой шрифт без засечек» требует дальнейшего обсуждения. Шрифт «Sans-serif» — это только одно из пяти семейств типовых шрифтов, которые вы можете задавать при помощи свойства **font-family**. Когда вы задаете семейство типовых шрифтов, браузер выбирает доступный шрифт из этой стилистической категории. На рис. 12.2 показаны примеры из каждого семейства. Имена семейств типовых шрифтов не обязательно писать с заглавной буквы.

Шрифты с засечками

Примеры: Cambria, Times New Roman, Georgia

Гарнитуры шрифтов с засечками имеют декоративные засечки или отростки наподобие росчерков на концах определенных штрихов знака.

Шрифты без засечек

Примеры: Arial, Arial Black, Verdana, Trebuchet MS, Helvetica, Geneva

Гарнитуры шрифтов без засечек имеют прямые штрихи знаков, на концах которых нет засечек.

Моноширинные шрифты

Примеры: Courier, Courier New и Lucida Console

В моноширинных гарнитурах (также называемых одинаковой ширины, равноширинные) все знаки занимают площади одинаковой ширины на строке. Например, буква «ш» будет той же ширины, что и «з». Сравните их с пропорциональными шрифтами (такими как тот, что вы сейчас читаете), в которых у разных символов различная ширина.

Рукописные шрифты

Примеры: Monotype Corsiva, Segoe Script и Comic Sans

Рукописные шрифты имитируют почерк или рукописный вид.

Рис. 12.2. *Примеры пяти семейств типовых шрифтов*

Фантазийные шрифты

Примеры: Impact, Western или другие декоративные шрифты

Фантазийные шрифты исключительно декоративные и подходят для заголовков и других акцентируемых надписей. Они редко используются на веб-страницах из-за несоблюдения требований кроссплатформенной доступности и удобочитаемости.

Стеки шрифтов

Лучший способ задать шрифты для веб-страницы — это начать с указания первого шрифта, предоставить несколько похожих альтернатив, и затем завершить семейством типовых шрифтов, которое, по крайне мере, предоставит пользователям правильный диапазон стилей. Например, если вам нужен прямой шрифт без засечек, вы можете начать со своего любимого шрифта (Futura), перечислить несколько более по-

Безопасные стеки шрифтов с поддержкой кириллицы

Ниже приведен список шрифтов и соответствующих им безопасных стеков (с учетом кириллицы):

- Arial Black
- Arial
- Comic Sans MS
- Courier New
- Georgia
- Impact
- Lucida Console
- Lucida Sans Unicode
- Palatino Linotype
- Tahoma
- Times New Roman
- Trebuchet MS
- Verdana

Соответствующие правила CSS выглядят следующим образом:

```
font-family: "Arial Black",
"Helvetica CY", "Nimbus Sans
L" sans-serif;
font-family: Arial,
"Helvetica CY", "Nimbus Sans
L", sans-serif;
font-family: "Comic Sans MS",
"Monaco CY", cursive;
font-family: "Courier New",
"Nimbus Mono L", monospace;
font-family: Georgia, "Century
Schoolbook L", Serif;
font-family: Impact,
"Charcoal CY", sans-serif;
font-family: "Lucida
Console", Monaco, monospace;
font-family: "Lucida Sans
Unicode", "Lucida Grande",
sans-serif;
font-family: "Palatino
Linotype", "Book Antiqua",
Palatino, serif;
font-family: Tahoma, "Geneva
CY", sans-serif;
font-family: "Times New
Roman", "Times CY", "Nimbus
Roman No9 L", serif;
font-family: "Trebuchet MS",
"Helvetica CY", sans-serif;
font-family: Verdana, "Geneva
CY", "DejaVu Sans", sans-
serif;
```

пулярных (Univers, Tahoma, Geneva) и в конце указать типовой шрифт без засечек. По количеству шрифтов, которые можно добавить, ограничений нет, но в основном дизайнеры стараются указывать не более десяти.

```
font-family: Futura, Univers, Tahoma, Geneva, sans-serif;}
```

Хороший стек шрифтов будет содержать стилистически близкие шрифты, которые точно будут установлены на компьютере пользователя. Придерживаясь тех, которые устанавливаются вместе с операционной системой Windows, OS X и Linux, а также шрифтов, устанавливаемых с популярным пакетом прикладных программ, таких как Microsoft Office или Adobe Creative Suite, вы получаете на выбор длинный список «веб-безопасных» шрифтов. Диаграммы и статистические данные, приведенные на следующих сайтах, являются замечательными ресурсами для поиска распространенных шрифтов.

- Полное руководство по предварительно установленным шрифтам в операционных системах Linux, OS X и Windows (**www.apaddedcell.com/sites/www.apaddedcell.com/files/fonts-article/final/index.html**)
- Шрифт Matrix (**media.24ways.org/2007/17/fontmatrix.html**)
- Обзор веб-шрифтов и конструктор стеков шрифтов на сайте Code Style (**www.codestyle.org/css/font-family/index.shtml**)

Если вы хотите больше узнать о том, как использовать стеки шрифтов, я рекомендую прочитать следующие статьи, но также не забудьте самостоятельно поискать актуальные советы во Всемирной паутине.

- Статья «Безопасные шрифтовые CSS стеки для рунета» (**www.xiper.net/collect/html-and-css-tricks/typographics/safe-fonts-part3.html**)
- Статья «Верстальщику о шрифтах. Безопасные шрифты» (**lamp-dev.ru/html-css/verstalshhiku-o-shriftax-bezopasnye-shrifty/**)

Итак, как вы видите, задавание шрифтов для веб-страниц больше похоже на рекомендацию: у вас нет абсолютного контроля над тем, какой шрифт увидят ваши пользователи. Возможно, это окажется тот, что вы указали первым, а возможно — типовой запасной вариант. Это одна из таких особенностей веб-дизайна, о которых не стоит забывать.

Теперь настало время для форматирования меню бистро «Черный Гусь». Мы будем добавлять новые правила стилей по одному за раз по мере изучения каждого нового свойства.

УПРАЖНЕНИЕ 12.1. ФОРМАТИРОВАНИЕ МЕНЮ БИСТРО

В этом упражнении мы добавим свойства шрифта к документу меню бистро «Черный Гусь», *menu.html*, который доступен на диске, прилагающемся к книге. Откройте документ в текстовом редакторе. Также вы можете открыть его в браузере, чтобы просмотреть актуальный на данный момент вид страницы. Она должна выглядеть так, как показано на рис. 12.1. Сохраните этот документ, потому что упражнение будет продолжаться по мере изучения нами дополнительных свойств шрифтов.

В это упражнение я добавила вложенный шрифт, чтобы показать вам, как прост в использовании сервис Google Web Fonts.

1. Для этого упражнения мы собираемся использовать глобальную таблицу стилей. Начните с добавления элемента `style` в раздел заголовка документа, как показано ниже:

```
<head>
<title>Бистро "Черный гусь". Летнее меню</title>
<style>
</style>
</head>
```

2. Мне нужно, чтобы весь текст страницы отображался шрифтом Verdana или каким-нибудь другим шрифтом без засечек. Вместо написания правила для каждого элемента в документе, создайте одно для элемента `body`, которое будет наследоваться всеми другими элементами, содержащимися в нем. Добавьте это правило к глобальной таблице стилей.

```
<style type="text/css">
body {font-family: Verdana, sans-serif;}
</style>
```

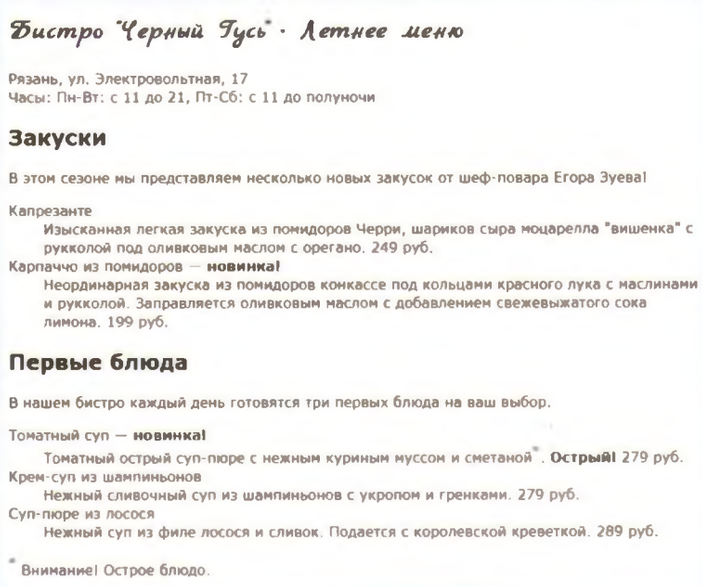
3. Для заголовка «Бистро "Черный гусь". Летнее меню» я хочу использовать фантазийный шрифт, поэтому я выбрала бесплатный шрифт Marck Script на сайте Google Web Fonts (**www.google.com/webfonts**). Сервис Google предоставил необходимый код, связывающий файл шрифта, хранящийся на сервере компании, с моим HTML-файлом (фактически это ссылка на внешнюю таблицу стилей). Его необходимо вставить в раздел заголовка документа, так что скопируйте без изменений.

```
<head>
<title>Black Goose Bistro</title>
<link href='http://fonts.googleapis.com/css?family=Marck+Script&subset=latin,cyrillic' rel='stylesheet'>
</head>
```

4. Далее напишите правило, которое будет применяться к элементу **h1**. Обратите внимание, что в качестве запасного варианта я указала Georgia или другой шрифт с засечками.

```
<style>
body {font-family: Verdana, sans-serif;}
h1 {font-family: "Marck Script", Georgia, serif;}
</style>
```

5. Сохраните документ и обновите страницу в браузере. Он должен выглядеть так, как показано на рис. 12.3. Обратите внимание, что для просмотра шрифта заголовка Marck Script вам необходимо подключение к Интернету. Мы поработаем над размером шрифта в следующем разделе.


Бистро "Черный Гусь" · Летнее меню

Рязань, ул. Электровольтная, 17
Часы: Пн-Вт: с 11 до 21, Пт-Сб: с 11 до полуночи

Закуски

В этом сезоне мы представляем несколько новых закусок от шеф-повара Егора Зуева!

Капрезанте
Изысканная легкая закуска из помидоров Черри, шариков сыра моцарелла "вишенка" с рукколой под оливковым маслом с орегано. 249 руб.

Карпаччо из помидоров — **новинка!**
Неординарная закуска из помидоров конкассе под кольцами красного лука с маслинами и рукколой. Заправляется оливковым маслом с добавлением свежевыжатого сока лимона. 199 руб.

Первые блюда

В нашем бистро каждый день готовятся три первых блюда на ваш выбор.

Томатный суп — **новинка!**
Томатный острый суп-пюре с нежным куриным муссом и сметаной*. **Острый!** 279 руб.

Крем-суп из шампиньонов
Нежный сливочный суп из шампиньонов с укропом и гренками. 279 руб.

Суп-пюре из лосося
Нежный суп из филе лосося и сливок. Подается с королевской креветкой. 289 руб.

* Внимание! Острое блюдо.

Рис. 12.3. Меню после изменения шрифтов

Определение размера шрифта

Чтобы задать размер текста, используйте свойство с именем **font-size**.

font-size

Принимаемые значения: *значение длины* | *проценты* | **xx-small** | **x-small** | **small** | **medium** | **large** | **x-large** | **xx-large** | **smaller** | **larger** | **inherit**

Значение по умолчанию: **medium**

Применение: ко всем элементам

Наследование: да

Вы можете задать размер шрифта текста следующими способами:

- Конкретным значением, используя одну из единиц измерения длины в CSS (см. врезку «Единицы измерения в CSS» для полного списка единиц измерения), как показано здесь:

```
h1 { font-size: 1.5em; }
```

При указании единиц удостоверьтесь, что аббревиатура единицы измерения расположена сразу за числом, без пробела между ними. Так писать **НЕПРАВИЛЬНО**:

```
h1 { font-size: 1.5 em; } /*пробел перед em*/
```

- Процентным значением, задающим размер шрифта больше или меньше размера шрифта элемента по умолчанию или унаследованного размера шрифта:

```
h1 { font-size: 150%; }
```

- Используя одно из абсолютных зарезервированных слов (**xx-small**, **x-small**, **small**, **medium**, **large**, **x-large**, **xx-large**). В большинстве современных браузеров зарезервированное слово **medium** соответствует размеру шрифта по умолчанию:

```
h1 { font-size: x-large; }
```

- Используя относительные (или сравнительные) зарезервированные слова (**larger** или **smaller**), чтобы заставить отображаться текст больше или меньше близлежащего текста:

```
strong { font-size: larger; }
```

Я сокращу перечень и отмечу, что, несмотря на все эти варианты, предпочитаемыми значениями для свойства **font-size** в современном вебдизайне являются единицы измерения em и процентные значения (или их сочетания). О других значениях свойства **font-size** я расскажу чуть позже, а сейчас давайте начнем обсуждение с наиболее часто используемого подхода.

Как единицы измерения em, так и проценты являются *относительными* единицами измерения, что означает, что они основываются на другом размере, а именно на размере шрифта **font-size** родительского элемента.

Процентные значения

В этом примере размер шрифта **font-size** «родителя» (**body**) элемента **h1** определен как 100% от установленного по умолчанию размера шрифта текста (обычно равного 16 пикселам). Элемент **h1** наследует размер **16px** от элемента **body**, а прибавление значения свойства **font-size** равное 150%, увеличивает *унаследованное* значение, и размер шрифта текста элемента **h1** в результате будет равен 24 пикселам. Если у пользователя задан размер шрифта 30 пикселов, например чтобы прочитать текст в окне браузера телевизора из другого угла комнаты, размер шрифта текста элемента **h1** в результате будет равен 45 пикселам, но пропорции относительно основного текста будут сохранены, в этом и смысл использования относительных единиц измерения.

```
body { font-size: 100%; }
h1 { font-size: 150%; } /* 150% от 16 = 24 */
```

Единицы измерения в CSS

Спецификация CSS3 предоставляет разнообразие единиц измерения. Они попадают в две обширные категории: *абсолютные* и *относительные*.

Относительные единицы измерения

Относительные единицы измерения основываются на размере чего-либо, например, размере шрифта текста по умолчанию или размере родительского элемента.

px пиксел (pixel) считался относительной единицей в CSS2.1, потому что зависит от разрешения экрана.

em единица измерения, равная текущему размеру шрифта.

ex высота «x», приблизительно высота строчной буквы «x» шрифта.

Следующие единицы являются новыми в CSS3. Возможно, потребуется некоторое время, чтобы они получили поддержку браузеров.

rem корневая единица измерения em, равна величине в em корневого элемента (**html**)

ch ширина нуля, равняется ширине символа ноль (0) текущего шрифта и размера

vw единица ширины окна просмотра, равная 1/100 от текущей ширины окна просмотра (окна браузера)

vh единица высоты окна просмотра, равная 1/100 от текущей высоты окна просмотра (окна браузера)

vm минимальная единица окна просмотра, равная значению **vw** или **vh**, в зависимости от того, какое из них меньше.

Абсолютные единицы измерения

Абсолютные единицы измерения имеют зарезервированные значения или реальные эквиваленты.

px пиксел, определяется в CSS3 как абсолютная единица измерения, равная 1/96 дюйма

pt пункты (point) (1/72 дюйма в CSS2.1)

pc пика (pica) (1 пика = 12 пунктам)

mm миллиметры (millimeter)

cm сантиметры (centimeter)

in дюймы (inch)

Следует избегать использования абсолютных единиц измерения для таблиц стилей веб-страниц, потому что они не связаны с экранами компьютеров. Однако если вы создаете таблицу стилей, используемую для документа, который нужно будет распечатать, они могут прийтись очень кстати.

Заметили, что пиксел (**px**) встречается в обоих списках? Это потому, что консорциум Всемирной паутины еще не принял окончательного решения. Если отвлечься от определений, на практике пикселы используются как абсолютные единицы измерения, которые не настолько гибкие, как истинные относительные единицы измерения.

Единицы измерения em

ПРИМЕЧАНИЕ

Не путайте единицу измерения em и HTML-элемент **em**, используемый для разметки выделенного текста. Это абсолютно разные вещи.

Em — это относительная единица измерения, которая, в традиционной типографике, основывается на ширине заглавной буквы «M» (отсюда и название «em» («эм»)). В спецификации CSS, единица em вычисляется как расстояние между базовыми строками, когда шрифт размещается без какого-либо дополнительного промежутка между строками (также известного как межстрочный интервал, или интерлиньяж). Для текста с размером шрифта равным 16 пикселам, единица измерения em имеет размер 16 пикселов; для размера шрифта текста в 12 пикселов, единица измерения em равняется 12 пикселам и т. д., как показано на рис. 12.4.

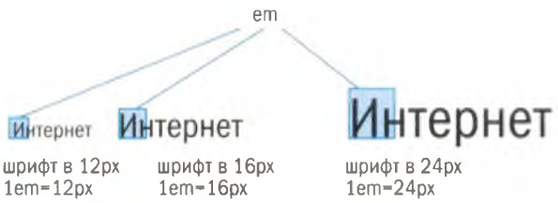


Рис. 12.4. Единица измерения em основывается на размере шрифта текста

Как только браузер вычислили размер единицы измерения em для текстового элемента, полученный данные можно использовать для любого вида измерений, таких как отступы, поля, ширина элемента на странице и т. д.

Для задания свойства **font-size** в единицах **em**, значение em работает как масштабный коэффициент, подобно процентам. В следующем примере размер шрифта текста элемента **body** указан 100% (предположим, что это по умолчанию 16px). Задание элементам **h1** значения свойства **font-size**, равного 1.5 em, делает их размер в полтора раза больше наследуемого значения или равным 24 пикселам.

```
body { font-size: 100%; }

h1 { font-size: 1.5em; } /* 1.5 x 16 = 24 */
```

Примеры использования единицы измерения em

На момент написания книги наиболее популярным решением для достижения единообразного отображения единиц em, было задавание размера шрифта текста элемента **body** равным 100% (сохраняя его значение, заданное по умолчанию или предпочитаемое пользователем), а затем использовать единицы em при задании размеров шрифта впоследствии, как мы делали в предыдущем примере. Это сохраняет заданные пользователем размер отображения шрифта, в то же время, гарантируя, что размеры элементов текста будут меняться пропорционально.

При работе с единицами измерения em существуют некоторые затруднения. Одно из них связано с ошибками округления, из-за чего суще-

ствует много противоречий между браузерами и платформами, когда размер шрифта текста задается в единицах измерения em.

Другая сложность при использовании единиц измерения em заключается в том, что в их основе лежит наследуемый размер элемента, а значит, их размер зависит от контекста, к которому они применяются. Если у вас множество вложенных элементов, увеличение или уменьшение размера будет возрастать с каждым уровнем вложения. Рассмотрим пример.

Допустим, изначально величина текста раздела **body** документа равна 100% (16 пикселов), но вам нужно, чтобы текст элемента **article** был высотой всего 14 пикселов. Поделив целевое значение (14 пикселов) на контекст (16 пикселов), вы получите значение свойства **font-size** равное .875em. Теперь, предположим, вам нужно, чтобы высота шрифта текста элементов **h2** в этой публикации составляла 18 пикселов. Так, делим целевое значение (18px) на контекст (14px) и получаем конечное значение в единицах измерения em равное 1,28571429. Ничего себе значение! Можно его округлить (оставив хотя бы четыре знака после запятой), но в этом нет необходимости.

```
body {font-size: 100%;}

article {font-size: .875em;}

/*на основе наследуемого размера шрифта текста элемента body
*/

article h2 {font-size: 1.28571429em; }

/*на основе размера шрифта элемента article, а не body */
```

Итан Маркотт (известный как автор книги «Отзывчивый веб-дизайн») уже несколько лет использует формулу «целевое значение ÷ контекст = результат», и она оказывается полезной для построения жидких макетов и выполнения иных задач, связанных с относительными размерами. Конечно, мы еще встретимся с ней в этой книге.

Поэтому обратите пристальное внимание и напишите правила стилей таким образом, чтобы компенсировать этот эффект составления. Современный подход, позволяющий обойти такую проблему, описан во врезке «Единица измерения em корневого элемента»

Несмотря на то что некоторые разработчики предпочитают измерять размер шрифта в пикселах, в силу того что так предоставляется более точный контроль, большинство считает пикселы слишком негибкими, а относительные единицы измерения (em или %) — более подходящими для данной среды. А пока мы отбрасываем пикселы, все абсолютные единицы измерения, такие как pt, pc, in, mm и cm, не подходят для мониторов компьютеров (хотя могут быть использованы в таблицах стилей для печатных документов).

Еще один недостаток пиксельных значений свойства **font-size** заключается в том, что браузер Internet Explorer (все версии) не позволяет масштабировать текст, если величина шрифта указана в пикселах. Это значит, что пользователи не смогут изменить 10- или 11-пиксельный

Единица измерения em корневого элемента

В спецификации CSS3 введена новая относительная единица измерения, которая называется **rem** (сокращение от «root em» — «корневая единица em»), которая определяет размер шрифта на основе размера корневого элемента (**html**). Если указать размер HTML-элемента (предположить, что он равен 100%), все элементы, измеряемые в единицах **rem**, будут указываться относительно этого размера, а не наследуемого размера. Это позволяет избавиться от проблемы составления, из-за которой единицы измерения **em** становятся потенциально отягощающими. Недостаток в том, что Internet Explorer 8 и его более ранние версии, а также старые версии других браузеров, не поддерживают единицы измерения **rem**, поэтому необходимо предоставить запасной вариант размера шрифта в пикселах. Браузеры, которые поддерживают единицы измерения **rem**, используют последнее определение в стеке.

```
html {
font-size: 100%;
}
#main {
font-size: 12px;
font-size: .75rem;
}
```

Единицы измерения **rem** приобретают все большую популярность среди веб-разработчиков. Для более подробного знакомства с ними, я рекомендую прочитать статью Джонатана Снука, доступную по адресу **snook.ca/archives/html_and_css/font-size-with-rem**.

Чтобы рассчитать значение в единицах em и процентах, примените формулу: ***Целевое значение ÷ контекст = результат.***

шрифт, даже если они не способны его прочитать. Это большой минус с точки зрения доступности. Браузеры Internet Explorer 7 и более поздние версии позволяют увеличивать масштаб всей страницы, что уже является улучшением, но все же не идеальным вариантом для пользователя.

Использование зарезервированных слов

Еще один способ задать размер шрифта — при помощи одного из предопределенных абсолютных зарезервированных слов: **xx-small**, **x-small**, **small**, **medium**, **large**, **x-large** и **xx-large**. Зарезервированные слова не соответствуют конкретным значениям, а последовательно изменяются друг относительно друга. В современных браузерах размером по умолчанию является **medium**. На рис. 12.5 показано, как каждое из абсолютных зарезервированных слов представляется в браузере, когда размер шрифта текста по умолчанию установлен в 16 пикселов. Я привела примеры со шрифтами Verdana и Times New Roman, чтобы показать, что даже с одним и тем же базовым размером, существует большая разница в удобочитаемости при размерах **small** и ниже.

Преимуществом зарезервированных слов является то, что современные браузеры в стандартном режиме никогда не позволят тексту, размер шрифта которого задан при помощи зарезервированных слов, отображаться размером меньше 9 пикселов, поэтому пользователи защищены от вывода шрифта, трудного для чтения (хотя я бы по-прежнему предпочла шрифт Verdana для большего удобства прочтения).

Недостаток зарезервированных слов в том, что для задавания размера они неточны и непредсказуемы. Например, тогда как большинство браузеров увеличивают каждое значение на 120%, некоторые используют иной коэффициент масштаба, равный 150%. Относительные зарезервированные слова, **larger** и **smaller**, используются для изменения размера шрифта текста относительно размера шрифта текста родительского элемента. Точная величина изменения размера определяется каждым браузером, и находится вне вашего контроля. Несмотря на данное ограничение, это легкий способ небольшого увеличения или уменьшения размера шрифта, если точные пропорции не слишком важны.

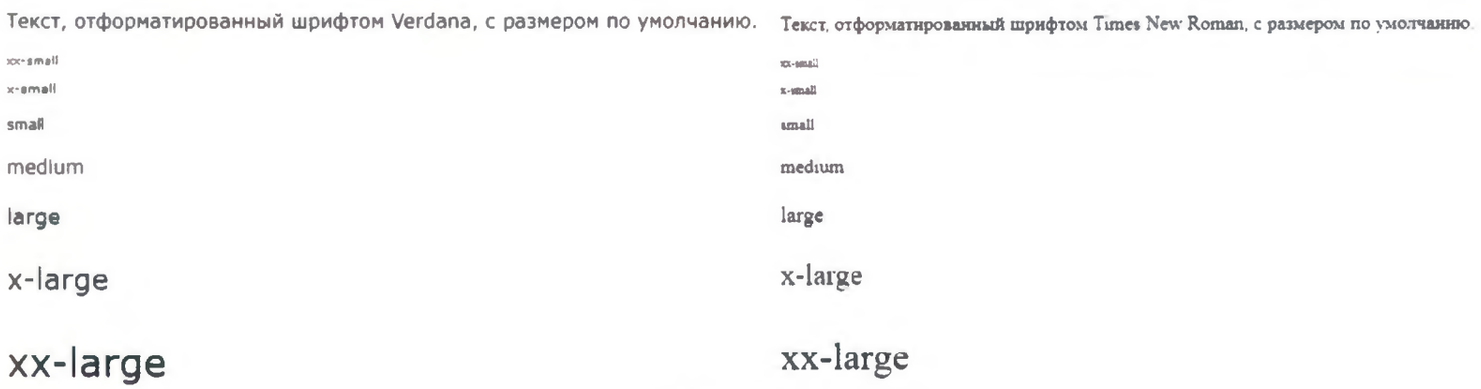


Рис. 12.5. Размер шрифта текста, заданный при помощи абсолютных зарезервированных слов

Насыщенность шрифта

Оставшиеся свойства шрифтов просты. Например, если вы хотите, чтобы текстовый элемент отображался шрифтом с полужирным начертанием, используйте свойство **font-weight**.

font-weight

Принимаемые значения: **normal** | **bold** | **bolder** | **lighter** | **100** | **200** | **300** | **400** | **500** | **600** | **700** | **800** |**900** | **inherit**

Значение по умолчанию: **normal**

Применение: ко всем элементам

Наследование: да

Как вы могли заметить, свойство **font-weight** имеет много предопределенных значений, включая описательные термины (**normal**, **bold**, **bolder** и **lighter**) и девять числовых значений (от **100** до **900**) для

О наследовании

Как вы видите, свойства CSS включают **inherit** в свой список зарезервированных значений. Значение **inherit** позволяет вам в прямой форме принудить элемент унаследовать значение свойства стилей от его родительского элемента. Это может оказаться полезным для замены других стилей, примененных к этому элементу, и гарантировать, что он всегда будет соответствовать своему «родителю».

УПРАЖНЕНИЕ12.2. НАСТРОЙКА РАЗМЕРА ШРИФТА

Уточним размер некоторых текстовых элементов для придания меню более изысканного вида. Откройте файл *menu_summer.html* в текстовом редакторе и выполните указанные ниже шаги. Вы можете в любой момент сохранить документ и заглянуть в браузер, чтобы увидеть результаты вашей работы. Не стесняйтесь также пробовать другие значения размера в ходе упражнения.

1. Существует множество способов изменения размера шрифта текста на веб-страницах. В этом примере я буду придерживаться любимого метода лучших веб-разработчиков, которых я знаю — начну с того, что задам отправную точку, и установлю размер шрифта элемента **body** равным 100%, предоставив таким образом возможность для дальнейшего использования единиц измерения em.

```
body {
font-family: Verdana, sans-serif;
font-size: 100%;
}
```

2. Я бы предпочла, чтобы основная часть текста документа отображалась шрифтом 14 пикселов, вместо общепринятого размера, по умолчанию равного 16 пикселам (если моим посетителям он покажется слишком мелким, они всегда могут увеличить масштаб текста в браузере). Я добавлю новое правило стилей со сгруппированным селектором, чтобы задать размер элементов **p** и **dl** равный .875em, применив формулу: целевое значение (14) ÷ контекст (16) = .875.

```
p, dl {
font-size: .875em;
}
```

3. Теперь возьмем под контроль размер заголовка. Я собираюсь сделать основной заголовок (**h1**) в 1.5 раза больше, чем размер основной части текста. Размер шрифта текста элементов **h2** может быть установленным по умолчанию (1em).

```
h1 {
font-family: "Marck Script", Georgia,
serif;
font-size: 1.5em
}
h2 {
font-size: 1em;
}
```

На рис. 12.6 показан результат наших усилий по заданию размера шрифта.

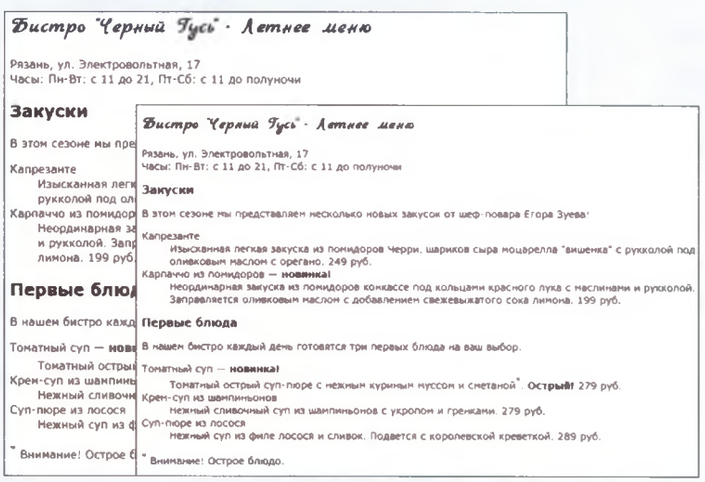
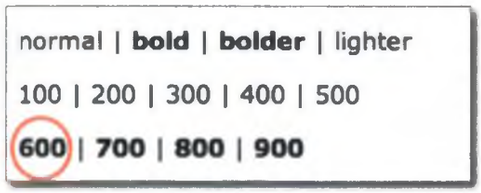


Рис. 12.6. *Страница меню после нескольких незначительных изменений свойства* ***font-size***

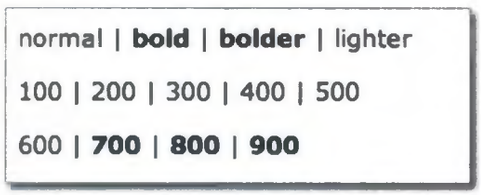
задавания различных вариантов насыщенности шрифта, если они доступны. Из-за того, что большинство шрифтов, распространенных во Всемирной паутине, имеют только два варианта насыщенности, обычный (или нормальный) и полужирный, единственное значение насыщенности шрифта, которое вы будете использовать в большинстве случаев — это **bold**. Также вы можете использовать значение **normal**, чтобы форматировать обычным начертанием текст, который в противном случае будет отображаться полужирным начертанием (например заголовки или акцентированный текст).

Список числовых значений — интересная идея, но из-за того, что не многие шрифты имеют такой диапазон весов, и из-за того, что поддержка

браузерами не стабильна, они не часто используются. В общем, установка числового значения 600 и более дает полужирный текст, хотя даже это может отличаться в зависимости от браузера, как показано на рис. 12.7.



Отображение в браузере Safari



Отображение в браузере Firefox (OS X)

***Рис. 12.7.** Результат применения значений свойства* `font-weight`

УПРАЖНЕНИЕ 12.3. УСТАНОВКА ПОЛУЖИРНОГО НАЧЕРТАНИЯ ШРИФТА ТЕКСТА

Возвратимся к странице меню. Я решила, что названия всех его пунктов следует выделить полужирным текстом. Что *не* следует делать, так это заключать каждый из них в теги **<b>** — это так устарело! Также не нужно помечать их как акцентированные элементы **strong** — это семантически неверно. Правильным подходом будет применение стиля к семантически верным элементам **dt** (определение термина), чтобы сделать их всех полужирными за один раз. Добавьте это правило в конец таблицы стилей, сохраните файл и протестируйте его в браузере (рис. 12.8).

```
dt { font-weight: bold; }
```

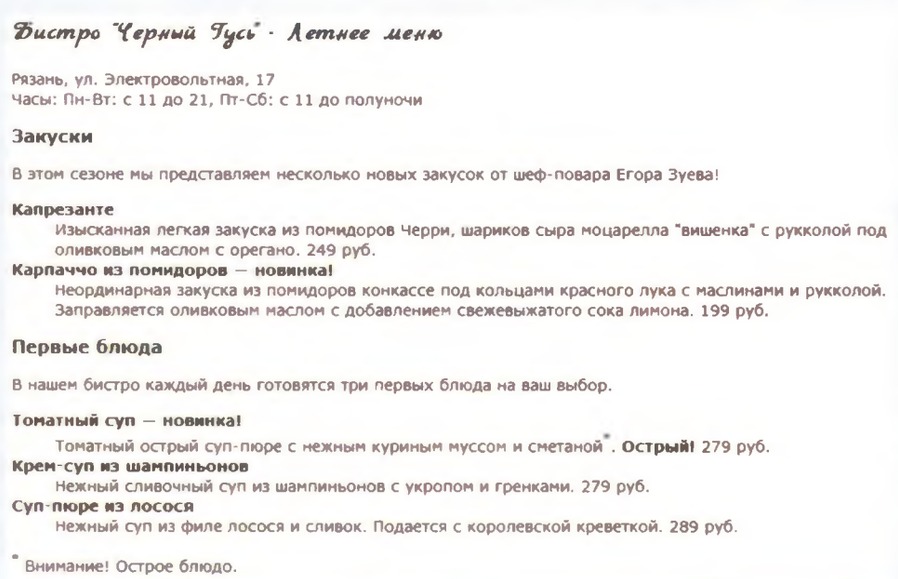


***Рис. 12.8.** Применение свойства **font-weight** к элементам **dt** в меню*

Начертание шрифта

Свойство **font-style** влияет на *положение* текста, то есть имеют ли символы вертикальный (**normal**) или наклонный (**italic** и **oblique**) вид.

font-style

Принимаемые значения: **normal** | **italic** | **oblique** | **inherit**

Значение по умолчанию: **normal**

Применение: ко всем элементам

Наследование: да

Курсивное и наклонное начертания шрифта внешне схожи. Различие в том, что курсивный шрифт — это обычно отдельный дизайн шрифта с искривленными символами, в то время как наклонный текст создается путем наклона обычного шрифта. Суть в том, что в большинстве браузеров они

УПРАЖНЕНИЕ 12.4. ФОРМАТИРОВАНИЕ ТЕКСТА КУРСИВОМ

Теперь, когда все названия пунктов меню отображаются полужирным начертанием, некоторый текст, помеченный как **strong**, не очень хорошо выделяется, поэтому следует оформить их курсивным шрифтом для дальнейшего выделения. Для этого примените свойство **font-style** к элементу **strong**.

```
strong { font-style: italic;}
```

Снова сохраните страницу и обновите ее в браузере. Акцентированные элементы должны выглядеть как части изображения, выделенные на рис. 12.10.

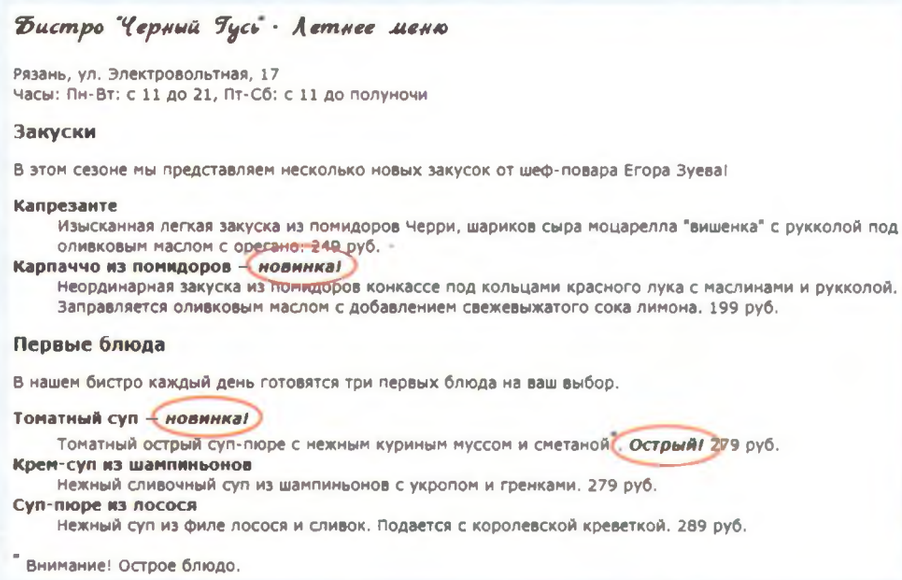


Рис. 12.10. *Применение свойства* ***font-style*** *к акцентированным элементам*

могут выглядеть совершенно одинаково (см. рис. 12.9). Вероятнее всего, вы будете использовать свойство **font-style**, только чтобы сделать отображение текста курсивом **italic** или обычным **normal**, если текст отображается курсивом по умолчанию (например, акцентированный текст).

Рис. 12.9. Форматирование текста курсивным и наклонным шрифтами

Капитель

Некоторые шрифты выглядят как капитель. Это отдельный дизайн шрифта, который использует уменьшенные заглавные буквы вместо строчных. Односложное свойство **font-variant** предназначено для того, чтобы позволить дизайнерам задавать подобный капительный шрифт для текстовых элементов.

font-variant

Принимаемые значения: **normal** | **small-caps** | **inherit**

Значение по умолчанию: **normal**

Применение: ко всем элементам

Наследование: да

В большинстве случаев капительный шрифт действительно недоступен, поэтому браузеры имитируют его уменьшением масштаба заглавных букв текущего шрифта. Для дизайнеров с опытом работы в типографии это не идеальное решение, так как ширина штрихов не согласована, но вариант приемлем для внесения разнообразия в небольшой фрагмент текста. Мы используем свойство **font-variant** в упражнении 12.6.

Сокращенная запись свойства шрифта

Определение нескольких свойств шрифта для каждого текстового элемента может стать повторяющимся и долгим занятием, поэтому разработчики CSS предоставили краткую запись свойства **font**, которая объединяет все связанные со шрифтом свойства в одно правило.

font

Принимаемые значения: **font-style font-weight font-variant font-size/line-height font-family** | **inherit**

Значение по умолчанию: зависит от значения по умолчанию каждого перечисленного свойства

Применение: ко всем элементам

Наследование: да

Значение свойства **font** — это список значений всех свойств шрифта, которые мы только что видели, разделенных символами пробела. В этом свойстве порядок следования значений важен:

```
{ font: style weight variant size/line-height font-family }
```

Как минимум свойство **font** *должно* включать в себя значение **font-size** и значение **font-family** в указанном порядке. Пропуск одного или расположение их в неправильном порядке станет причиной того, что целое правило будет недействительным. Ниже приведен пример значения минимального свойства шрифта:

```
p { font: 1em sans-serif; }
```

Как только указаны требования для размера и семейства шрифтов, другие значения необязательны и могут располагаться в любом порядке *до* свойства **font-size**. Когда начертание, насыщенность или вариант пропускаются, они будут возвращены со значением **normal**. Для свойства указано еще значение, **line-height**, которое мы пока что не видели. Оно устанавливает высоту строки текста между базовыми линиями. Значение указывается сразу после свойства **font-size** и отделяется слешем, как показано в примерах ниже.

```
h3 { font: oblique bold small-caps 1.5em/1.8em Verdana, sans-serif; }
h2 { font: bold 1.75em/2 sans-serif; }
```

Теперь используем краткую запись свойства **font** для внесения некоторых изменений заголовков **h2**.

УПРАЖНЕНИЕ 12.5. ПРИМЕНЕНИЕ СОКРАЩЕННОЙ ЗАПИСИ СВОЙСТВА ШРИФТА

Еще одна последняя настройка — и немного передохнем. Чтобы сэкономить пространство, мы можем объединить все свойства шрифта, указанные для элемента **h1** в одно определение с коротким свойством **font**.

```
h1 {
font: bold 1.5em "Marck Script", Georgia, serif;
}
```

Вы, должно быть, находите излишним добавление в правило значения насыщенности шрифта **bold**. Ведь элемент **h1** уже был полужирным, верно? Дело в том, что, если вы пропускаете значение в записи свойства **font**, оно устанавливается по умолчанию и не остается таким, как было ранее. В данном случае, значением насыщенности по умолчанию **font-weight** является **normal**. Из-за того, что написанное нами правило таблицы стилей переопределяет установки заголовков в браузере, элемент **h1** отобразился бы текстом с обычным начертанием шрифта, если бы мы явно не сделали их полужирным **bold** в свойстве **font**. В этом смысле краткая запись свойств может быть коварной... будьте внимательны, чтобы ничего не пропустить и заменить значение по умолчанию или унаследованное значение, как и планировали.

Можете сохранить файл и просмотреть его в браузере, но если вы все сделали верно, он будет выглядеть точно так же, как в предыдущем шаге.

Изменение цвета текста

Вы бегло ознакомились с тем, как изменять цвет текста, в главе 11, и честно говоря, здесь к этому мало что можно добавить. Цвет текста меняется при помощи свойства **color**.

color

Принимаемые значения: *значение цвета (имя или число)* | **inherit**

Значение по умолчанию: зависит от браузера и предпочтений пользователя

Применение: ко всем элементам

Наследование: да

Используется свойство **color** очень прямолинейно. Его значением может быть предопределенное имя цвета (см. врезку «Имена цветов») или числовое значение, описывающее конкретный оттенок в цветовой модели RGB. Ниже приведены несколько примеров, все из которых делают элементы **h1** в документе серыми:

```
h1 { color: gray; }
h1 { color: #666666; }
h1 { color: #666; }
h1 { color: rgb(102,102,102); }
```

Не волнуйтесь по поводу числовых значений сейчас — я всего лишь хотела, чтобы вы увидели, как они выглядят. Оттенки в модели RGB подробно рассматривается в главе 13, поэтому в данной главе мы будем придерживаться довольно ограниченного списка имен цветов с целью демонстрации.

Цвет наследуется, поэтому вы можете менять цвет всего текста в документе, применив свойство **color** к элементу **body**, как показано ниже:

```
body { color: fuchsia; }
```

Вы, вероятно, не захотите, чтобы весь ваш текст отображался цветом фуксии, но общий смысл вы уловили.

Я хочу обратить ваше внимание на то, что свойство **color** не является строго текстовым. На самом деле, в соответствии со спецификацией CSS, оно используется для изменения *основного цвета* (*цвета переднего плана*) (как противоположность фоновому цвету или заднему плану) элемента. Основной цвет элемента состоит как из текста, который он содержит, так и его границы.

Когда вы применяете к элементу (включая элементы изображения) цвет, он будет использоваться также и для границ, если нет свойства **border-color**, которое переопределит его. Мы поговорим более подробно о границах в главе 14.

Перед тем как добавить цвет на страницу меню, я хочу немного отвлечься и представить вам еще несколько типов селекторов, которые придадут намного больше гибкости в целенаправленном выборе элементов в документе для стилизации.

НА ЗАМЕТКУ

Имена цветов

В CSS2.1 определены 17 стандартных имен цветов:

black (Черный)

white (Белый)

purple (Фиолетовый, или пурпурный)

lime (Лаймовый, или желто-зеленый, или салатовый)

navy (Темно-синий)

aqua (Цвет морской волны)

silver (Серебристый)

maroon (Коричнево-малиновый, или красно-коричневый, или каштановый)

fuchsia (Цвет фуксии)

olive (Оливковый, или желтовато-зеленый, или желто-коричневый)

blue (Синий)

orange (Оранжевый)

gray (Серый)

red (Красный)

green (Зеленый)

yellow (Желтый)

teal (Сине-зеленый)

Обновленный цветовой модуль CSS3 позволяет указывать в таблице стилей более широкий набор имен цветов (147 имен). Примеры каждого цвета можно найти на прилагающемся к книге DVD-диске, в файле ***Имена цветов.pdf***.

Еще несколько типов селекторов

До сих пор мы использовали имена элементов в качестве селекторов. В предыдущей главе вы видели, как селекторы могут быть сгруппированы вместе в разделенный запятыми список, так что допустимо применять свойства к нескольким элементам одновременно. Ниже приведены примеры селекторов, которые вы уже знаете.

Селектор элемента `p { color: navy; }`

Сгруппированные селекторы `p, ul, p, td, th { color: navy; }`

Отрицательной стороной такого способа выбора элементов согласно селекторам, конечно же, является то, что свойство (в данном случае, темно-синий цвет текста) будет применяться к каждому абзацу и другим перечисленным элементам в документе. А иногда требуется применить правило к конкретному абзацу или абзацам. В этом разделе, мы рассмотрим три типа селекторов, которые позволят сделать это: селекторы потомков, селекторы идентификаторов и селекторы классов.

Селекторы потомков

Символ пробела между именами элементов означает, что второй элемент должен содержаться внутри первого.

Селектор потомков целенаправленно выбирает элементы, которые содержатся внутри (и поэтому они потомки) другого элемента. Это пример *контекстуального селектора* (или *контекстного селектора*), потому что он выбирает элемент на основе его контекста или связи с другим элементом. Во врезке «Другие контекстуальные селекторы» перечислены дополнительные контекстуальные селекторы.

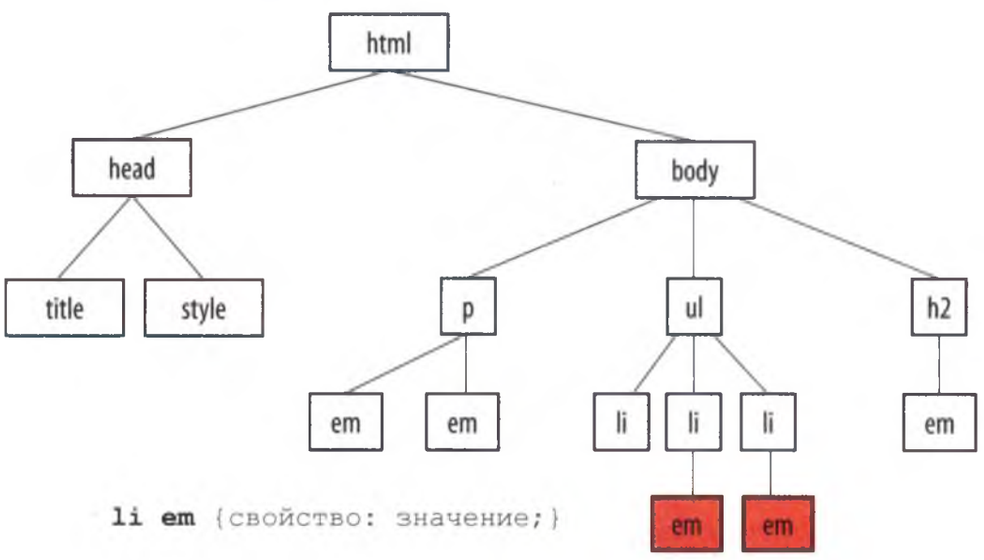


Рис. 12.11. *Выбраны только элементы* ***em*** *внутри элементов* ***li****. Другие элементы* ***em*** *не затронуты*

Селекторы потомков указываются в разделенном символом пробела списке. Этот пример целенаправленно выбирает элементы акцентиро-

Другие контекстуальные селекторы

Селекторы потомков являются одним из четырех типов контекстуальных селекторов. Остальные три — *селекторы дочерних элементов* и *селекторы смежных элементов одного уровня* и *обобщенные селекторы одного уровня*.

Селектор дочерних элементов

Селектор дочерних элементов похож на селектор потомков, но он целенаправленно выбирает только прямых потомков данного элемента (между ними не может быть других уровней иерархии). Они указываются при помощи знака «больше» (**>**). Это правило воздействует на акцентированный текст, но только когда он напрямую содержится в элементе **p**. Элементы **em** в других элементах, таких как пункты списка (**li**) или якоря (**a**), не будут затрагиваться.

```
p > em {font-weight: bold; }
```

Селектор смежных элементов одного уровня

Селекторы смежных элементов одного уровня используются для целенаправленного выбора элемента, который расположен сразу после другого элемента с тем же родителем. Он указывается при помощи знака «плюс» (**+**). Это правило воздействует только на те абзацы, которые следуют за элементом **h1**. Другие абзацы не затрагиваются.

```
h1 + p {font-style: italic;}
```

Обобщенные селекторы одного уровня

Новый в CSS3

Обобщенный селектор одного уровня выбирает элемент, имеющий общий с ним родительский элемент и указанный после него в исходном коде. Элементы не обязательно должны следовать непосредственно друг за другом. Этот тип селектора является новым в CSS3 и не поддерживается браузером Internet Explorer 8 и его более ранними версиями. Приведенное ниже правило выбирает все элементы **h2**, разделяющие один родительский элемент (например, **section** или **article**) с элементом **h1** и стоящие в документе после него.

```
h1 ~ h2 {font-weight: normal;}
```

ванного текста (**em**), но *только* когда они появляются в пунктах списка (**li**). Акцентированный текст в абзацах и другие элементы не подвергнутся воздействию (рис. 12.11).

```
li em { color: olive; }
```

Ниже приведен другой пример, который показывает, как контекстуальные селекторы могут быть сгруппированы в разделенном запятыми списке, точно как мы видели раньше. Это правило целенаправленно выбирает элементы **em**, но только когда они появляются в заголовках **h1**, **h2** и **h3**.

```
h1 em, h2 em, h3 em { color: red; }
```

Также возможно формировать вложенные в несколько уровней селекторы потомков. В примере ниже целенаправленно выбираются

элементы **em**, которые появляются в якорях (**a**) в нумерованных списках (**ol**).

```
ol a em { font-variant: small-caps; }
```

Селекторы идентификатора

В главе 5 вы узнали об атрибуте **id**, который назначает элементу уникальное идентифицирующее имя (его *идентифицирующая ссылка*). Атрибут **id** может быть использован с любым HTML-элементом и, как правило, используется для придания смысловых значений универсальным элементам **div** и **span**.

Символ # определяет селектор идентификатора.

Селекторы идентификатора позволяют вам целенаправленно выбирать элементы по их **id** значениям. Символ, с помощью которого распознаются селекторы идентификатора, — знак числа **#**, также называемый знаком «решетка» или «октоторп».

Ниже приведен пример пункта списка с идентифицирующей ссылкой.

```
<li id="catalog1234">Майка со смайликом</li>
```

НА ЗАМЕТКУ

Значения идентификаторов должны начинаться с латинской буквы (**A** - **Z** или **a** - **z**). Кроме букв имя может содержать цифры (**0** - **9**), дефисы (**-**), символы подчеркивания (**_**), двоеточия (**:**) и точки (**.**). Обратите внимание, что двоеточий и точек рекомендуется избегать, поскольку при использовании значений идентификаторов в качестве селекторов, они могут быть приняты за элементы синтаксиса CSS.

Теперь вы можете написать правило стилей для этого пункта списка, используя селектор идентификатора наподобие этого (обратите внимание, символ **#** предшествует ссылке **id**):

```
li#catalog1234 { color: red; }
```

Из-за того, что значения **id** должны быть уникальными в документе, допускается пропустить имя элемента. Это правило эквивалентно последнему:

```
#catalog1234 { color: red; }
```

Вы также можете использовать селекторы идентификатора как часть контекстуального селектора. В этом примере стиль применяется только к элементам **li**, которые появляются внутри любого элемента, определенного как «links». Таким образом, вы можете обрабатывать пункты списка «links» отлично от всех других пунктов списка на странице без какой-либо дополнительной разметки.

```
#links li { margin-left: 10px; }
```

Вы поймете возможности селекторов и то, как они могут быть искусно использованы наряду с хорошо продуманной семантической разметкой.

Селекторы классов

Точка (.) указывает на селектор класса.

Это последний тип селекторов, а затем мы вернемся к свойствам текстовых стилей. Другим идентификатором элемента, о котором вы узнали в главе 5, является идентификатор **class**, используемый для группировки элементов в одну концептуальную группу. В отличие от атрибута **id**, имя класса **class** могут разделять несколько элементов. Кроме того, элемент может принадлежать более чем одному классу.

Универсальный селектор

В версии CSS2 был введен универсальный селектор элемента (*), который соответствует любому элементу (иными словами, является чем-то вроде джокера). Правило стилей

```
* {color: gray; }
```

сделает отображение *каждого* элемента в документе серым цветом. Этот селектор также полезен в качестве контекстуального, как показано в этом примере, который выбирает все элементы в разделе «intro»:

```
#intro * { color: gray; }
```

Универсальные селекторы вызывают проблемы с элементами форм в некоторых браузерах. Если ваша страница содержит элементы ввода внутри формы, самый безопасный выбор — избегать использования универсальных селекторов.

Вы можете целенаправленно выбирать элементы, принадлежащие одному и тому же классу при помощи селектора класса. Имена классов указываются при помощи точки (.) в селекторе. Например, чтобы выбрать все абзацы класса **class="special"**, используйте этот селектор (точка указывает, что следующее за ней слово является селектором класса):

```
p.special { color: orange; }
```

Чтобы применить свойство ко *всем* элементам одного класса, пропустите имя элемента в селекторе (убедитесь, что оставили точку; это символ, который указывает на класс). Так вы целенаправленно выберете все абзацы и любой другой элемент, который был размечен как **class="special"**.

```
.special { color: orange; }
```

Основы специфичности

В главе 11 я познакомила вас с понятием *специфичность*, которая имеет отношение к тому факту, что более специфичные селекторы имеют больший вес, когда приходится регулировать конфликты правил стилей. Теперь, так как вы узнали о существовании некоторых дополнительных селекторов, самое время пересмотреть этот очень важный принцип.

Реальная система, которую CSS использует для вычислений специфичности селектора, довольно запутана, но этот список типов селекторов, начиная с наиболее и заканчивая наименее специфичным, должен хорошо служить вам в большинстве случаев.

- Селекторы идентификатора более специфичны, чем (и заменяют)
- Селекторы классов, которые более специфичны, чем (и заменяют)

Подробнее о специфичности

Обзор специфичности, проведенный этой главе, достаточен для начала, но когда вы станете более опытными, а ваши таблицы стилей — более сложными, тогда вам потребуется более полное понимание внутреннего функционирования.

В статье «Спецификации CSS» по адресу **www.aranea.ru/css/doc/cssdoc61.php** вы можете ознакомиться с техническим объяснением вычислений специфичности. Эрик Мейер предоставляет полное, еще более легко усваиваемое описание этой системы в книге «CSS-каскадные таблицы стилей. Подробное руководство» (2-е издание, Символ-Плюс).

Также я рекомендую прочитать статью Энди Кларка «CSS: Войны специфичности», которая объясняет специфичность в терминах «силы Ситхов», героев фильма «Звездные войны» (**manuylov.wordpress.com/2009/03/01/specificity-wars/**).

- Контекстуальные селекторы, которые более специфичны, чем (и заменяют)
- Селекторы отдельных элементов

Так, например, если таблица стилей имеет два конфликтующих правила для элемента **strong**,

```
strong { color: red;}
h1 strong { color: blue; }
```

контекстуальный селектор (**h1 strong**) более специфичный, и поэтому имеет больший вес, чем селектор элемента.

Вы можете учитывать специфичность, чтобы сохранить таблицы стилей простыми, а разметку минимальной. Например можно установить стиль для элемента (в данном примере, **p**), затем заменить его при необходимости, используя более специфичные селекторы.

```
p { line-height: 1.2em; }
blockquote p { line-height: 1em; }
p.intro { line-height: 2em; }
```

В этих примерах элементы **p**, которые появляются внутри **blockquote**, имеют меньшую высоту строки, чем обычные абзацы. Однако все абзацы с классом «intro» будут иметь высоту строки в 2em, даже если абзац появится внутри блочной цитаты (**blockquote**), потому что селекторы классов специфичнее контекстуальных селекторов.

Понимание принципов наследования и специфичности важны для овладения CSS. Еще многое можно сказать на эту тему, и вы найдете полезные ссылки во врезке «Подробнее о специфичности».

Теперь вернемся к странице меню. К счастью, она была размечена тщательно и семантически, так что мы имеем много вариантов для выбора отдельных элементов. Предоставьте новым селекторам одну попытку в упражнении 12.6.

УПРАЖНЕНИЕ 12.6. ИСПОЛЬЗОВАНИЕ СЕЛЕКТОРОВ

На этот раз мы добавим правила стилей, используя селекторы потомков, идентификатора и классов в комбинации со свойствами шрифта и цвета, которые уже изучили.

1. Добавьте немного цвета к элементам «newitem» рядом с определенными названиями пунктов меню. Они размечены как **strong**, так что мы применим свойство цвета к элементу **strong**. Добавьте это правило к глобальной таблице стилей, сохраните файл и перезагрузите его в браузере.

```
strong { font-style: italic; color: maroon; }
```

Это сработало, но теперь элемент **strong** в описании острого блюда отображается тоже коричнево-малиновым цветом, а это не то, что я хотела.

Решением будет использование контекстуального селектора, целенаправленно выбирающего только элементы **strong**, которые появляются в элементах **dt**. Удалите из правила **strong** только что добавленное определение **color** и создайте новое правило стилей, целенаправленно выбирающее только элементы **strong**, которые появляются в элементах **dt**.

```
dt strong { color: maroon; }
```

2. Просмотрите исходный код документа, и вы увидите, что содержимое было разделено на три уникальных элемента **div**: **info**, **appetizers** и **entrees**. Мы можем использовать их, когда придет время стилизации. А сейчас давайте сделаем что-то простое и заставим весь текст в заголовке **info** отображаться сине-зеленым цветом. Из-за того, что цвет наследуется, нам только нужно применить свойство к элементу **div**, и оно передастся элементам **h1** и **p**.

```
#info { color: teal; }
```

3. Теперь, как настоящие эксперты, давайте изменим шрифт абзаца внутри заголовка на курсивный таким способом, который не повлияет на другие абзацы на странице. Снова решением станет использование контекстуального селектора. Это правило воздействует только на те абзацы, которые содержатся внутри раздела **header**.

```
#info p { font-style: italic; }
```

4. Теперь следует подвергнуть специальной обработке все цены в меню. К счастью, они были размечены элементами **span** примерно таким образом:

```
<span class="price">700 руб.</span>
```

Итак, сейчас все, что нам надо сделать — это написать правило, используя селектор класса для смены шрифта на Georgia или некоторый шрифт с засечками и для того, чтобы сделать его курсивным.

```
.price {
font-family: Georgia, serif;
font-style: italic;
color: gray;
}
```

5. Подобным образом следует изменить внешний вид текста в заголовке, который размечен как принадлежащий к классу «label», чтобы выделить его.

```
.label {
font-weight: bold;
font-variant: small-caps;
font-style: normal;
}
```

6. Наконец, в нижней части страницы есть предупреждение, которое нужно отобразить мелким шрифтом красного цвета. Предупреждению был задан класс «warning», поэтому можно использовать его как селектор для целенаправленного выбора только этого абзаца для стилизации.

Требуется применить один и тот же стиль к элементу **sup** (знак сноски), указанному ранее на странице, для обеспечения согласованности дизайна. Обратите внимание, что следует использовать сгруппированный селектор, так что не надо писать отдельное правило.

```
p.warning, sup {
font-size: x-small;
color: red;
}
```

На рис. 12.12 показаны результаты всех этих изменений.

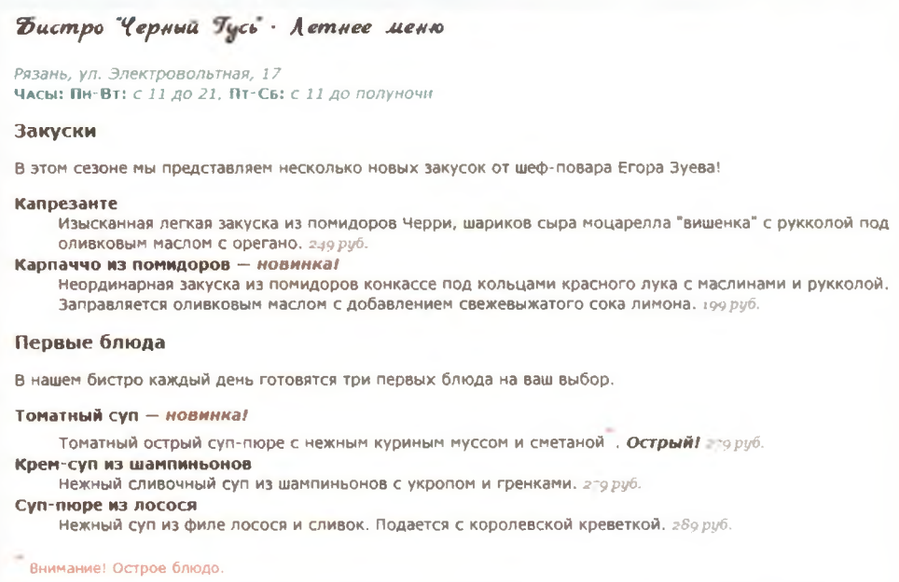


Рис. 12.12. *Текущее состояние страницы меню бистро «Черный Гусь»*

Выравнивание строк текста

Следующая группа свойств текста имеет дело не столько с формами символов, сколько с обработкой целых строк текста. Они позволяют веб-дизайнерам форматировать текст с помощью отступов, добавления интерлиньяжа (пространства между строками) и различных горизонтальных выравниваний подобно печатным документам.

Высота строки

Свойство **line-height** определяет минимальное расстояние между базовыми линиями строк текста. Мы уже встречали его как часть сокращенной записи свойства **font**. *Базовая линия* — воображаемая линия, на которой располагаются нижние части символов. Высота строки в CSS подобна межстрочному интервалу в традиционном типографском на-

боре. Хотя высота строки вычисляется согласно расстоянию между базовыми линиями, большинство браузеров распределяют дополнительное пространство выше и ниже текста, таким образом центрируя его по всей высоте строки (рис. 12.13).

Говорят, что свойство **line-height** задает только «минимальное» расстояние, потому что если вы разместите изображение в строке, высота ее увеличится согласно размерам рисунка.

line-height

Принимаемые значения: *число* | *значение длины* | *проценты* | **normal** | **inherit**

Значение по умолчанию: **normal**

Применение: ко всем элементам

Наследование: да

Эти примеры показывают три разных способа задать высоту строки вдвое больше размера шрифта.

```
p { line-height: 2; }
p { line-height: 2em; }
p { line-height: 200%; }
```

Когда задается только число, как показано в первом примере, оно действует как масштабный множитель, который умножается на текущий размер шрифта для вычисления значения высоты строки. Высота строк также может быть задана при помощи одной из единиц длины в CSS, но опять же, лучшим выбором является относительная единица измерения em. Единицы измерения em и процентные значения основываются на текущем размере шрифта. Во всех трех примерах: если размер шрифта текста равен 16 пикселам, вычисленная высота строки будет равна 32 пикселам (см. рис. 12.13).

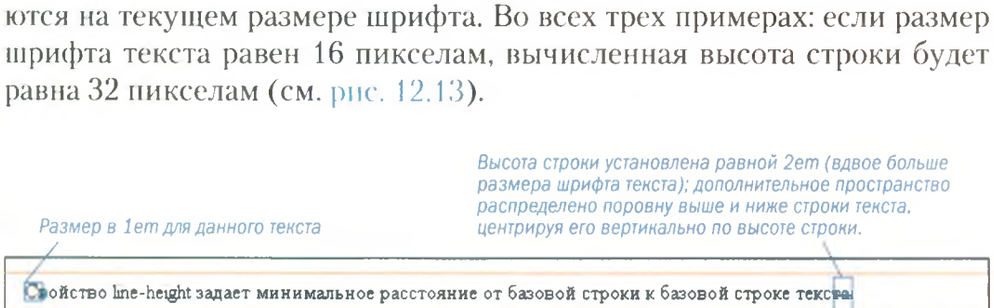


Рис. 12.13. *В CSS высота строки измеряется между базовыми линиями, но браузеры центрируют текст вертикально по высоте строки*

Отступы

Свойство **text-indent** задает отступ первой строки текста определенной ширины (см. примечание).

ПРИМЕЧАНИЕ

Свойство **text-indent** задает отступ только первой строки блока. Если вы хотите оставить пространство вдоль всей стороны блока текста, используйте для его добавления одно из свойств: **margin** или **padding**.

Дизайнеры могут задавать отступы и поля вместе, но согласно тому, как CSS обрабатывает их, поля будут рассматриваться как часть блочной модели в главе 16.

text-indent

Принимаемые значения: *значение длины* | *проценты* | **inherit**

Значение по умолчанию: 0

Применение: к блочным элементам и ячейкам таблицы

Наследование: да

Вы можете задать расстояние отступа или процентное значение для **text-indent**. Процентные значения вычисляются на основе ширины *родительского* элемента. Ниже приведено несколько примеров. Результаты показаны на рис. 12.14.

```
p#1 { text-indent: 2em; }
p#2 { text-indent: 25%; }
p#3 { text-indent: -35px; }
```

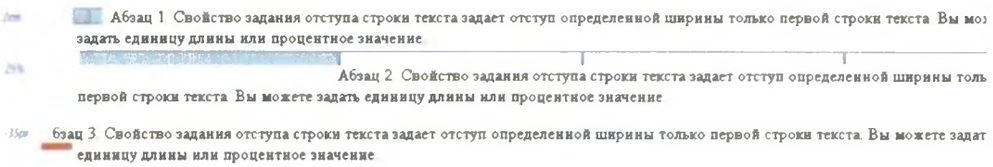


Рис. 12.14. Примеры применения свойства ***text-indent***

СОВЕТ ДИЗАЙНЕРА

Если вы используете обратный отступ, убедитесь, что у элемента еще есть левое поле. В противном случае выступающий текст может выйти за пределы левого края окна браузера.

Обратите внимание, что в третьем примере было задано отрицательное значение, и все в порядке. Таким образом первая строка текста будет расположена слева от самого левого края текста (это так называемый *обратный* или *отрицательный отступ*).

Свойство **text-indent** передается по наследству, но стоит отметить, что переходить к элементам-потомкам будут *вычисленные* значения. Так, если ширина элемента **div** установлена в 800 пикселов с 10%-ным отступом, то по наследству к элементам, содержащимся внутри элемента **div**, передастся свойство **text-indent** в 80 пикселов (а не значение 10%). Свойство **text-indent** также применяется к встроенным блокам.

Горизонтальное выравнивание

Вы можете выравнивать текст на веб-страницах при помощи свойства **text-align** так, как будто работаете в текстовом редакторе или программе предпечатной подготовки.

text-align

Принимаемые значения: **left** | **right** | **center** | **justify** | **inherit**

Значение по умолчанию: **left** для языков, в которых слова читаются слева направо, **right** — справа налево

Применение: к блочным элементам и ячейкам таблицы

Наследование: да

Это свойство используется довольно прямо. Результаты различных значений свойства **text-align** показаны на рис. 12.15.

text-align: left выравнивает текст по левому краю

text-align: right выравнивает текст по правому краю

text-align: center выравнивает текст по центру в текстовом блоке

text-align: justify выравнивает текст и по правому и левому краям (по ширине)

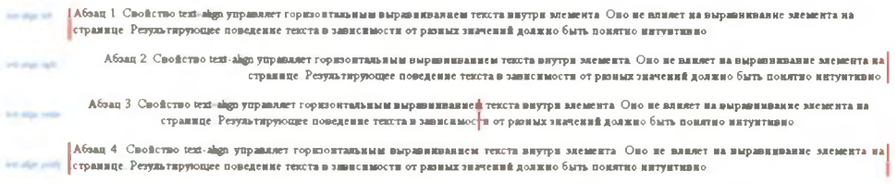


Рис. 12.15. Примеры применения значений свойства ***text-align***

Хорошие новости — осталось только четыре свойства текста! Затем вы будете готовы опробовать некоторые из них на странице меню бистро «Черный Гусь».

ПРИМЕЧАНИЕ

В текстовом модуле CSS3 указаны два новых связанных свойства — **text-align-last** (для выравнивания последней строки текста) и **textjustify** (для более точного контроля нам тем, как распределяются пробелы в выровненном по ширине тексте), однако на момент написания книги они недостаточно хорошо поддерживались браузерами.

Подчеркивания и другие декоративные эффекты

Если вы хотите разместить линию под или над текстом, зачеркнуть его или отключить подчеркивание ссылок, тогда свойство **text-decoration** для вас.

text-decoration

Принимаемые значения: **none** | **underline** | **overline** | **line-through** | **blink** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: нет, но после того, как проведена линия через дочерние элементы, они могут выглядеть так, как будто их тоже «украсили»

Результаты применения значений свойства **text-decoration** показаны на рис. 12.16.

text-decoration: underline подчеркивает элемент

text-decoration: overline рисует линию над текстом

text-decoration: line-through рисует линию через текст

text-decoration: blink делает текст мерцающим

Наиболее распространено использование свойства **text-decoration** для отключения линии, которая появляется автоматически под текстом ссылки, как показано здесь:

```
a { text-decoration: none; }
```

ПРИМЕЧАНИЕ

В текстовом модуле CSS3 содержатся улучшения для свойства **text-decoration**, в том числе свойства **text-decoration-line**, **text-decoration-color**, **text-decoration-style**, **text-decoration-skip** и **text-underline-position**, но на момент написания книги они еще находились в стадии разработки.

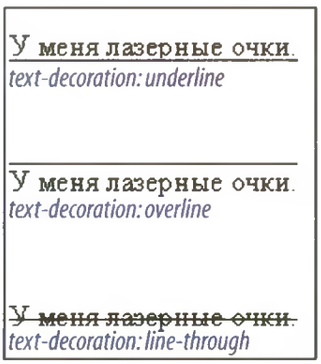


Рис. 12.16. Примеры применения значений свойства ***text-decoration***

Несколько предостережений относительно свойства **text-decoration**:

- Если вы убрали подчеркивание ссылок, убедитесь, что для их выделения используются другие визуальные эффекты, такие как цвет или насыщенность.
- Обратная сторона подчеркивания: из-за того, что подчеркивающие линии являются выделенными визуальными подсказками «Щелкните здесь», подчеркивание текста, который *не является* ссылкой, может вводить в заблуждение и мешать. Подумайте, может ли курсив стать допустимой альтернативой.
- Наконец, нет причины делать ваш текст мерцающим. В любом случае браузер Internet Explorer не поддерживает мерцание.

Изменение регистра

Я помню, когда в моей программе предпечатной подготовки появилась полезная функция, позволявшая менять регистр букв на ходу. Благодаря этому стало легко видеть, как мои заголовки могли бы выглядеть, если бы состояли только из заглавных букв, без необходимости перепечатывания. Таблицы CSS также включают такую особенность при помощи свойства **text-transform**.

text-transform

Принимаемые значения: **none** | **capitalize** | **lowercase** | **uppercase** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: да

Когда вы применяете свойство **text-transform** к текстовому элементу, оно меняет регистр его букв при отображении в браузере без изменения написания в исходном коде. Принимаемые значения следующие (рис. 12.17):

`text-transform: none`	как в исходном коде
`text-transform: capitalize`	преобразует первую букву каждого слова в заглавную
`text-transform: lowercase`	преобразует все буквы в строчные (нижний регистр)
`text-transform: uppercase`	преобразует все буквы в заглавные (верхний регистр)

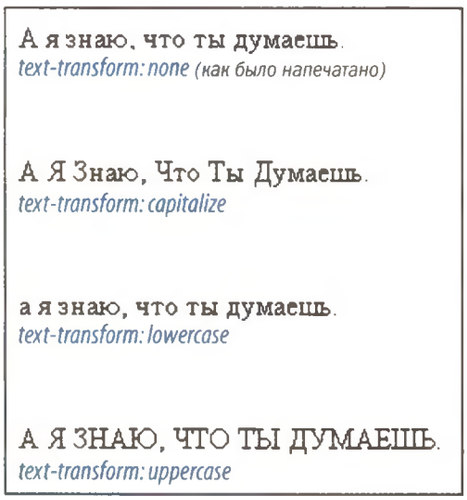


Рис. 12.17. Свойство `text-transform` *меняет регистр символов при отображении страницы в браузере независимо от того, как они напечатаны в исходном коде*

Кернинг и интервал между словами

Последние два текстовых свойства в этой главе используются для вставки интервала между знаками (`letter-spacing`) (другими словами, кернинг) или словами (`word-spacing`) при отображении текста.

`letter-spacing`

Принимаемые значения: *значение длины* | `normal` | `inherit`

Значение по умолчанию: `normal`

Применение: ко всем элементам

Наследование: да

`word-spacing`

Принимаемые значения: *значение длины* | `normal` | `inherit`

Значение по умолчанию: `normal`

Применение: ко всем элементам

Наследование: да

Функции свойств **letter-spacing** и **word-spacing** отражены в их именах: они добавляют интервал между знаками текста или словами в строке, соответственно. На рис. 12.18 показаны результаты этих примеров правил, примененных к простому абзацу, показанному здесь.

```
<p>Бистро "Черный Гусь" Летнее меню</p>
```

Пример 1

```
p { letter-spacing: 8px; }
```

Пример 2

```
p { word-spacing: 1.5em; }
```

Стоит отметить, что, когда вы задаете значение в единицах измерения em, вычисленный размер передается по наследству дочерним элементам, даже если они имеют меньший размер шрифта, чем у родительского.

В упражнении 12.7 мы последний раз вернемся к странице меню бистро «Черный Гусь», чтобы добавить некоторые из этих новых свойств и выполнить несколько улучшений.

*Рис. 12.18. Пример применения свойства **letter-spacing** (вверху) и **word-spacing** (внизу)*

Тень текста

Тени, придающие объем текстовым и графическим элементам на странице, стали в последнее десятилетие пиком моды. Теперь появился способ добавить к тексту тень, используя только CSS-свойство **text-shadow**. Тень накладывается позади текста, но перед фоном или границей, если таковые имеются.

Тени текста поддерживаются всеми современными браузерами.

text-shadow

Новый в CSS3

Принимаемые значения: *«смещение по горизонтали» «смещение по вертикали» «радиус размытия» «цвет»* | **none**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: да

Значение свойства **text-shadow** может содержать до трех значений (смещение по горизонтали, смещение по вертикали и необязательный радиус размытия), а также цвет. На рис. 12.19 показан пример минимального определения тени текста.

```
h1 {
color: darkgreen; color:
text-shadow: .2em .2em silver;
}
h1 {
darkgreen;
text-shadow: -.3em -.3em silver;}
```

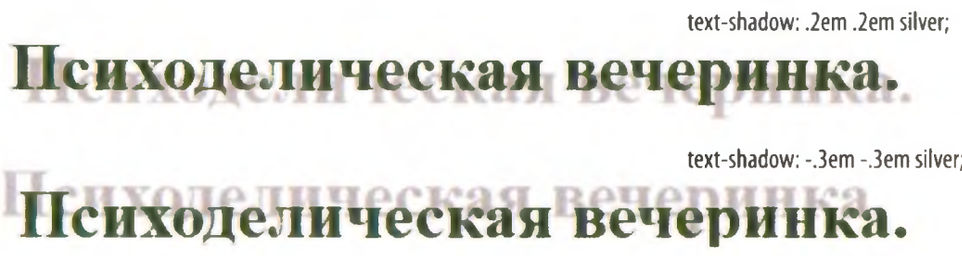


Рис. 12.19. Минимальная тень текста

Первое значение — это смещение по горизонтали, располагающее тень справа от текста (отрицательное значение перемещает тень *влево* от текста). Второе значение — смещение по вертикали, которое перемещает тень вниз на указанное расстояние (отрицательное значение перемещает ее *вверх*). Определение заканчивается указанием цвета (серебристый). Если цвет не указан, будет использоваться такой же, как и у текста.

Это должно дать вам представление о том, как работают первые два значения, но резкая тень выглядит не очень естественно. Что нам нужно, так это значение радиуса размытия. При нулевом значении (0) размытие отсутствует, а по мере увеличения значения усиливается (рис. 12.20). Как правило, приходится просто возиться со значениями, пока вы не добьетесь нужного эффекта.

Рис. 12.20. Добавление радиуса размытия к тени текста

Другие свойства текста

В интересах экономии места в книге и сохранения уровня «для начинающих», упомянутые свойства не были рассмотрены полностью. Но поскольку я из тех верстальщиков, которые «не прячут ничего за спиной», то все-таки опишу их ниже.

`vertical-align`

Принимаемые значения: **`baseline`** | **`sub`** | **`super`** | **`top`** | **`text-top`** | **`middle`** | **`textbottom`** | **`bottom`** | *проценты* | *длина* | **`inherit`**

Задает вертикальное выравнивание базовой линии встроенных элементов относительно базовой линии близлежащего текста. Также используется для задавания вертикального выравнивания содержимого в ячейке таблицы (**`td`**).

`white-space`

Принимаемые значения: **`normal`** | **`pre`** | **`nowrap`** | **`pre-wrap`** | **`pre-line`** | **`inherit`**

Определяет, как символ пробела обрабатывается в разметке.

Например, значение **`pre`** сохраняет символы пробела, найденные в исходном коде, и возвращает их подобно HTML-элементу **`pre`**.

`visibility`

Принимаемые значения: **`visible`** | **`hidden`** | **`collapse`** | **`inherit`**

Используется, чтобы спрятать элемент. Когда значение установлено в **`hidden`**, элемент невидим, но пространство, которое он занимает, сохраняется, оставляя пустую область в контенте. Элемент по-прежнему там, просто вы его не видите.

`text-direction`

Принимаемые значения: **`ltr`** | **`rtl`** | **`inherit`**

Задает направление чтения текста, слева направо (**`ltr`**) или справа налево (**`rtl`**).

`unicode-bidi`

Принимаемые значения: **`normal`** | **`embed`** | **`bidi-override`** | **`inherit`**

Относится к двунаправленным свойствам Unicode. В «рекомендациях» утверждается, что свойство позволяет верстальщику генерировать уровни внедрения внутри Unicode-алгоритма. Если вы понятия не имеете, что это значит, не беспокойтесь. Я тоже. Но догадываюсь, это необходимо для профессиональных многоязычных сайтов.

`font-size-adjust`

Новый в CSS3

Принимаемые значения: *число* | **`none`**

Это довольно запутанная новая система для задавания размера текстовых элементов на основе высоты-x (высоты строчной буквы «x») с целью обеспечения единообразия при использовании запасных шрифтов. Остальное объяснит консорциум Всемирной паутины по адресу: **www.w3.org/TR/css3-fonts/#font-size-adjustprop**.

Можно применить к одному элементу текста даже несколько теней. При их перечислении сначала отображается та, которая указана первой в списке, а последующие тени накладываются на нее в указанном порядке. Вы также можете добавить тексту свечение, расположив цветную тень непосредственно за текстом. На рис. 12.21 показано несколько способов применения свойства **text-shadow**.

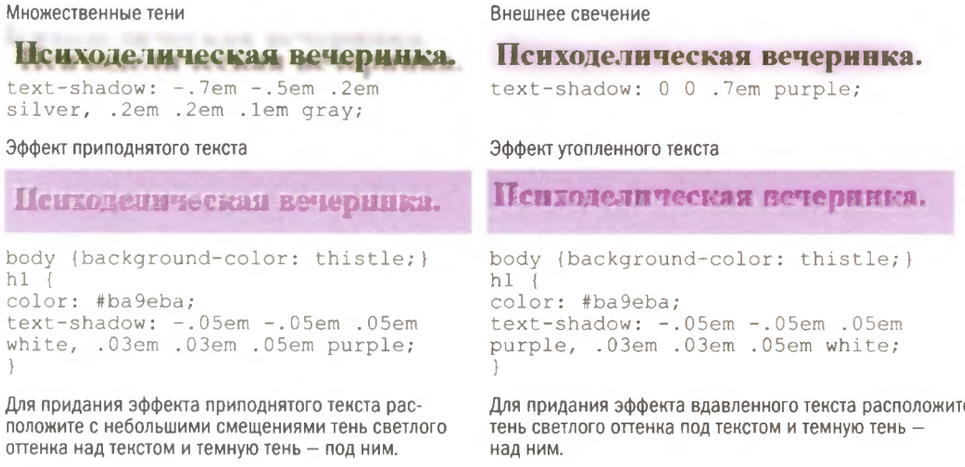


Рис. 12.21. Спецэффекты, созданные с помощью тени текста

Стоит отметить, что тени могут не только затруднить чтение текста. Их добавление ко всему тексту подряд может замедлить производительность страницы (прокрутка, отклик на щелчки мыши, и т. д.), что особенно проблематично для мобильных браузеров с небольшой мощностью обработки информации. Кроме того, убедитесь, что к вашему тексту не требуется добавлять тень, чтобы сделать его видимым. Пользователи браузеров, не поддерживающих это свойство, ничего не увидят. Мой совет — используйте тени как улучшение, чтобы, если они не отобразятся, это не было важно.

СОВЕТ ДИЗАЙНЕРА

Изменение кернинга мелкого шрифта — один из моих любимых приемов оформления заголовков. Это хорошая альтернатива крупному шрифту для обращения внимания на элемент.

УПРАЖНЕНИЕ 12.7. ЗАВЕРШАЮЩИЙ ЭТАП РАБОТЫ НАД СТРАНИЦЕЙ МЕНЮ

Сейчас мы добавим несколько завершающих штрихов к странице меню, *menu_summer.html*. Вероятно, будет полезным сохранять файл и обновлять его в браузере после каждого шага, чтобы видеть результаты изменений и убеждаться, что вы делаете успехи. Завершенная таблица стилей предоставлена в приложении А.

1. Во-первых, нужно выполнить несколько глобальных изменений элемента **body**. Я передумала по поводу **font-family**. Следует использовать шрифт с засечками, такой как Georgia, который будет более современным и подходящим для меню бистро. Также используйте свойство **line-height**, чтобы «раздвинуть» строки текста для более легкого прочтения. Внесите эти обновления в правило стилей элемента **body**, как показано здесь:

```
body {
font-family: Georgia, serif;
font-size: small;
line-height: 1.75em;
}
```

2. Переконструируйте раздел заголовка документа. Уберите определение сине-зеленого цвета, удалив целое правило. Как только это будет выполнено, сделайте элемент **h1** фиолетовым, а абзац в заголовке — серым. Вы можете просто добавить определения цвета к существующим правилам.

```
#header { color: teal; } /* удаляем */

h1 {

font: bold 1.5em "Marck Script", Georgia,
serif;

color: purple;}

#info p {

font-style: italic;

color: gray;}
```

3. Далее, для имитации необычно напечатанного меню, центрируйте несколько ключевых элементов на странице, используя свойство **text-align**. Напишите правило при помощи сгруппированного селектора для центрирования всех заголовков и раздела **#info**, наподобие этого:

```
h1, h2, #info {

text-align: center;}
```

4. Сделайте заголовки **h2** «appetizers» и «entrees» особенными. Вместо крупного полужирного шрифта оформите все буквы в верхнем регистре, измените кернинг и цвет для акцентирования заголовков. Ниже приведено новое правило для элементов **h2**, которое включает все эти изменения.

```
h2 {

font: bold 1em;

text-transform: uppercase;

letter-spacing: 5em;

color: purple;}
```

5. Мы почти закончили, еще несколько настроек абзацев, расположенных после элементов заголовков **h2**. Давайте их также центрируем и сделаем курсивными.

```
h2 + p {

text-align: center;

font-style: italic; }
```

Обратите внимание, что я применила селектор смежных элементов одного уровня (**h2 + p**), чтобы выбрать «любой абзац, следующий за элементом **h2**».

6. Теперь добавьте более мягкий цвет названиям пунктов меню (в элементах **dt**). Я выбрала охру (**sienna**), одно из имен из цветового модуля CSS3. Обратите внимание, что элементы **strong** в тех элементах **dt** остаются коричнево-малиновыми, потому что цвет, примененный к элементам **strong**, заменяет цвет, унаследованный от их родителей.

```
dt {

font-weight: bold;

color: sienna;}
```

7. Наконец, для красоты добавьте тень под заголовком **h1**.

```
h1 {

font: bold 1.5em "Marko

One", Georgia, serif;

color: purple;

text-shadow: .1em .1em .2em
lightslategray;}
```

На этом все! На рис. 12.22 показано, как теперь выглядит меню. Если сравнивать с версией без стилей, это существенный прогресс, хотя мы использовали только свойства текста. Обратите внимание, что в процессе мы не затронули ни одного элемента разметки документа. В этом вся прелесть содержания стилей отдельно от структуры.


Бистро "Черный Гусь" · Летнее меню

Рязань, ул. Электрозаводская, 27

Часы: Пн-Вт: с 11 до 23, Пт-Сб: с 11 до полуночи

ЗАКУСКИ

В этом сезоне мы представляем несколько новых закусок от шеф-повара Егора Зуева!

Капрезанте

Изысканная легкая закуска из помидоров Черри, шариков сыра моцарелла "вишенка" с рукколой под оливковым маслом с орегано. 149 руб.

Карпаччо из помидоров — новинка!

Неординарная закуска из помидоров конкассе под кольцами красного лука с маслинами и рукколой. Заправляется оливковым маслом с добавлением свежевыжатого сока лимона. 199 руб.

ПЕРВЫЕ БЛЮДА

В нашем бистро каждый день готовятся три первых блюда на ваш выбор.

Томатный суп — новинка!

Томатный острый суп-пюре с нежным куриным мусом и сметаной. *Острый!* 179 руб.

Крем-суп из шампиньонов

Нежный сливочный суп из шампиньонов с укропом и гренками. 179 руб.

Суп-пюре из лосося

Нежный суп из филе лосося и сливок. Подается с королевской креветкой. 249 руб.

* Внимание! Острое блюдо.

Рис. 12.22. Отформатированная страница меню бистро «Черный Гусь»

Изменение маркеров и нумерации списков

Перед тем как мы закончим эту главу, посвященную свойствам текста, я хочу показать несколько хитростей, которые можно применить к маркированным и нумерованным спискам. Как вы знаете, браузеры автоматически указывают маркеры перед пунктами маркированного списка и числа перед пунктами в нумерованных списках. В большинстве случаев отображение этих маркеров определяется браузером. Однако CSS предоставляет несколько свойств, которые позволяют верстальщикам выбирать тип и положение маркера или заменять его специальным изображением.

Выбор маркера

Используйте свойство **list-style-type**, чтобы выбрать тип маркера, который появится перед каждым пунктом списка.

list-style-type

Принимаемые значения: **none** | **disc** | **circle** | **square** | **decimal** | **decimal-leading-zero** | **lower-alpha** | **upper-alpha** | **lower-latin** | **upper-latin** | **lower-roman** | **upper-roman** | **lower-greek** | **inherit**

Значение по умолчанию: **disc**

Применение: **ul**, **ol** и **li** (или элементы, у которых значением свойства отображения является **list-item**)

Наследование: да

Чаще всего веб-дизайнеры используют значение **none** свойства **list-style-type**, чтобы отключить маркер. Это полезно при использовании семантически размеченного списка как основы для горизонтального меню навигации или полей ввода данных форм. Вы можете сохранить семантику, но избавиться от надоедливых маркеров.

Значения **disc**, **circle** и **square** генерируют формы маркеров точно так, как браузеры делали с самого начала (рис. 12.23). К сожалению, нет способа изменения внешнего вида (размера, цвета и т. д.) генерируемых маркеров, так что вы в основном привязаны к их виду, по умолчанию используемому в браузерах.

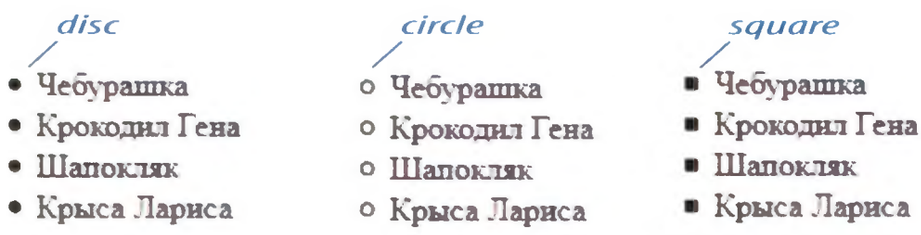


*Рис. 12.23. Значения **disc**, **circle** и **square** свойства **list-style-type***

ПРИМЕЧАНИЕ

Этот раздел описывает типы свойства **list-style** спецификации CSS2.1, которые хорошо поддерживаются современными браузерами. CSS3 расширяют функциональность маркеров, описанных здесь, добавляя метод, позволяющий верстальщикам самим определять стили списка и вводить нумерацию на многих языках (**www.w3.org/TR/css3-lists/**).

ПРИМЕЧАНИЕ

Спецификация CSS3 вводит новые типы маркеров «квадрат», «флажок», «ромб» и «тире», используя новое правило **@counter-style**. Подробнее об этом читайте в спецификации.

Типы отображения элементов списка

Возможно, вы заметили, что свойства стилей списка применяются к элементам, отображаемое значение которых — элемент списка. Спецификация CSS2.1 позволяет любому элементу функционировать как элемент списка, присвоив значение **listitem** свойству **display**.

Это свойство может быть применено к любому элементу HTML или другого языка XML. Например, можно автоматически добавить маркеры или числа к нескольким абзацам, задав значение **listitem** свойства **display** для элементов абзаца (**p**), как показано в примере:

```
p.bulleted {
display: list-item;
list-style-type:
upperalpha;
}
```

Оставшиеся зарезервированные слова (табл. 12.1) задают различные стили цифр и букв для использования в упорядоченных списках.

Табл. 12.1. Зарезервированные слова нумерации списков в CSS2.1

Зарезервированное слово	Система
decimal	1, 2, 3, 4, 5 ...
decimal-leading-zero	01, 02, 03, 04, 05...
lower-alpha	a, b, c, d, e...
upper-alpha	A, B, C, D, E...
lower-latin	a, b, c, d, e... (то же, что **lower-alpha**)
upper-latin	A, B, C, D, E... (то же, что **upper-alpha**)
lower-roman	i, ii, iii, iv, v...
upper-roman	I, II, III, IV, V...
lower-greek	α, β, γ, δ, ε...

Положение маркера

По умолчанию маркер располагается с внешней стороны пункта списка, отображаясь как отрицательный отступ. Свойство **list-style-position** позволяет вам убрать маркер внутрь области контента так, что он оказывается частью контента списка.

list-style-position

Принимаемые значения: **inside** | **outside** | **inherit**

Значение по умолчанию: **outside**

Применение: **ul**, **ol** и **li** (или элементы, у которых значением свойства отображения является **list-item**)

Наследование: да

ПРИМЕЧАНИЕ

Спецификация CSS3 добавляет к этому свойству значение **hanging**. Оно похоже на значение **inside**, но маркеры должны появиться снаружи, примыкая к левому краю затененной области, как показано на рис. 12.24.

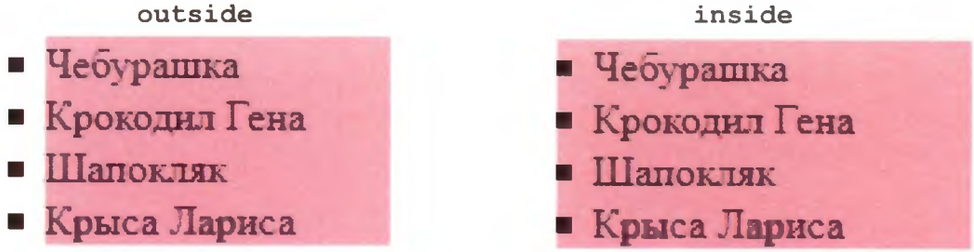


Рис. 12.24. Применение свойства ***list-style-position***

Я применила цвет фона к пунктам списка на рис. 12.24, чтобы показать пределы их блоков области контента. Вы можете видеть, что, когда

положение установлено в **outside** (рисунок слева), маркеры располагаются вне области контента, а когда оно установлено в **inside**, блок области контента расширяется и включает в себя маркер.

```
li {background-color: #F99;}
ul#outside {list-style-position: outside;}
ul#inside {list-style-position: inside;}
```

Создание пользовательских маркеров

Вы также можете использовать ваше собственное изображение в качестве маркера при помощи свойства **list-style-image**.

list-style-image

Принимаемые значения: *URL-адрес* | **none** | **inherit**

Значение по умолчанию: **none**

Применение: **ul**, **ol** и **li** (или элементы, у которых значением свойства отображения является **list-item**)

Наследование: да

Значением свойства **list-style-image** является URL-адрес изображения, которое вы хотите использовать в качестве маркера. Свойство **list-style-type** установлено в значение **disc** как резервный вариант на тот случай, если изображение не отобразится, или свойство не поддерживается браузером или другим пользовательским агентом. Результат показан на рис. 12.25.

```
ul {
list-style-image: url('images/pic.gif');
list-style-type: circle;
list-style-position: outside;
}
```

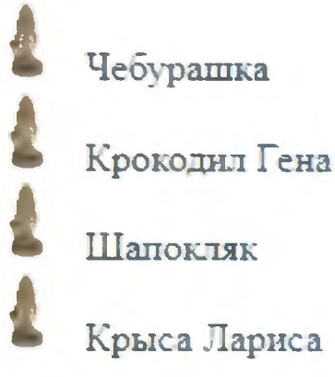


Рис. 12.25. Использование изображения в качестве маркера

ПРИМЕЧАНИЕ

Существует сокращенное свойство **list-style**, объединяющее в себе значения типа, положения и изображения в любом порядке. Например:

```
ul { list-style:
url('images/pic.gif')
circle outside; }
```

При использовании сокращенных свойств будьте осторожны, чтобы не заменить стилевые свойства списков, перечисленные ранее в таблице стилей.

Резюме

В этой главе мы охватили свойства, используемые для форматирования текстовых элементов. Здесь приведена их сводка в алфавитном порядке.

Свойство	Описание
font	Кратко записанное свойство, которое объединяет свойства шрифта
font-family	Задает гарнитуру или семейство типовых шрифтов
font-size	Размер шрифта
font-style	Задает курсивный или наклонный шрифты
font-variant	Задает капительный шрифт
font-weight	Задает насыщенность шрифта
letter-spacing	Вставляет интервал между знаками (кернинг)
line-height	Расстояние между базовыми линиями соседних текстовых строк
text-align	Горизонтальное выравнивание текста
text-decoration	Подчеркивания, надчеркивания и зачеркивания
text-direction	Читает ли текст слева направо или справа налево
text-indent	Величина отступа первой строки в блоке
text-shadow	Добавляет тень к тексту
text-transform	Меняет регистр букв текста при его отображении
unicode-bidi	Работает с двунаправленными алгоритмами Unicode
vertical-align	Устанавливает вертикальное положение встроенного в строку элемента относительно базовой линии
visibility	Устанавливает, отображается элемент или невидим
white-space	Определяет, как отображается символ пробел в исходном коде
word-spacing	Вставляет интервал между словами

ГЛАВА 13

ЦВЕТА И ФОН (ВКЛЮЧАЯ СЕЛЕКТОРЫ И ВНЕШНИЕ ТАБЛИЦЫ СТИЛЕЙ)

Вам случалось посещать Всемирную паутину в 1993 году? В то время это было довольно скучно, так как фон каждой страницы был серым, а весь текст — черным. Затем появился браузер Netscape, а вместе с ним — небольшое количество атрибутов, которые (наконец-то) позволяли элементарно управлять цветами шрифта и фоном. Такая ситуация сохранялась довольно долго. Но теперь, к счастью, у нас есть свойства таблиц стилей, которые вытеснили эти старые атрибуты.

В этой главе я представлю вам все свойства для указания цветов и фона, а также расскажу о дополнительных типах селекторов и продемонстрирую, как создавать внешние таблицы стилей и таблицы стилей для печати.

Сначала поговорим о том, как задавать цвет в CSS, и о природе цвета в компьютерных мониторах.

В этой главе

- Имена цветов в CSS
- Значения цветов модели RGB
- Основной и фоновый цвета
- Селекторы псевдоклассов и псевдоэлементов
- Замощение фоновых изображений
- Цветовые градиенты
- Внешние таблицы стилей

Определение значений цвета

Существует два основных способа задать цвет в таблицах стилей — при помощи предопределенных имен цветов, как мы уже делали:

```
color: red;   color: olive; color: blue;
```

или, что более распространено, при помощи числового значения, которое описывает конкретный *цвет RGB* (модель цвета в компьютерных мониторах). Вы, вероятно, видели значения, которые выглядели как эти:

```
color: #FF0000;      color: #808000;      color: #00F;
```

Вы узнаете все о цветах в формате RGB через минуту, но для начала короткий и интересный раздел о стандартных именах цветов.

Имена цветов

ПРИМЕЧАНИЕ

Расширенные имена цветов, также известные как *имена цветов X11*, изначально были введены системой X Window System для Unix.

Наиболее интуитивный способ задать цвет — это указать его имя. К несчастью, вы не можете просто выдумать любое имя цвета, чтобы тот отобразился на странице. Оно должно быть одним из зарезервированных, предопределенных в рекомендации CSS. Спецификации CSS1 и CSS2 по наследству получили 16 стандартных имен цветов, изначально введенных в HTML 4.01. Спецификация CSS2.1 была дополнена именем **orange**, и их число увеличилось до 17 (рис. 13.1). В CSS3 добавлена поддержка расширенного набора из 130 имен цветов и всего их стало 147. Расширенный список имен цветов представлен на рис. 13.2, но если вы любознательны, то можете просмотреть полный список в файле ***Имена цветов.pdf*** на диске, прилагающемся к этой книге.

Рис. 13.1. 17 стандартных имен цветов в CSS2.1

Имена цветов легко использовать — просто вставьте одно из них в качестве значения для любого свойства, связанного с цветом:

```
color: silver;
background-color: gray;
border-bottom-color: teal;
```

Рис. 13.2. Расширенный список из 140 новых имен цветов в CSS3. Помните, что на экране они будут выглядеть иначе

Значения цветов в формате RGB

Имена просты, но, как вы можете видеть, их количество ограниченно. Общепризнанно, что наиболее распространенный способ задать цвет — указать его значение в цветовой модели RGB. Так вам становится доступен выбор из миллионов оттенков.

Для тех, кто не знает, как компьютеры работают с цветом, я начну с основ, прежде чем погружаться в синтаксис CSS.

Несколько слов о цветах RGB

Оттенки, которые вы видите на мониторе, производятся смешиванием света трех цветов: красного, зеленого и синего. Этот способ известен как *цветовая модель RGB*. Вы можете предоставлять своего рода «рецепты» оттенков, сообщая компьютеру, какое количество каждого цвета подмешать. Количество света в каждом цветовом «канале» обычно описывается по шкале чисел от 0 (нет) до 255 (максимально полно), хотя оно также может быть задано как процентное значение. Чем ближе эти три значения к 255 (100%), тем ближе к белому будет итоговый оттенок (рис. 13.3).

Любой цвет, который вы видите на мониторе, может быть описан последовательностью из трех чисел: значением красного, зеленого и синего цветов. Это один из способов, которым графические редакторы, такие как Adobe Photoshop, создают оттенок каждого пиксела в изображении. При помощи цветовой модели RGB приятный бледно-лиловый цвет может быть описан как 200, 178, 230.

Подбор цвета

Самый легкий способ подобрать цвет и найти его значение в модели RGB — это использовать графический редактор, такой как Adobe Photoshop, Adobe Fireworks или Corel PaintShop Pro. Большинство подобных программ содержат *палитру цветов*, такую как на рис. 13.4. Если у вас нет графического редактора, цвет можно выбрать из онлайн-палитры, например, **ColorPicker.com** (Рис. 13.4).

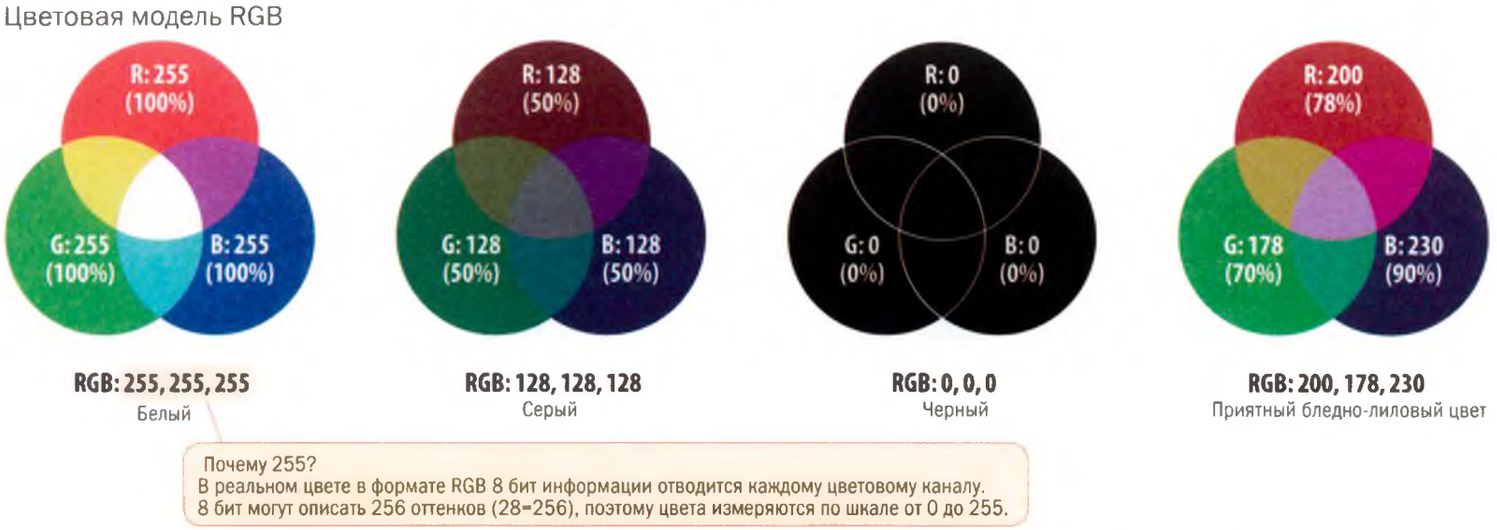
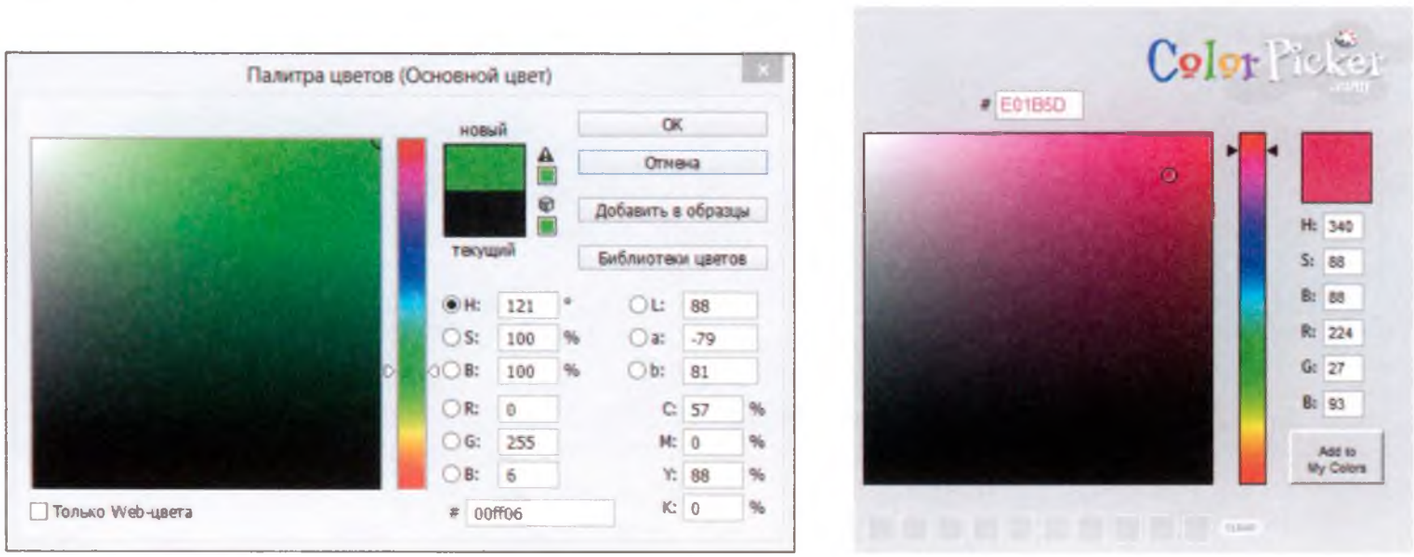


Рис. 13.3. *Цвета на компьютерных мониторах образуются смешиванием различного количества красного, зеленого и синего света (отсюда название цветовой модели — RGB*). Цвет в центре каждой диаграммы показывает, что происходит, когда соединяются три цветовых канала. Чем больше света в каждом канале, тем ближе сочетание к белому цвету*

* RGB — аббревиатура английских слов «red» (красный), «green» (зеленый), «blue» (синий).

Существует несколько способов определения цвета на мониторах. Два из них, имеющие отношение к CSS — RGB и HSL*. Цветовая модель RGB наиболее часто используется и хорошо поддерживается, поэтому мы сосредоточимся на ней. Прочитайте врезку «Цвета HSL», чтобы узнать больше об альтернативном варианте.

Когда вы выбираете цвет из спектра в палитре цветов, значения красного, зеленого и синего перечислены, как показано на рис. 13.4. Обратите внимание, рядом с символом # указаны те же самые значения, преобразованные в шестнадцатеричные эквиваленты и готовые к использованию в таблице стилей. Я расскажу о шестизначных шестнадцатеричных значениях далее.



Палитра цветов программы Photoshop

Colorpicker.com

Рис. 13.4. *Палитры цветов, такие как в программе Adobe Photoshop (слева) и Colorpicker.com (справа) предоставляют RGB значения для выбранного цвета*

Задавание значений цвета RGB в таблицах стилей

Спецификация CSS позволяет задавать значения цвета RGB рядом форматов. Возвращаясь к приятному бледно-лиловому цвету, мы могли добавить его к таблице стилей перечислением каждого значения по шкале от 0 до 255.

```
color: rgb(200, 178, 230);
```

Допустимо также перечислить их в виде процентных значений, хотя это менее распространенный способ.

```
color: rgb(78%, 70%, 90%);
```

А еще можно использовать значение цвета для Всемирной паутины, которое отображается в палитре цветов. Эти 6 цифр представляют собой

НА ЗАМЕТКУ

Указание значений цвета RGB

В CSS существуют четыре формата задавания значений цвета RGB:

```
rgb (255, 255, 255)
rgb (100%, 100%, 100%)
#FFFFFF
#FFF
```

Все эти примеры задают белый цвет.

* HSL — аббревиатура английских слов «hue» (цветовой тон), «saturation» (насыщенность), «lightness» (светлота).

Цвета HSL

Спецификация CSS3 предоставляет возможность указывать цвета в значениях HSL: цветовой тон (hue), насыщенность (saturation) и светлота (lightness).

В палитре цветов на рис. 13.4 также представлены значения HSL для выбранного цвета, хотя последнее значение здесь называется яркость (brightness, B).

В этой системе цвета распределяются по кругу в той последовательности, как они представлены в радуге. В верхней части (на 12 часов) находится красный цвет. Значения цветового тона затем измеряются в градусах по кругу: красный — 0 °, зеленый — 120 ° и синий — 240 ° Другие цвета расположены между ними. Насыщенность определяется в процентных значениях от 0% (серый) до 100% (цвет в полной мере). Светлота (или яркость) также является процентным значением от 0% (самый темный) до 100% (самый светлый).

Некоторые люди считают, что эта система интуитивно понятна в использовании потому, что выбрав цвет, легко сделать его более насыщенным, светлым или темным.

Значения RGB не столь понятны, хотя у некоторых опытных дизайнеров развивается к ним чутье.

В CSS цвета HSL представлены в виде значения оттенка и двух процентных значений.

Они никогда не преобразуются в шестнадцатеричные значения, как это можно сделать с RGB. Ниже представлено, как будет обозначаться лавандовый оттенок, показанный на рис. 13.3, в таблице стилей с использованием цветов HSL:

```
color: hsl(265, 23%, 90%);
```

те же три значения RGB, только преобразованные в шестнадцатеричные. Я объясню эту систему счисления в следующем разделе. Обратите внимание, что шестнадцатеричным значениям RGB предшествует символ # и не требуется запись `rgb()`, показанная выше. Можно использовать буквы в верхнем или нижнем регистре, но рекомендуется придерживаться одного формата.

```
color: #C8B2E6;
```

Существует еще один простой способ указать шестнадцатеричное значение цвета. Если оно оказалось составленным из трех пар дублированных цифр, таких как:

```
color: #FFCC00; или color: #993366;
```

вы можете вместо каждой пары указать одну цифру. Эти примеры эквивалентны тем, что перечислены выше:

```
color: #FC0; или color: #936;
```

О шестнадцатеричных значениях

Пора прояснить, что происходит с 6-разрядной строкой символов. То, на что вы смотрите, на самом деле последовательность трех двузначных чисел, по одному для красного, зеленого и синего цветов. Но вместо десятичной (система счисления по основанию 10, которую мы используем), эти значения указаны в шестнадцатеричной системе, или по основанию 16. На рис. 13.5 показана структура шестнадцатеричного значения RGB.

Рис. 13.5. *Шестнадцатеричные значения цвета RGB составлены из трех двузначных чисел — по одному для красного, зеленого и синего цветов*

Шестнадцатеричная система счисления использует 16 символов: цифры 0–9 и буквы A–F (для представления чисел 10–15). На рис. 13.6 показано, как это работает. Шестнадцатеричная система широко используется в вычислительной технике, потому что уменьшает объем определенной информации. Например, значения цветов RGB уменьшаются с трех до двух цифр, когда они преобразованы в шестнадцатеричный формат.

В настоящее время, когда большая часть графических редакторов и программ для верстки веб-страниц предоставляет доступ к шестнад-

цатеричным значениям цвета (как мы видели на рис. 13.4), нет особой необходимости самостоятельно переводить значения цветов RGB в шестнадцатеричные, как это приходилось делать раньше. Но если возникнет такая потребность, выноска «Преобразование в шестнадцатеричные значения» должна вам помочь.

Рис. 13.6. Шестнадцатеричная система счисления имеет основание 16

Цвета RGBa

Цвета RGBa позволяют указать цвет, а также сделать его настолько прозрачным, насколько вам нужно. Буква «a» в аббревиатуре RGBa означает *альфа*, то есть дополнительный канал, который контролирует уровень прозрачности по шкале от 0 (полностью прозрачный) до 1 (полностью непрозрачный). Ниже показано, как это выглядит в правиле стилей:

```
color: rgba(0, 0, 0, .5);
```

Первые три цифры в скобках — это привычные значения RGB, в данном случае создающие черный цвет. Четвертое значение — `.5` — это уровень прозрачности. В итоге получается черный цвет с прозрачностью 50%. Другие цвета или фоновые узоры смогут немного просматриваться сквозь него (рис. 13.7).

Заголовок 1. `h1 {color: rgba(0, 0, 0, .1);}`

Заголовок 2. `h2 {color: rgba(0, 0, 0, .5);}`

Заголовок 3. `h3 {color: rgba(0, 0, 0, 1);}`

Рис. 13.7. Заголовки с различными уровнями прозрачности, созданные с использованием значений RGBa

СОВЕТ

Полезные шестнадцатеричные значения

Белый = **#FFFFFF** или **#FFF** (эквивалент 255,255,255)

Черный = **#000000** или **#000** (эквивалент 0,0,0)

Преобразование в шестнадцатеричные значения

В операционной системе Windows в стандартном калькуляторе есть шестнадцатеричный конвертор в инженерном режиме (**Вид** ⇒ **Инженерный**). Пользователи OS X могут загрузить бесплатный калькулятор Mac Dec BinCalculator по адресу **versiontracker.com**.

Чтобы вычислить шестнадцатеричное значение самостоятельно, нужно разделить ваше число на 16. Целый результат будет первой цифрой, а неделимый остаток — второй. Например, 200 преобразуется в C8, потому что 200=(16*12)+8. Это {12,8} по основанию 16 или C8 в шестнадцатеричном формате.

Думаю, лучше пользоваться палитрой цветов.

ПРИМЕЧАНИЕ

Цветам HSL также можно придать определенный уровень прозрачности, используя цветовой формат HSLa, который имеет тот же синтаксис, что и цвета RGBa:

```
color: hsla(0, 0%, 0%, .5);
```

Однако существует проблема с браузером Internet Explorer версии 8 и более ранних — они не поддерживают цвета RGBa, поэтому вам необходимо предоставить резервный вариант для пользователей этих браузеров. Проще всего просто выбрать полностью непрозрачный цвет, приблизительно похожий на тот, который вам нужен, и в правиле стилей указать его первым. Браузер Internet Explorer проигнорирует значение RGBA, а в других браузерах непрозрачный цвет будет отменен, когда задействуется второе правило определения.

```
h1 {
color: rgb(120, 120, 120);
color: rgba(0, 0, 0, .5);
}
```

Но если вам просто *необходимо* добиться прозрачности в браузере IE, можно предоставить альтернативы (прозрачный рисунок PNG или собственный фильтр Internet Explorer) специально для версий 6–8, заключив правила или элементы стиля в *условные комментарии*, понятные

Ориентация на браузер Internet Explorer с помощью условных комментариев

Синтаксис условных комментариев в Internet Explorer предоставляет возможность определить стиль только для данного браузера или даже конкретной его версии. Другие браузеры проигнорируют все, что указано в комментариях, но Internet Explorer применит все стили, которые там найдет. Условные комментарии могут находиться внутри таблицы стилей или, как показано в примере ниже, использоваться для предоставления отдельных глобальных таблиц стилей с помощью элемента **`style`**. Убедитесь, что условные комментарии указаны после обычных правил стилей. Используя запасной цвет RGBa в качестве примера, этот условный комментарий ориентирован на браузер Internet Explorer версии 8 или более ранней (**`if lte IE 8`**) и применяет в качестве фона элемента **`p`** файл PNG с прозрачностью 50%. (Файлы PNG с прозрачностью обсуждаются в главе 19.)

```
<!--[if lte IE 8]>
<style>
p {background: transparent url(black-50.png);}
</style>
<![endif]-->
```

Вы должны знать, что отношение к использованию условных комментариев в сообществе веб-разработчиков довольно спорное. Некоторые избегают их любой ценой, взамен выбирая сценарии JavaScript. Другие считают такие комментарии подходящими для работы и не переживают, что они не являются строго действительной разметкой. Надеюсь, в один прекрасный день, когда старые версии браузера Internet Explorer окончательно выйдут из употребления, эта техника больше не пригодится.

только браузеру Internet Explorer (см. врезку «Ориентация на браузер Internet Explorer с помощью условных комментариев»). К счастью, цвета RGBa поддерживаются в браузере Internet Explorer 9 и более новых, и поскольку старые версии выходят из употребления, скоро вам не нужно будет совершать лишние действия.

Суммирование значений цвета

Чтобы добраться до этого раздела, вам пришлось прочитать несколько страниц, но процесс подбора и задавания цветов в таблицах стилей на самом деле прост.

- Выбрать одно из заданных имен цветов

  или

- Использовать таблицу цветов для выбора цвета и записать значения RGB (желательно 6-разрядные шестнадцатеричные). Вставить их в правило стилей, используя один из четырех форматов значений цветов RGB, и все готово. Или можно использовать цвета HSL, если вы работаете с ними.

Существует еще один способ цветовой заливки элемента, и это — градиенты (цвета, переходящие из одного оттенка в другой), но он затрагивает множество проблем (а именно: вендорных префиксов), которых я не хочу касаться прямо сейчас, поэтому прибережем градиенты CSS на конец главы.

Основной цвет

Теперь, когда вы знаете, как указывать значения цвета, давайте перейдем к связанным с ним свойствам. Вы можете задавать основные и фоновые цвета для любого HTML-элемента. Существуют также свойства, задающие цвет границ, но они будут рассмотрены в главе 14.

Основной цвет элемента состоит из его текста и границ (если они определены). Основной цвет задается свойством `color`, как мы видели в предыдущей главе, когда рассматривали его, чтобы сделать текст привлекательным. Приведем еще раз детали свойства `color`.

`color`

Принимаемые значения: *значение цвета (имя или число)* | `inherit`

Значение по умолчанию: зависит от браузера и предпочтений пользователя

Применение: ко всем элементам

Наследование: да

В следующем примере основной цвет переднего плана элемента `blockquote` указан как приятный зеленый посредством значений

Веб-палитра

Вы наверняка встретитесь с понятиями веб-палитры или «безопасных» веб-цветов, читая материалы о веб-дизайне или используя такие программы, как Dreamweaver или Photoshop. «Безопасные» веб-цвета легко узнать. Они составлены исключительно из шестнадцатеричных значений 00, 33, 66, 99, CC и FF.

Веб-палитра — это набор из 216 цветов, которые браузеры используют для отображения оттенков на мониторах с уровнем цветопередачи в 256 цветов. Все цвета веб-палитры состоят из профилей, используемых как операционной системой Windows, так и OS X.

В прошлом, когда у большинства пользователей были мониторы, отображающие не более 256 цветов, веб-дизайнеры придерживались «безопасных» веб-цветов, потому что они отображались именно так, как задумывалось. Поскольку на момент написания книги мониторы с уровнем цветопередачи в 256 цветов практически исчезли, браузеры при отображении редко переделывают цвета в веб-безопасные. Это значит, что больше нет необходимости ограничивать выбор цветов — прогресс вывел веб-палитру из употребления.

R:80, G:140 и B:25 (мы будем использовать шестнадцатеричный код **#508C19**). Вы можете видеть, что если применить свойство **color** к элементу **blockquote**, цвет наследуется элементами **p** и **em**, которые он содержит (рис. 13.8). Толстые пунктирные границы вокруг всего блока цитаты также зеленые; однако, если бы мы должны были применить свойство **border-color** к этому самому элементу, основной цвет заменил бы установленный зеленый.

Правило стилей

```
blockquote {
border: 4px dashed;
color: #508C19;
}
```

Разметка

```
<blockquote>
<p>Я бы порекомендовала хлопья "Любятово" каждому, кто любит хлопья. Это <em>единственный</em> способ начать день!.</p>
<cite>&mdash; Светлана Рыжикова, довольный покупатель</cite>
</blockquote>
```

Я бы порекомендовала хлопья "Любятово" каждому, кто любит хлопья. Это *единственный* способ начать день!

— *Светлана Рыжикова, довольный покупатель*

Рис. 13.8. *Применение основного цвета к элементу*

СОВЕТ ДИЗАЙНЕРА

Использование цвета

Ниже приведены несколько коротких советов, относящихся к работе с цветом:

- Ограничьте количество цветов, которые вы используете на странице. Ничто не создает визуальный хаос быстрее, чем слишком большое количество цветов. Я обычно выбираю один основной цвет и один — для выделения. Я также могу использовать пару оттенков каждого из них, но воздерживаюсь от добавления слишком большого количества различных тонов.
- При задавании основного и фонового цветов убедитесь, что существует достаточный контраст. Посетителям веб-сайтов больше нравится читать темный текст на очень светлом фоне. Мой пример на рис. 13.9, несмотря на формулирование этой точки зрения, как ни удивительно, терпит неудачу при проверке на контраст.
- Хорошая идея —задавать основной и фоновый цвета в тандеме, особенно если это делается для всех страниц. Такой способ поможет вам избежать возможных цветовых противоречий и проблем с контрастом, если пользователь выполнил ту или иную настройку с применением пользовательской таблицы стилей.

Фоновый цвет

При помощи свойства **background-color** вы можете применять фоновые цвета к любому элементу.

background-color

Принимаемые значения: *значение цвета (имя или число)* | **transparent** | **inherit**

Значение по умолчанию: **transparent**

Применение: ко всем элементам

Наследование: нет

Фоновый цвет заполняет ***холст***, то есть пространство позади элемента, которое включает в себя область контента и отступ — некоторое рассояние между контентом и внешними границами. Давайте рассмотрим, что происходит, когда мы используем свойство **background-color** для того, чтобы сделать фон блока цитаты **blockquote** светло-синим (рис. 13.9).

```
blockquote {
border: 4px dashed;
color: #508C19;
background-color: #B4DBE6;
}
```

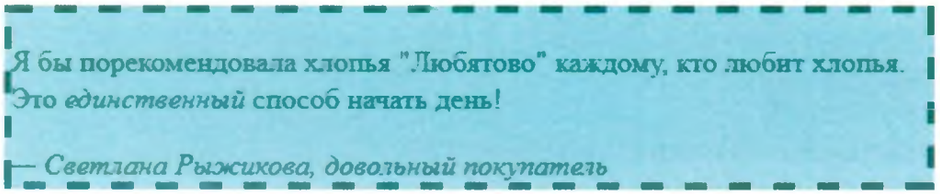


Рис. 13.9. Добавление светло-синего фонового цвета к блоку цитаты

Как и ожидалось, фоновый цвет заполнил область позади текста полностью, до границ. Присмотритесь к промежуткам в границах, и вы увидите, что в них виден фоновый цвет. Если применить отступы к данному элементу, фоновый цвет не будет отображаться в промежутках. Когда будем говорить о блочной модели CSS, мы пересмотрим все эти компоненты элемента. А сейчас просто знайте, что если границы прерывистые, фон будет виден в промежутках.

Стоит отметить, что фоновые цвета не наследуются, но из-за того, что настройкой фона по умолчанию для всех элементов является **transparent**, фоновый цвет родительского элемента отображается сквозь его дочерние элементы. Например, вы можете изменить фоновый цвет всей страницы применением свойства **background-color** к элементу **body**. Цвет будет виден сквозь все элементы на странице.

Чтобы задать фоновый цвет для всей страницы, примените свойство **background-color** *к элементу* **body**.

Вы можете не только задавать цвет всей страницы, но и изменять фон любого элемента, как блочного (как **blockquote**, показанный в предыдущем примере), так и встроенного. В этом примере я использовала свойства **color** и **background-color** для выделения слова, размеченного как элемент «glossary». Вы можете видеть на рис. 13.10, что фоновый цвет заполняет небольшой блок, созданный встроенным элементом **dfn**.

Правило стилей

```
.glossary {
color: #7C3306; /* темно-коричневый */
background-color: #F2F288; /* светло-желтый */
}
```

Разметка

```
<p><dfn class="glossary">Базовая линия</dfn> — это воображаемая линия, на которой располагаются символы.</p>
```

Базовая линия — это воображаемая линия, на которой располагаются символы.

Рис. 13.10. Применение свойства **background-color** *к встроенному элементу*

Непрозрачность

Ранее мы говорили о цветовом формате RGBa, который добавляет уровень прозрачности, когда применяется к цвету или фону. Существует еще один способ сделать элемент частично прозрачным, и это свойство **opacity** спецификации CSS3.

opacity

Новый в CSS3

Принимаемые значения: *число (от 0 до 1)*

Значение по умолчанию: 1

Применение: ко всем элементам

Наследование: нет

Значение свойства **opacity** представляет собой число от 0 (совершенно прозрачный) до 1 (совершенно непрозрачный). Значение .5 делает элемент прозрачным на 50%. Эта настройка применяется ко всему элементу, как к переднему плану, так и к фону (если он был задан). Когда вы хотите повлиять на передний план или фон по отдельности, вместо свойства **opacity** используйте значение цвета RGBa.

В следующем примере кода (и на рис. 13.11) заголовку был присвоен зеленый цвет, а фону — белый. Когда задано свойство **opacity**, оно позволяет фону страницы проглядывать сквозь текст и рамку элемента.

```
h1 {color: green; background: white; opacity: .25;}
h2 {color: green; background: white; opacity: .5;}
h3 {color: green; background: white; opacity: 1;}
```

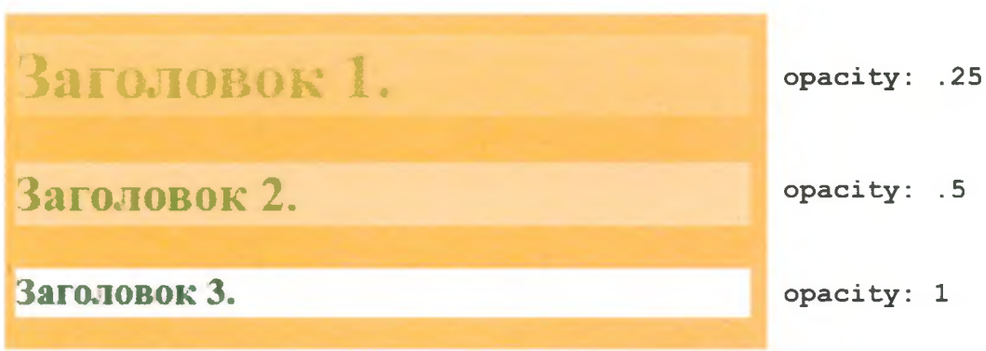


Рис. 13.11. Параметр прозрачности элемента влияет на цвета переднего плана и фона

Наверное вы спешите опробовать в действии свойства цветов и фона, и чуть позже мы этим займемся, но сначала я хочу познакомить вас с некоторыми более интересными селекторами CSS в довершение коллекции. Во врезке «На заметку» перечислены те, с которыми вы к этому моменту должны уже вполне уверенно работать.

НА ЗАМЕТКУ

Ниже приведена краткая сводка типов селекторов, которые мы уже рассмотрели:

- Селектор элемента

```
p {property: value;}
```

- Сгруппированные селекторы

```
p, h1, h2 {property: value;}
```

- Селектор потомков (контекстуальный)

```
ol li {property: value;}
```

- Селектор идентификатора

```
#sidebar {property: value;}
```

```
div#sidebar {property: value;}
```

- Селектор класса

```
p.warning {property: value;}
```

```
warning {property: value;}
```

- Универсальный селектор

```
* {property: value;}
```

Селекторы псевдокласса

Вы когда-нибудь замечали, что ссылка часто бывает одного цвета, когда вы щелкаете по ней, и другого цвета, когда вы возвращаетесь обратно на ту же страницу? Это происходит потому, что «за кадром» ваш браузер отслеживает, по каким ссылкам вы щелкали (то есть какие страницы посещали). Браузер также отслеживает и другие состояния ссылок, такие как: наведен ли указатель мыши пользователя на ссылку (наведение), является ли элемент первым из элементов данного типа, является ли он первым или последним потомком родительского элемента, был ли элемент формы выделен или отключен, и это только некоторые.

В CSS вы можете применять стили к ссылкам в каждом из этих состояний, используя специальный вид селектора, называемый *селектором псевдокласса*. Это необычное название, но можно представить, что ссылки в определенном состоянии принадлежат к тому же классу. Однако имя класса не находится в разметке — это то, за чем следит браузер. Так что это *почти* класс... *псевдокласс*.

Селекторы псевдокласса обозначаются символом двоеточия (`:`). Обычно они следуют сразу за именем элемента, например, **`li:first-child`**.

Полный список селекторов CSS3 с описаниями и примерами к каждому можно найти в приложении Б.

В CSS3 всего несколько псведоклассов. В этом разделе я познакомлю вас с наиболее распространенными и хорошо поддерживаемыми.

Якорные псевдоклассы

Самые основные селекторы псевдоклассов нацелены на ссылки (элементы **`a`**) на основании того, были ли они нажаты. Якорные псевдоклассы — это тип *динамических псевдоклассов*, так как они применяются в результате взаимодействия пользователя со страницей, а не благодаря элементам разметки.

`:link`	Применяет стиль к не нажатым (не посещенным) ссылкам
`:visited`	Применяет стиль к ссылкам, которые уже были нажаты

По умолчанию браузеры обычно отображают ссылки синим цветом, а посещенные ссылки — фиолетовым. Но вы можете изменить это, применив несколько правил стилей.

ПРЕДУПРЕЖДЕНИЕ

Когда вы меняете внешний вид непосещенных и посещенных ссылок, убедитесь, что они по-прежнему похожи на ссылки.

В этих примерах я изменила цвет ссылок (синий по умолчанию) на темно-бордовый, а посещенные ссылки теперь будут отображаться серым цветом. Обычно посещенные ссылки отображаются менее ярким цветом, чем непосещенные.

```
a:link {
color: maroon;
}
a:visited {
color: gray;
}
```

Псевдоклассы действий пользователя

Другой тип динамических псевдоклассов нацелен на состояния элементов, возникающие в результате прямых действий пользователя.

:focus Применяет стиль, когда элемент выделен и готов ко вводу

:hover Применяет стиль, когда на элемент наведен указатель мыши

:active Применяет стиль, пока элемент (ссылка или кнопка) нажат

Состояние фокуса

Если вы когда-нибудь пользовались веб-формой, вы, должно быть, знаете, как браузер визуально выделяет элемент формы, когда вы его выбираете. Когда элемент выделен и принимает вводимые пользователем данные, говорят, что он имеет «фокус». Селектор **:focus** позволяет применить пользовательские стили к элементам, которые попадают в фокус. В этом примере, когда пользователь вводит текст, появляется желтый цвет фона, чтобы выделить его из других элементов формы.

```
input:focus { background-color: yellow; }
```

Состояние «при наведении»

Интерес представляет селектор **:hover**. Он применяет стиль, когда указатель мыши находится непосредственно над элементами. И хотя состояние «при наведении» можно применять к любому элементу, чаще всего его используют со ссылками, чтобы изменить их внешний вид и предоставить пользователю визуальную обратную связь, указывая, что действие возможно. Правило задает ссылке светло-розовый цвет фона при наведении на нее курсора.

```
a:hover {
color: maroon;
background-color: #ffd9d9;
}
```

Важно отметить, что состояние «при наведении» отсутствует в устройствах с сенсорным экраном, таких как смартфоны или планшеты, на них этот элемент обратной связи будет потерян (см. примечание). Поэтому важно обеспечить ссылки четкими визуальными индикаторами, не полагаясь на взаимодействие с мышью.

Активное состояние

Наконец, селектор **:active** применяет стили к элементу, когда тот находится в состоянии активации. В случае со ссылкой этот стиль применяется, когда по ней щелкают мышью или касаются сенсорного экрана в ее позиции. Этот стиль будет отображаться только мгновение, но он может дать тонкий намек на то, что что-то произошло. В данном при-

мере я сделала цвет активного состояния более ярким (поменяла с бордового на красный).

```
a:active {
color: red;
background-color: #ffd9d9;}
```

Резюме

Веб-дизайнеры обычно предоставляют стили для всех состояний ссылки, потому что это удобный способ обеспечить обратную связь на каждом этапе (и обычно это улучшает заданные по умолчанию настройки браузера). На самом деле, пользователи ожидают обратную связь: чтобы с первого взгляда можно было определить, по каким ссылкам они переходили, чтобы ссылки что-нибудь делали при наведении на них, и чтобы они предоставляли подтверждение успешного нажатия.

При применении стилей к элементам **a** всех пяти псевдоклассов, для их правильной работы очень важен порядок появления. Например, если разместить псевдоклассы **:link** или **:visited** в конце, они отменят все остальные состояния и не дадут их отобразить. Необходимый порядок следования псевдоклассов таков: **:link**, **:visited**, **:focus**, **:hover**, **:active**. Рекомендуется предоставлять стиль **:focus** пользователям, которые используют клавиатуру для перехода по ссылкам, а не щелкают по ним мышью. Тот же стиль часто применяется для состояния **:hover**, хотя это необязательно. Подводя итоги, отмечу, что все упомянутые мной стили ссылок должны выглядеть в таблице стилей так, как показано ниже. На рис. 13.12 представлен результат.

```
a { text-decoration: none; } /* отключает подчеркивание всех
ссылок */
a:link { color: maroon; }
a:visited { color: gray; }
a:focus { color: maroon; background-color: #ffd9d9; }
a:hover { color: maroon; background-color: #ffd9d9; }
a:active { color: red; background-color: #ffd9d9; }
```

ПРИМЕЧАНИЕ

Несмотря на то, что навести палец на элемент нельзя, браузер Safari в устройствах под управлением операционной системы iOS и некоторые устройства на платформе Android могут отображать стили состояния `:hover` после однократного касания. Для перехода по ссылке пользователь должен коснуться ее еще раз.

Такой подход гарантирует, что раскрывающиеся списки, управляемые CSS, которые открываются при наведении на них, будут доступны и на устройствах с сенсорным экраном. С другой стороны, с иными объектами, которые могут менять состояние при наведении, это может ввести посетителя в заблуждение или оказаться нежелательным. По этой причине некоторые разработчики решили создать отдельную таблицу стилей без стилей псевдокласса `:hover` для мобильных устройств, у которых могут быть сенсорные интерфейсы.

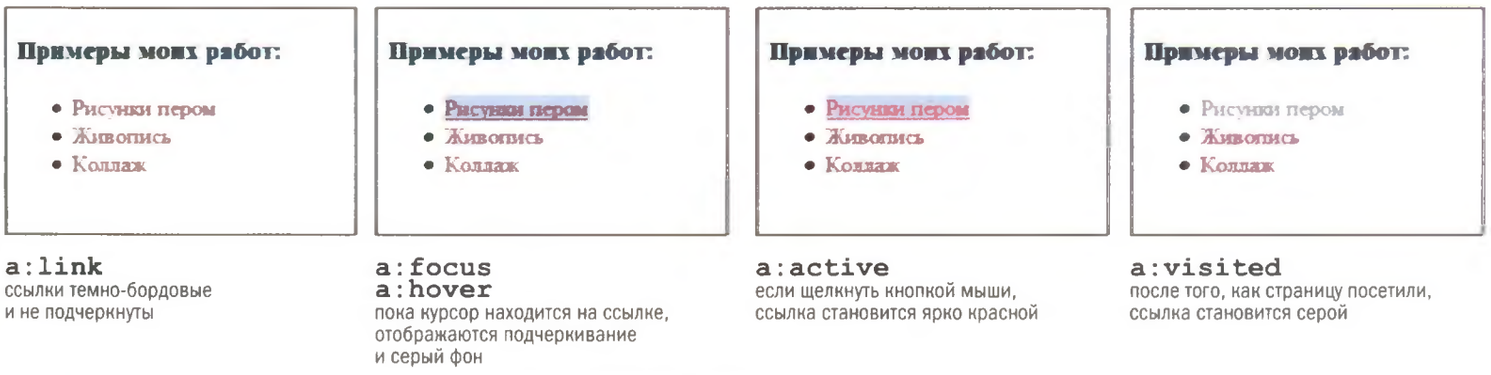


Рис. 13.12. Изменение цветов и фона ссылок при помощи селекторов

Другие селекторы псевдоклассов

Мы освоили пять псевдоклассов CSS3, осталось еще 18! Не знаю, как вам, а мне это кажется скучным. Я хочу, чтобы вы узнали обо всех возможностях, поэтому и вынесла их во врезку «Еще псевдоклассы», теперь мы можем перейти к другим типам селекторов. Кроме того, вы найдете их все в приложении Б с описаниями и примерами. Когда будете готовы заняться более сложной работой с селекторами, используйте это приложение в качестве справочного материала.

Еще псевдоклассы

Кроме динамических псевдоклассов, модуль селекторов CSS3 включает в себя другие типы селекторов, основанные на состояниях, которые браузер отслеживает на ходу.

Следует отметить, что ни один из этих типов селекторов не поддерживается в браузере Internet Explorer 8 или его более ранних версиях. Для создания поддержки с помощью JavaScript, попробуйте применить инструмент Selectivizr (**selectivizr.com**), который представляет собой сценарий, добавляемый в файл и эмулирующий поддержку в старых версиях браузера Internet Explorer. Selectivizr является примером полизаполнения — сценария, который добавляет поддержку современных веб-функций в старые браузеры (полизаполнения обсуждаются в главе 22.

Структурные псевдоклассы

Позволяют сделать выбор на основании того, где находится элемент в структуре документа (дерево документа). Вы увидите, что их имена адекватно описывают, на что они нацелены.

```
:root          :only-child      :nth-child()
:empty         :first-of-type   :nth-last-child()
:first-child   :last-of-type    :nth-of-type()
:last-child    :only-of-type    :nth-last-of-type()
```

Селекторы пользовательского интерфейса

Эти селекторы применяются к состояниям, типичным для виджетов формы.

```
:enabled       :disabled        :checked
```

Прочие

Дополнительные псевдоклассы включают в себя:

`:target` (выбирает элементы, на которые нацелен идентификатор фрагментов в URL-адресе)

`:lang()` (выбирает на основе двухсимвольного кода языка)

`:not()` (выбирает каждый элемент, кроме того, что указан в скобках)

Селекторы псевдоэлементов

Псевдоклассы не являются единственным видом псевдоселекторов. Существует еще четыре псевдоэлемента, которые действуют, словно вставляя в структуру документа фиктивные элементы стилизации. В спецификации CSS3 псевдоэлементы обозначаются двойным символом двоеточия (`::`), но для лучшей совместимости с более старыми версиями используйте один символ двоеточия (`:`), как обозначалось в CSS2.

Первые буква и строка

Следующие два псевдоэлемента используются для выбора первой буквы или первой строки текста элемента.

`:first-line`

Этот селектор применяет правило стилей к первой строке определенного элемента. Тем не менее единственные свойства, которые вы можете применить, это:

`color`	`font`	`background`
`word-spacing`	`letter-spacing`	`text-decoration`
`vertical-align`	`text-transform`	`line-height`

`:first-letter`

Применяет правило стилей к первой букве определенного элемента. Свойства, которые вы можете применять, ограничиваются:

`color`	`font`	`text-decoration`
`text-transform`	`vertical-align`	`padding`
`background`	`margin`	`line-height`
`border`	`float`	
`letter-spacing`	`word-spacing`	

На рис. 13.13 показаны примеры селекторов псевдоэлементов **`:first-line`** и **`:first-letter`**.

```
p:first-line {letter-spacing: 8px;}
```

Белоснежку изгнали за то, что она была самой красивой, ей помогали семь гномов, однажды она съела отравленное яблоко и уснула, и спала в хрустальном гробу до тех пор, пока прекрасный принц не поцеловал ее, женился на ней, и они жили долго и счастливо до конца своих дней.

```
p:first-letter {font-size: 300%; color: orange;}
```

Белоснежку изгнали за то, что она была самой красивой, ей помогали семь гномов, однажды она съела отравленное яблоко и уснула, и спала в хрустальном гробу до тех пор, пока прекрасный принц не поцеловал ее, женился на ней, и они жили долго и счастливо до конца своих дней.

*Рис. 13.13. Примеры селекторов псевдоэлементов **:first-line** и **:first-letter***

ПРИМЕЧАНИЕ

В этом списке есть несколько свойств, которые вы еще не видели. Мы охватим связанные с блоками свойства (поля, отступы, границы) в главе 14. Свойство **`float`** будет рассмотрено в главе 15.

Контент, генерируемый при помощи псевдоэлементов :before и :after

В спецификации CSS2 были введены псевдоэлементы **:before** и **:after**, которые используются для вставки контента до или после определенного элемента без действительного добавления символов в исходном коде документа (именуется как *генерируемый контент* в CSS).

Генерируемый контент может быть использован для вставки соответствующих языку кавычек, в которые заключена цитата, автоматических счетчиков или даже для отображения URL-адресов рядом с указателями ссылок, когда документ выводится на печать.

Ниже приведен простой пример, который вставляет слова «Однажды:» перед абзацем и «Конец.» в конце абзаца (рис. 13.14).

Таблица стилей

```
p:before {
content: "Однажды: ";
font-weight: bold;
color: purple;
}
p:after {
content: "Конец.";
font-weight: bold;
color: purple;
}
```

Разметка

```
<p> Белоснежку изгнали за то, что она была самой красивой,
... и они жили долго и счастливо до конца своих дней.</p>
```

Однажды: Белоснежку изгнали за то, что она была самой красивой, ей помогали семь гномов, однажды она съела отравленное яблоко и уснула, и спала в хрустальном гробу до тех пор, пока прекрасный принц не поцеловал ее, женился на ней, и они жили долго и счастливо до конца своих дней. **Конец.**

Рис. 13.14. *Генерируемый контент добавлен при помощи псевдоселекторов* ***:before*** *и* ***:after****, показанный в браузере Firefox*

Генерируемый контент хорошо поддерживается всеми современными браузерами. Вероятно, вы не станете использовать генерируемый контент в своих первых проектах веб-сайтов, но если хотите узнать о нем больше, посетите страницы **habrahabr.ru/post/154319/** и **htmlbook.ru/css/content**. А если вам требуется описание всех методов, посетите сайт **www.w3.org/TR/css3-content/**.

Селекторы атрибутов

Наконец мы выходим на финишную прямую в изучении селекторов. Последний тип селекторов целенаправленно выбирает элементы на основе их атрибутов. Вы можете взять имена атрибутов или их значения, что обеспечивает довольно большую степень гибкости при выборе элементов без необходимости многократно добавлять разметку с элементами **class** или **id**.

Следующие селекторы атрибутов были представлены в CSS2 и хорошо поддерживаются браузерами.

элемент[атрибут]

Селектор простых атрибутов — целенаправленно выбирает элементы с конкретным атрибутом независимо от его значения. В примере, приведенном ниже, выбирается любое изображение с атрибутом **title**.

```
img[title] {border: 3px solid;}
```

элемент[атрибут="точное значение"]

Селектор точного значения атрибута — выбирает элементы с конкретным значением атрибута. Селектор выбирает изображения с точным значением заголовка «first grade».

```
img[title="first grade"] {border: 3px solid;}
```

элемент[атрибут~="значение"]

Селектор неполного значения атрибута — позволяет задать часть значения атрибута, отделенную пробелами. В следующем примере мы ищем слово «grade» в заголовке, так что будут выбраны изображения со значениями заголовков «first grade», «second grade» и пр.

```
img[title~="grade"] border: 3px solid;}
```

элемент[атрибут|="значение"]

Селектор разделенного дефисом значения атрибута — позволяет задать часть значения атрибута, отделенную дефисами. Этот селектор выбирает любую ссылку, указывающую на документ, написанный на разновидности английского языка (**en**), неважно, будет ли в значении атрибута обозначение **en-us** (американский английский), **en-in** (индийский английский), **en-au-tas** (австралийский английский) и т. д. (Помните, что символ * обозначает универсальный селектор, который выбирает любой элемент).

```
*[hreflang|="en"] {border: 3px solid;}
```

Следующие расширенные селекторы атрибутов появились в спецификации CSS3, поэтому они только обретают популярность.

элемент[атрибут^="первая часть значения"]

Селектор атрибута значения начала подстроки— выбирает элементы, чьи указанные значения атрибутов *начинаются* со строки символов се-

лектора. Он применяет стиль ко всем изображениям, которые находятся в каталоге /images/icons (<img src="/images/icons…>).

```
img[src^="/images/icons"] {border: 3px solid;}
```

```
элемент[атрибут$="последняя часть значения"]
```

Селектор атрибута значения конца подстроки — выбирает элементы, чьи указанные значения атрибутов *заканчиваются* строкой символов селектора. В этом примере вы можете применить стиль только к элементам **a**, которые ссылаются на файлы PDF.

```
a[href$=".pdf"] {border: 3px solid;}
```

```
элемент[атрибут*="любая часть значения"]
```

Селектор атрибута произвольного значения подстроки ищет указанную текстовую строку в любой части заданного значения атрибута. Это правило выбирает любое изображение, которое содержит слово «year» где-либо в названии.

```
img[title*="year"] {border: 3px solid;}
```

Итак, мы закончили с селекторами! Я думаю, что теперь самое время поработать с цветами переднего плана и фона, а также с некоторыми из новых типов селекторов в упражнении 13.1.

УПРАЖНЕНИЕ 13.1. ДОБАВЛЕНИЕ ЦВЕТА К ДОКУМЕНТУ

В этом упражнении мы изменим вид простой черно-белой статьи и придадим ей индивидуальность с помощью основных и фоновых цветов (рис. 13.15). У вас должно быть уже достаточно опыта в написании правил стилей, так что я не собираюсь детально все объяснять, как делала в предыдущих упражнениях. На этот раз вы *сами* напишете правила. Можете проверить свою работу на соответствие с завершенной таблицей стилей, предоставленной в приложении А.

Откройте файл *bistro.html* (доступный на диске, прилагающемся к книге) в текстовом редакторе. Вы обнаружите, что в нем уже есть глобальная таблица стилей, обеспечивающая базовое форматирование текста (в нем даже есть демонстрация свойств **margin** и **padding**, которые будут вводиться в следующей главе). Когда текст полностью готов, все что вам нужно сделать — это поработать над цветом. Сохраняйте документ на каждом шаге выполнения упражнения и оценивайте свои достижения в браузере.

1. Сделайте заголовок **h1** фиолетовым (R:153, G:51, B:153 или **#993399**), добавив определение в существующее правило стилей **h1**. Обратите внимание, что из-за того, что это значение имеет дублированные цифры, вы можете (и должны) использовать сжатую версию (**#939**) и сэкономить несколько байтов в таблице стилей.
2. Сделайте заголовки **h2** светло-кирпичного оттенка (R:204, G:102, B:0 или **#cc6600**).
3. Сделайте фон всей страницы светло-зеленым (R:210, G:220, B:157 или **#d2dc9d**). Теперь, наверное, самое время сохранить файл, просмотреть его в браузере и исправить ошибки, если фон и заголовки не отображаются в цвете.

ПРЕДУПРЕЖДЕНИЕ

Не забудьте указывать символ # перед шестнадцатеричными значениями цвета. Правило не будет работать без него.

4. Сделайте элемент **div** «header» прозрачным на 50% (R:255, G:255, B:255, .5), чтобы сквозь него слегка просматривался цвет фона.
5. Я уже добавила правило стилей, отключающее подчеркивание ссылок (**textdecoration: none**), поэтому нам придется полагаться на цвет, чтобы сделать их заметными. Напишите правило, которое придаст ссылкам такой же цвет, что и у заголовка **h1** (**#993399**).
6. Сделайте посещенные ссылки тускло-фиолетовыми (**#937393**).
7. Для моментов, когда указатель мыши располагается на ссылках, сделайте фиолетовый цвет текста более ярким (**#c700f2**) и добавьте белый цвет фона (**#fff**). Ссылки будут словно светиться, когда на них наведен указатель мыши. Используйте те же самые правила стилей для случаев, когда ссылки в фокусе.
8. При щелчке по ссылке мышью (или касании на устройстве с сенсорным экраном) добавьте белый цвет фона и сделайте текст ярко фиолетовым (**#ff00ff**).

Убедитесь, что все якорные псевдоклассы расположены в правильном порядке. Когда вы все сделаете, страница должна выглядеть так, как показано на рис. 13.15. Позже мы добавим к ней фоновые изображения, так что если хотите продолжить эксперименты с разными цветами элементов, сделайте копию этого документа и переименуйте ее.

Рис. 13.15. Палитра цветов для страницы меню (показаны варианты «до» и «после»)

НА ЗАМЕТКУ

Свойства, относящиеся к фоновым изображениям:

background-image

background-repeat

background-position

background-attachment

background-clip (CSS3)

background-size (CSS3)

background

Фоновые изображения

Мы рассматривали, как добавить изображения к контенту документа с помощью элемента **img**, но сегодня большинство декоративных изображений добавляются на страницы и к элементам в качестве фоновых с помощью CSS. В конце концов, украшения, такие как мозаичный фоновый узор, определенно являются частью представления, а не структуры. Это также позволяет дизайнерам менять внешний вид сайта путем редактирования файла *.css* Мы оставили далеко позади то время, когда сайты содержали графику огромного размера и состояли из таблиц.

В этом разделе мы рассмотрим набор свойств, используемых для размещения и управления фоновыми изображениями, начиная с основного свойства **background-image**.

Добавление фонового изображения

ПРИМЕЧАНИЕ

Правильным термином для «URL-контейнера» является *функциональная запись*. Это синтаксическая структура, схожая с той, которая используется для перечисления десятичных или процентных значений RGB.

Свойство **background-image** используется для добавления фонового изображения к элементу. Его главной задачей является указание расположения файла с изображением.

background-image

Принимаемые значения: *URL-адрес (расположение изображения)* | **none** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: нет

Значение свойства **background-image** — это что-то вроде URL-контейнера, который содержит URL-адрес изображения (см. примечание). URL относится к местоположению правила CSS в данный момент. Если правило находится в глобальной таблице стилей (элемент **style** HTML-документа), то путь, указанный в URL-адресе будет иметь отношение к файлу HTML. Если правило CSS находится во внешней таблице стилей, тогда путь к изображению должен прописываться относительно местоположения файла *.css*. Другой подход описан в совете дизайнера.

Эти примеры и рис. 13.16 демонстрируют фоновые изображения, наложенные позади всей страницы (**body**) и одного элемента **blockquote** с примененным отступом и границами.

```
body {
background-image: url(star.gif); }
blockquote {
background-image: url(dot.gif);
padding: 2em;
border: 4px dashed;}
```

МАТЕРИАЛЫ

На сайте Standardista содержатся невероятно подробные таблицы поддержки браузеров для всех возможных свойств, связанных с фоном. На них определенно стоит взглянуть. **www.standardista.com/css3/css3-background-properties/**.

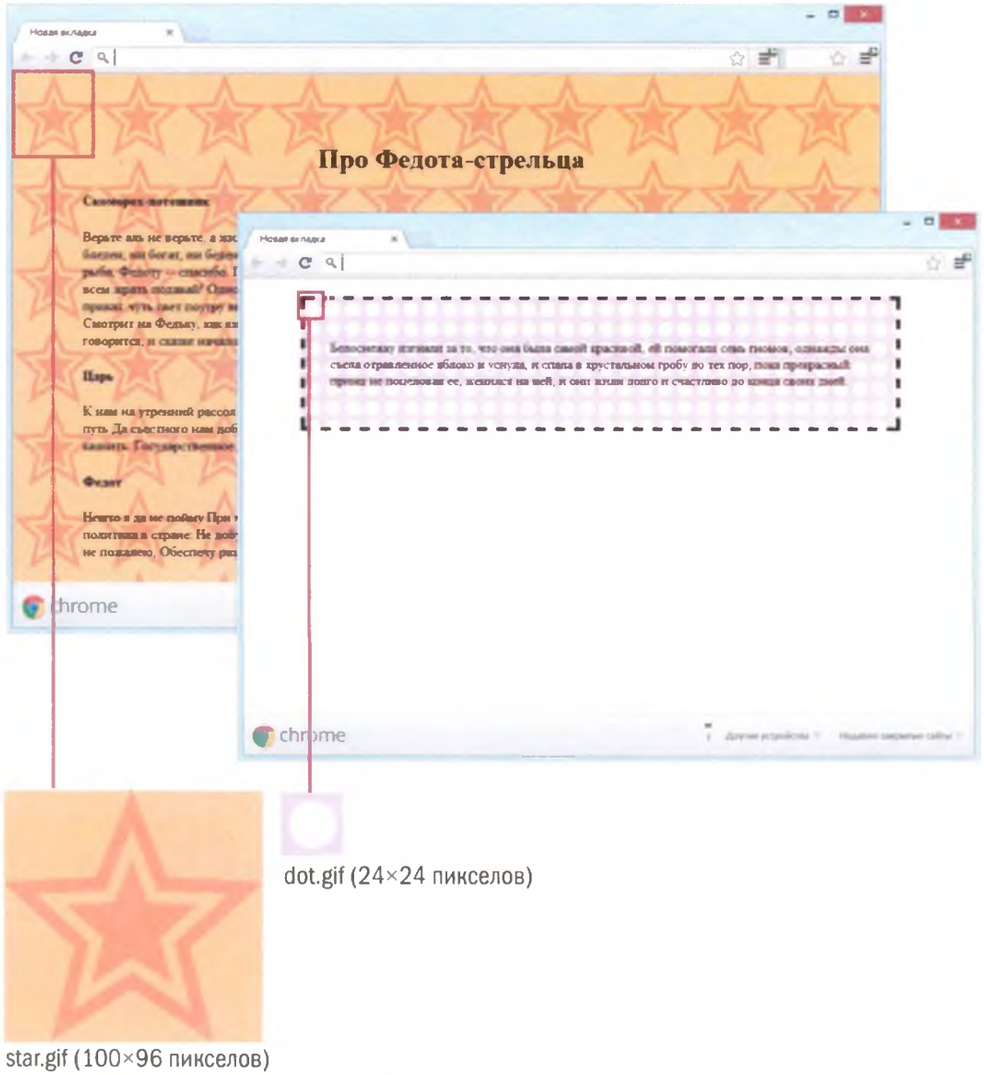


Рис. 13.16. *Примеры мозаичного размещения фоновых изображений, добавленных при помощи свойства* **`background-image`***

Здесь вы можете видеть поведение свойства **`background-image`** по умолчанию. Изображение помещается в верхний левый угол и располагается мозаичным способом в горизонтальном и вертикальном направлении до тех пор, пока весь элемент не заполнится (вы узнаете, как изменить это в один момент).

Подобно фоновым цветам, мозаичное размещение фоновых изображений заполняет область позади контента, дополнительную область отступов вокруг контента, и продолжается до внешних сторон границ (когда границы есть).

Если вы снабжаете элемент свойством **`background-color`** или **`background-image`**, изображение будет помещено поверх цвета. Тем не менее рекомендуется задавать цвет, схожий по тону, на случай, если произойдет сбой при отображении изображения.

* Данный отрывок взят из сказки Леонида Филатова «Про Федота-стрельца, удалого молодца».

СОВЕТ ДИЗАЙНЕРА

Фоновые изображения

При работе с фоновыми изображениями имейте в виду эти рекомендации и советы:

- Используйте простые изображения, которые не будет затруднять чтение текста, расположенного поверх.
- Всегда задавайте значение свойства **`background-color`**, которое соответствует основному цвету фонового изображения. Если произойдет сбой при отображении фонового изображения, по крайней мере, общий дизайн страницы будет однотонным. Это особенно важно, если цвет текста будет трудно читаемым на белом фоне.
- Для Всемирной паутины придерживайтесь как можно меньшего размера файла фоновых изображений.

УПРАЖНЕНИЕ 13.2. ДОБАВЛЕНИЕ МОЗАИЧНОГО ФОНОВОГО ИЗОБРАЖЕНИЯ

В этом упражнении мы добавим простое мозаичное фоновое изображение на страницу меню. Рисунки, предоставленные для этого упражнения, должны быть расположены в каталоге *images*.

Добавьте определение к правилу стилей **body**, которое размещает изображение *bullseye.png* мозаикой на заднем плане. Убедитесь, что путь указан относительно таблицы стилей (в данном случае относительно текущего HTML-документа).

```
background-image: url(images/bullseye.png)
```

Когда вы сохраните страницу и откроете в браузере, она должна выглядеть так же, как показано на рис. 13.17. Обратите внимание: изображение *bullseye.png* — частично прозрачный мозаичный рисунок, так что он будет сливаться с цветом фона. Вы узнаете, как создавать прозрачные PNG-изображения в главе 19. Попробуйте временно изменить свойство **background-color** элемента **body**, добавив второе определение **background-color** ниже в наборе определений так, чтобы оно отменяло предыдущее. Поэкспериментируйте с различными цветами и обратите внимание, как кольца сливаются с ними. Закончив экспериментировать, удалите второе определение, чтобы фон снова стал зеленым, и переходите к выполнению следующих упражнений.

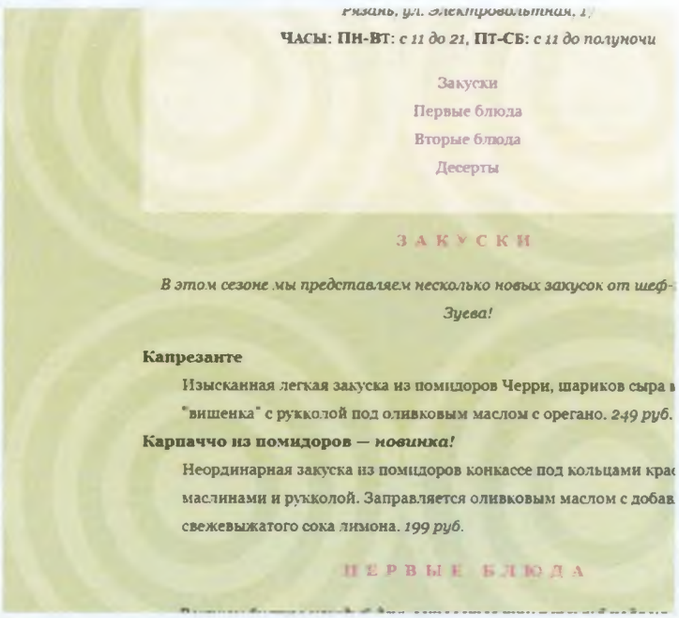


Рис. 13.17. Страница с простым мозаичным размещением фонового изображения

Управление мозаичным покрытием фона

Как мы видели на последнем рисунке, изображения размещались мозаикой вверх и вниз, вправо и влево, когда действовали установки по умолчанию. Вы можете изменить такое поведение при помощи свойства **background-repeat**.

background-repeat

Принимаемые значения: **repeat** | **repeat-x** | **repeat-y** | **no-repeat** | **inherit**

Значение по умолчанию: **repeat**

Применение: ко всем элементам

Наследование: нет

Если вы хотите, чтобы фоновое изображение появилось только один раз, используйте зарезервированное значение **no-repeat**, как показано ниже.

```
body {
background-image: url(star.gif);
background-repeat: no-repeat;
}
```

Вы также можете ограничить размещение изображения только горизонтальным (**repeat-x**) или вертикальным размещением (**repeat-y**), как показано в примерах ниже.

```
body {
background-image: url(star.gif);
background-repeat: repeat-x;
}
body {
background-image: url(star.gif);
background-repeat: repeat-y;
}
```

На рис. 13.18 приведены примеры применения каждого зарезервированного значения. Обратите внимание, что в каждом из них размещение начинается с верхнего левого угла элемента (или окна браузера, когда изображение применяется к элементу **body**). В следующем разделе я покажу вам, как это изменить.

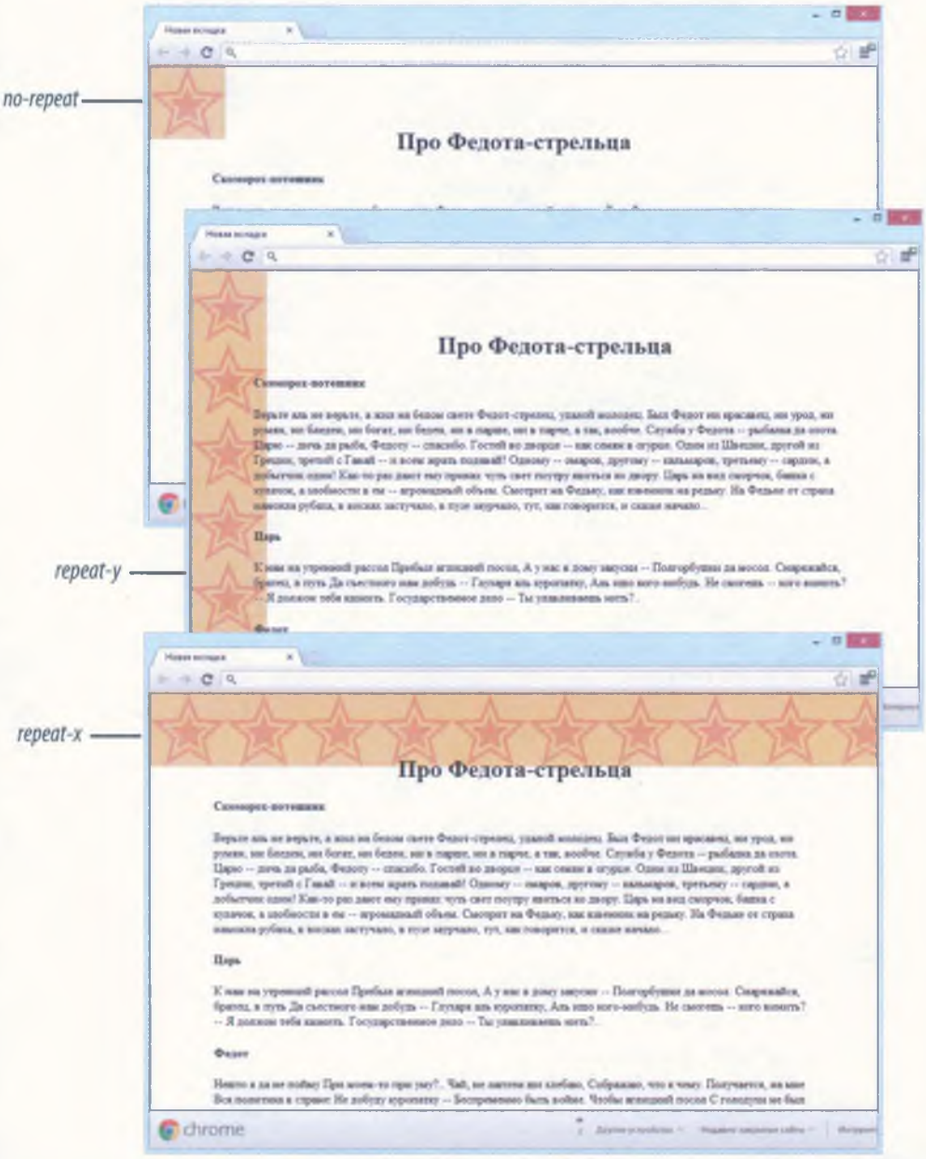


Рис. 13.18. *Отключение автоматического мозаичного размещения при помощи свойства* **`no-repeat`** *(верхнее изображение); мозаичные размещения по вертикальной оси при помощи свойства* **`repeat-y`** *(среднее изображение) и по горизонтальной оси при помощи свойства* **`repeat-x`** *(нижнее изображение)**

* Данный отрывок взят из сказки Леонида Филатова «Про Федота-стрельца, удалого молодца».

УПРАЖНЕНИЕ 13.3. УПРАВЛЕНИЕ МОЗАИЧНЫМ ПОКРЫТИЕМ ФОНА

Теперь попробуем реализовать немного более сложное размещение изображений на странице. На этот раз мы уберем назойливую мозаику на заднем плане и добавим более утонченный шаблон прямо внутри элемента **div** «header».

1. В правиле **#header**, добавьте изображение *purpledot.png* и задайте ему только горизонтальное повторение.

```
#header {
margin-top: 0;
padding: 3em 1em 2em 1em;
text-align: center;
background-color: rgba(255,255,255,.5);
background-image: url(images/purpledot.
png);
background-repeat: repeat-x;
}
```

2. Сохраните файл и взгляните на него в браузере. Он должен выглядеть так, как показано на рис. 13.19. Я рекомендую вам изменить размер окна браузера на более узкий или широкий и обратить внимание на положение фонового рисунка. Посмотрите, всегда ли он закреплен с левого края? Ниже мы узнаем, как настроить положение рисунка.
3. Попробуйте изменить правило, чтобы изображение повторялось по вертикали, затем сделайте так, чтобы оно вообще не повторялось (когда закончите, верните обратно значение **repeat-x**).

Рис. 13.19. Добавление горизонтально размещающегося изображения к элементу **#header div**

Положение фона

Свойство **background-position** задает положение *начального изображения* на заднем плане. Вы можете запомнить его как первое изображение, которое помещается на задний план, и от которого распространяется мозаичное размещение изображений. Ниже приведено свойство и его различные значения.

background-position

Принимаемые значения: *значение длины* | *проценты* | **left** | **center** | **right** | **top** | **bottom** | **inherit**

Значения по умолчанию: **0% 0%** (то же самое, что и **left top**)

Применение: ко всем элементам

Наследование: нет

Для размещения исходного изображения вы указываете горизонтальное и вертикальное значения, которые описывают, куда его поместить. Существуют различные способы реализации этого приема.

Зарезервированные значения положения

Зарезервированные значения (**left**, **center**, **right**, **top**, **bottom** и **center**) располагают начальное изображение относительно границ элемента. Например, значение **left** придвигает изображение вплотную к левой границе области фона. Исходное положение, задаваемое по умолчанию, соответствует значениям **"left, top"**.

Зарезервированные слова обычно используются в парах, как показано в примерах ниже:

```
{ background-position: left bottom; }
{ background-position: right center; }
```

Если вы задаете только одно зарезервированное слово, пропущенным считается центр. Таким образом, **background-position: right** дает тот же результат, что и **background-position: right center**.

Значения расстояния

Вы можете также задать положение изображения, указав в пикселах расстояние до него от верхнего левого угла элемента. При этом горизонтальное измерение всегда указывается первым.

```
{ background-position: 200px 50px; }
```

Проценты

Процентные значения задаются парами горизонтальные/вертикальные, с парой 0% 0%, соответствующей верхнему левому углу, и парой 100% 100%, соответствующей правому нижнему углу. Важно отметить, что процентное значение применяется как к области холста, так и к самому изображению. Например, значение 100% размещает правый нижний угол изображения в правом нижнем углу холста. Так же, как и с зарезервированными словами, если вы зададите только одно процентное значение, за другое значение будет принято 50% (центрированное).

```
{ background-position: 15% 100%; }
```

На рис. 13.20 показаны результаты каждого из примеров **background-position**, перечисленных выше, со свойством **background-repeat**, установленным для ясности в значение **no-repeat**. Можно поместить начальное изображение и задать ему распространение из этого положения в обоих направлениях или только горизонтально или вертикально. Когда изображение размещается мозаично, положение начального изображения будет неочевидным, но вы можете использовать свойство **background-position**, чтобы шаблон текстуры начинался в точке, отличной от левой границы изображения. Таким образом фоновый рисунок останется центрированным и симметричным.

СОВЕТ ПО CSS

Для гарантии лучшего представления в современных браузерах всегда задавайте горизонтальное измерение первым для всех типов значений.

ПРИМЕЧАНИЕ

Обратите внимание, что на рис. 13.20 фоновое изображение, размещаемое в углу элемента, находится внутри границы. Только повторяющиеся изображения простираются под границу и до ее внешнего края.

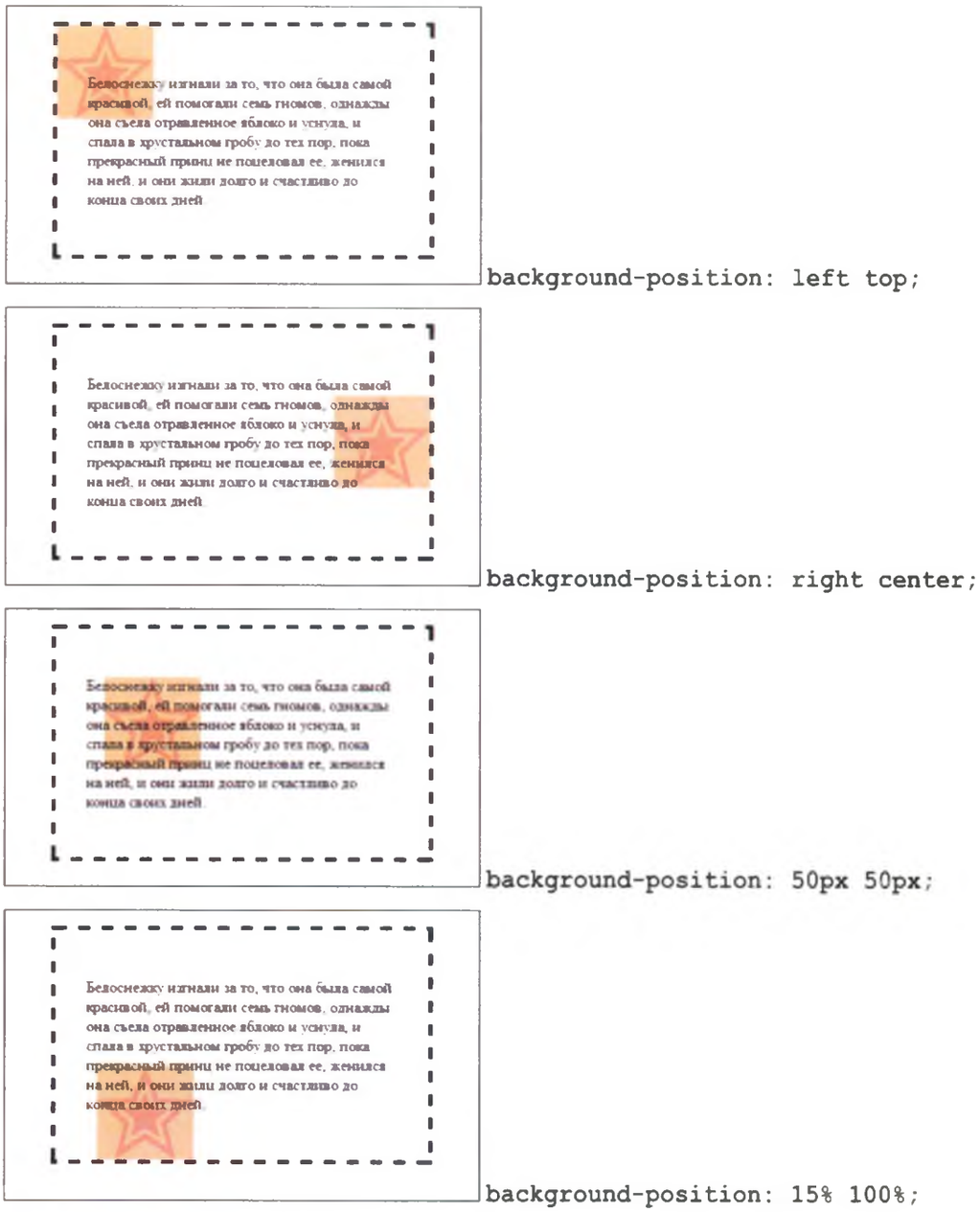


Рис. 13.20. Положение неповторяющегося фонового изображения

УПРАЖНЕНИЕ 13.4. РАЗМЕЩЕНИЕ ФОНОВЫХ ИЗОБРАЖЕНИЙ

Давайте повеселимся, размещая фоновое изображение на странице меню. Сначала мы произведем несколько тонких настроек уже имеющихся фоновых изображений, а затем совсем заменим их другим фоном и еще немного развлечемся. Мы по-прежнему работаем с файлом *bistro.html*, у которого должны присутствовать повторяющиеся мозаичные узоры в элементах **body** и **#header**.

1. Я полагаю, что, поскольку основные элементы меню выровнены по центру, будет неплохо, если фоновый узор также останется центрированным. Добавьте следующее определение к правилам элементов **body** и **#header**, затем сохраните файл и просмотрите его в браузере. Возможно, вы не заметите отличий, пока не измените ширину окна браузера на большую или меньшую. Теперь узор закреплен по центру и появляется с обеих сторон, а не только справа, как было раньше.

```
background-position: center top;
```

2. Ради забавы измените значение свойства **background-position**, чтобы фиолетовые точки располагались по нижнему краю элемента **div** заголовка **div** (**center bottom**). (Выглядит не очень красиво, я верну все обратно). Теперь попробуйте переместить рисунок *bullseye.png* вниз на 200 пикселов (**center 200px**). Обратите внимание, что узор все еще заполняет весь экран — мы переместили вниз исходное изображение, но распространение мозаичного фона по-прежнему задано во всех направлениях. На рис. 13.21 показан результат этих изменений.
3. Смотрится хорошо, но давайте пока уберем фон элемента **body**. Хочу показать вам маленькую хитрость. Занимаясь дизайном, я предпочитаю прятать стили в комментариях вместо того, чтобы удалять их безвозвратно. В этом случае мне не придется вспоминать их или вводить снова. Достаточно просто убрать символы комментариев, и они вернутся. Когда дизайн готов и настает время для публикации, я удаляю неиспользуемые стили, чтобы уменьшить размер файла. Ниже показано, как спрятать определения в комментариях:

```
body {
font-family: Georgia, serif;
font-size: 100%;
line-height: 175%;
margin: 0 15%;
background-color: #d2dc9d;
/* background-image: url(images/bullseye.png);
background-position: center 200px; */
}
```

4. Теперь добавьте изображение *blackgoose.png* (также частично прозрачный файл PNG) в фон страницы. Сделайте его неповторяющимся и выровненным по центру.

```
background-image: url(images/blackgoose.png);
background-repeat: no-repeat;
background-position: center top;
```

Просмотрите результат в окне браузера и понаблюдайте, как фон смещается вместе с контентом, когда вы прокручиваете страницу.

5. Я хочу, чтобы вы потренировались в использовании различных зарезервированных слов положения и числовых значений. Попробуйте применить каждое из них и посмотрите результат в браузере. Обязательно прокрутите страницу и посмотрите, что получится. Обратите внимание, что при указании процентного значения или зарезервированного слова для положения по вертикали, значение основывается на размере всего документа, а не только окна браузера. Вы можете также поэкспериментировать с собственными вариантами.

```
background-position: right top;
background-position: right bottom;
background-position: left 50%;
background-position: center 100px;
```

6. Оставьте изображение в положении **center 100px**, чтобы быть готовыми перейти к следующему упражнению. Ваша страница должна выглядеть так же, как на рис. 13.21.

Центрированный фоновый узор

Размещенное неповторяющееся изображение

Рис. 13.21. *Результат размещения исходного ихображения в виде мозаичного фонового узора (слева) и единственного фонового логотипа (справа)*

Фиксация фона

В предыдущем упражнении я попросила вас прокрутить страницу и посмотреть, что происходит с фоновым изображением. Как и ожидалось, оно прокручивается вместе с документом, что является его поведением по умолчанию. Однако вы можете использовать свойство **background-attachment**, чтобы отсоединить фон от контента и позволить ему остаться зафиксированным в одном положении, в то время как остальной контент прокручивается.

background-attachment

Принимаемые значения: **scroll** | **fixed** | **inherit**

Значения по умолчанию: **scroll**

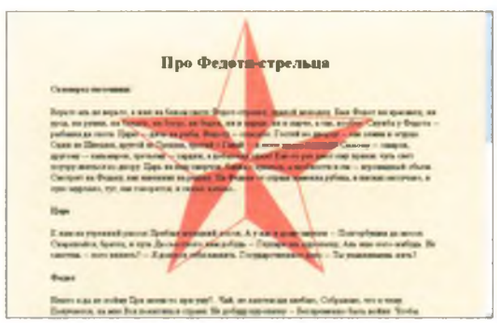
Применение: ко всем элементам

Наследование: нет

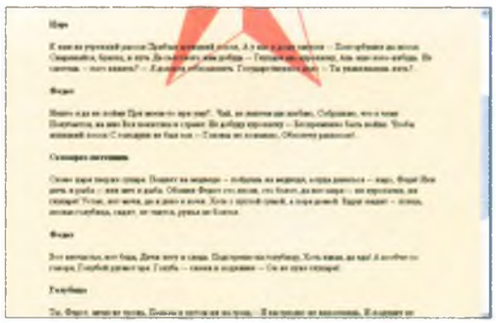
При использовании свойства **background-attachment** у вас есть выбор, будет ли фоновое изображение прокручиваться или останется зафиксированным. В последнем случае оно остается в одном и том же положении относительно видимой области окна браузера (в отличие от положения элемента, который заполняет). Вы увидите, что я имею в виду, через минуту. В следующем примере большое, не размещающееся мозаикой изображение помещено на задний план всего документа (элемент **body**). По умолчанию, когда документ прокручивается, изображение также прокручивается, перемещаясь вверх и за пределы страницы, как показано на рис. 13.22.

ПРИМЕЧАНИЕ

Вы можете зафиксировать положение фонового изображения для любого элемента, не только **body**.



Большое, неповторяющееся фоновое изображение в теле документа.



background-attachment: scroll;
По умолчанию фоновое изображение прикреплено к телу документа и прокручивается синхронно с контентом страницы.

background-attachment: fixed;
Когда фон зафиксирован, изображение остается в своем положении относительно видимой области окна браузера и не прокручивается с контентом.

Рис. 13.22. *Предотвращение прокручивания фонового изображения при помощи свойства* **background-attachment***

Однако если вы установите значение свойства **background-attachment** в **fixed**, оно остается там, где изначально располагалось, прокручивается только текст.

* Данный отрывок взят из сказки Леонида Филатова «Про Федота-стрельца, удалого молодца».

```
body {
background-image: url(images/bigstar.gif);
background-repeat: no-repeat;
background-position: center 300px;
background-attachment: fixed;
}
```

Значение **local**, добавленное в CSS3, заставляет фоновое изображение перемещаться вместе с контентом внутри элемента прокрутки, независимо от ползунка прокрутки окна браузера.

УПРАЖНЕНИЕ 13.5. ФИКСИРОВАННОЕ ПОЛОЖЕНИЕ

В последний раз, работая с меню бистро, мы применили к фону страницы крупный неповторяющийся логотип. Оставим это без изменений, но используем свойство **background-attachment**, чтобы зафиксировать изображение на заднем плане страницы, даже если страницу будут прокручивать.

```
body {
...
background-attachment: fixed;
}
```

Сохраните документ, откройте его в браузере, а теперь попробуйте прокрутить. Фоновое изображение остается расположенным в видимой области окна браузера.

В качестве дополнительного задания, посмотрите, что произойдет, когда вы зафиксируете положение точечного узора в **div#header**. (Подсказка: он останется на том же месте, но только внутри элемента **div**. Когда элемент **div** исчезнет из виду, то же произойдет и с его фоном.)

Свойства фона в CSS3

Модуль «Фоны и границы» спецификации CSS3 вводит еще несколько свойств для управления фоном. Модуль по-прежнему находится в черновом варианте, так что эта информация может измениться.

background-clip

Новый в CSS3

Принимаемые значения: **border-box** | **padding-box** | **content-box**

Данное свойство определяет, насколько далеко должно распространяться фоновое изображение. Мы видели, что по умолчанию оно простирается до края границы (**border-box**), но вы также можете заставить его оканчиваться у отступа или у границы блока контента, применив значения **padding-box** или **content-box** соответственно.

Мы обсудим эти компоненты блочной модели в следующей главе.

background-size

Новый в CSS3

Принимаемые значения: *значение длины* | *проценты* | **auto** | **cover** | **contain**

Это свойство позволяет дизайнерам задавать размер фонового изображения. Можно сообщить конкретную ширину и высоту. Если вы укажете только одно значение, вторым будет считаться **auto**. Вы также можете просто задать значение **contain**, которое изменит размер изображения таким образом, чтобы оно уместилось внутри элемента-контейнера, даже если останется лишнее пустое пространство, или **cover**, которое позволит охватить весь элемент, даже если часть фонового изображения выходит за края и исчезает из виду.

background-origin

Новый в CSS3

Принимаемые значения: **border-box** | **padding-box** | **content-box**

Определяет, как рассчитывается свойство **background-position**, другими словами, с чего начать отсчет измерения позиционирования. Вы можете начать с края границы, области отступа или области контента.

Сокращенное свойство фона

Вы можете использовать сокращенное свойство **background** для задания всех ваших стилей фона в одном определении.

background

Принимаемые значения: **background-color** **background-image** **background-repeat** **background-attachment** **background-position** | **inherit**

Значение по умолчанию: см. отдельные свойства

Применение: ко всем элементам

Наследование: нет

Так же, как и для сокращенного свойства **font**, суть свойства **background** — список значений, которые будут задаваться для отдельных свойств фона. Например, одно правило для фона:

```
body { background: black url(arlo.jpg) no-repeat right top fixed; }
```

заменяет правило с пятью отдельными определениями:

```
body {
background-color: black;
background-image: url(arlo.jpg);
background-repeat: no-repeat;
```

```
background-position: right top;
background-attachment: fixed;
}
```

Все значения свойства **background** необязательны и могут появляться в любом порядке. Единственное ограничение — это то, что когда задаются координаты для свойства **background-position**, горизонтальное значение должно указываться первым, а сразу за ним — вертикальное. Знайте, что если значение пропускается, оно будет подставлено по умолчанию (см. врезку «Будьте осторожны со случайными заменами»).

Будьте осторожны со случайными заменами

Свойство **background** эффективное, но используйте его осторожно. Когда вы пропускаете значение какого-либо свойства, вместо него будет использоваться значение по умолчанию. Следите, чтобы случайно не заменить ранние правила в таблице стилей более поздним сокращенным правилом, которое сбросит ваши настройки.

В этом примере фоновое изображение **dots.gif** *не* будет применено к элементам **h3**, из-за пропуска значения для свойства **background-image**, что, в сущности, устанавливает это значение равным **none**.

```
h1, h2, h3 { background: red url(dots.gif) repeat-x;}
h3 {background: green;}
```

Чтобы заменить конкретные параметры фона, используйте те свойства, которые вы собираетесь менять. Например, если в приведенном выше примере было намерение изменить только цвет фона элементов **h3**, то лучше применить свойство **background-color**.

УПРАЖНЕНИЕ 13.6. ПРЕОБРАЗОВАНИЕ В СОКРАЩЕННОЕ СВОЙСТВО

Это простое упражнение. Замените все определения, имеющие отношение к фону, в разделе **body** страницы меню при помощи единственного определения свойства **background**.

```
body {
font-family: Georgia,
serif;
font-size: 100%;
line-height: 175%;
margin: 0 15%;
background: #d2dc9d url(images/blackgoose.png) no-
repeat center 100px fixed;
}
```

Сделайте то же самое для элемента **div**, и все готово.

Множественные фоновые изображения

До недавнего времени к элементу можно было применить только одно фоновое изображение. В прошлом для наложения фоновых изображений единственным решением было добавить в разметку дополнительные элементы **div** и поместить по изображению в каждый из них. К счастью, CSS3 позволяет применять несколько фоновых изображений к одному элементу, и браузеры начинают их поддерживать.

ПРИМЕЧАНИЕ

Несмотря на то что определения CSS обычно работают по правилу «побеждает последний», в случае с множественными фоновыми изображениями, указанное последним располагается на заднем плане, а каждое изображение, указанное перед ним в списке, накладывается поверх него. Вы можете представить их, как слои в редакторе Photoshop, поскольку они тоже накладываются в том порядке, в каком появляются в списке. Самый первый рисунок списка находится сверху.

Чтобы применить несколько значений свойства **background-image** поместите их в список, разделив запятыми. Значения дополнительных свойств, имеющих отношение к фону, также перечисляются списком через запятую. Первое указанное значение относится к первому изображению, второе значение — ко второму, и так далее. Изображение, определяемое первым значением, будет идти первым, а остальные — выстраиваться за ним в том порядке, в котором они перечислены.

```
body {
background-image: url(image1.png), url(image2.png),
url(image3.png);
background-position: left top, center center, right bottom;
background-repeat: no-repeat; no-repeat; no-repeat;
…
}
```

Кроме того, вы можете воспользоваться сокращенным свойством **background**, чтобы упростить правило. Теперь у свойства **background** присутствует три набора значений, разделенных запятыми:

```
body {
background:
url(image1.png) left top no-repeat,
url(image2.png) center center no-repeat,
url(image3.png) right bottom no-repeat;
…
}
```

На рис. 13.23 показан результат.

До тех пор, пока поддержка множественных фоновых изображений не станет универсальной, вы можете использовать их как «украшения» для браузеров, которые способны их отобразить.

```
body {
background: url(image_fallback.png) top left no-repeat;
/* для неподдерживающих браузеров */
background:
url(image1.png) left top no-repeat,
url(image2.png) center center no-repeat,
url(image3.png) right bottom no-repeat;
background-color: papayawhip; /* цвет фона */
}
```

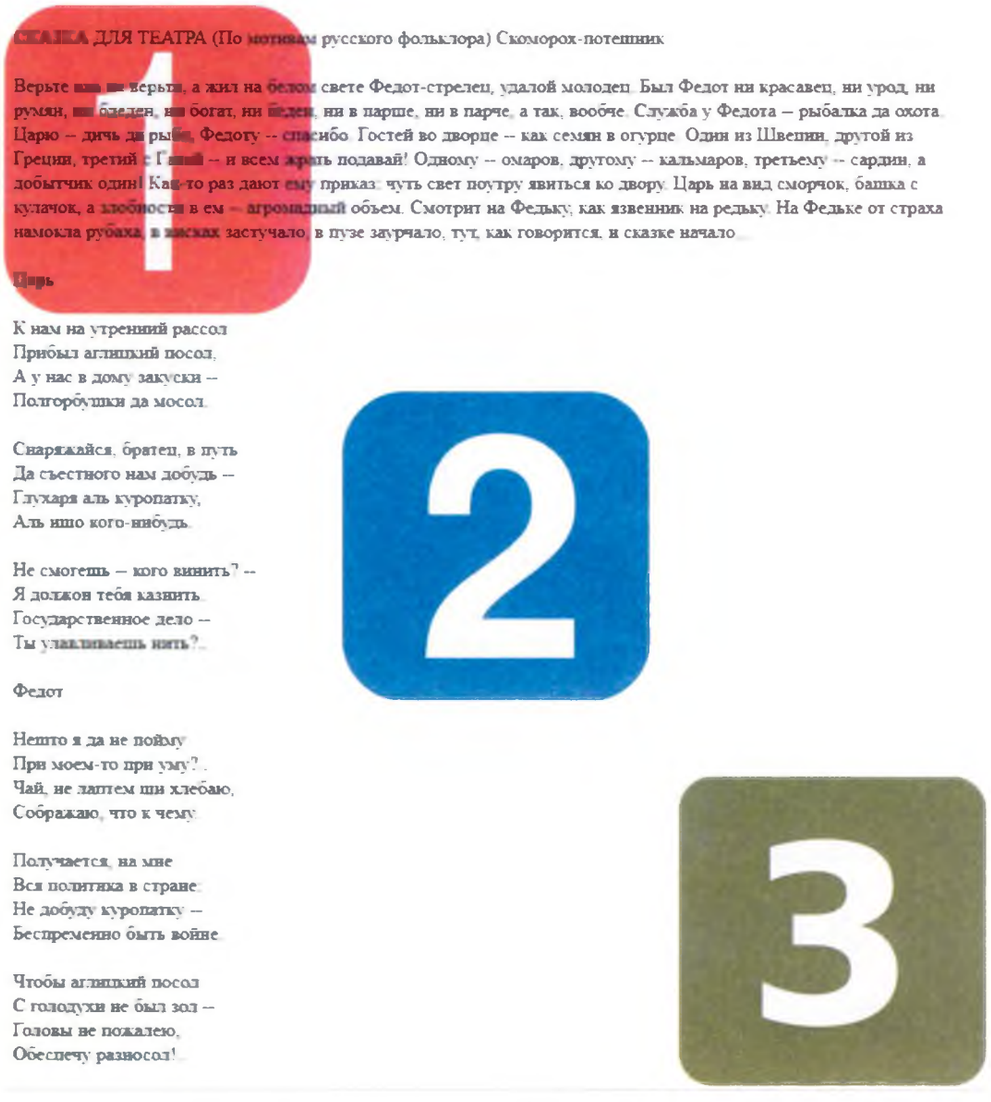


Рис. 13.23. Три отдельных фоновых изображения, добавленные к элементу **`body`**

УПРАЖНЕНИЕ 13.7 | МНОЖЕСТВЕННЫЕ ФОНОВЫЕ ИЗОБРАЖЕНИЯ

В этом упражнении мы попробуем применить множественные фоновые изображения. Обратите внимание, что, если вы пользуетесь браузером Internet Explorer версии 8 или ниже, вы не сможете увидеть множественные фоновые изображения, так что либо обновите браузер, либо используйте программу Chrome, Safari или Firefox.

Я хотела расположить узор из точек в элементе **`#header div`** по левой и правой сторонам. Также у меня есть маленький силуэт гуся (*gooseshadow.png*), который, наверное, будет мило смотреться в нижней части заголовка. В соответствии с лучшей на сегодняшний день техникой, я начала писать правило, начиная с запасного варианта для свойства **`background-image`** (горизонтальный ряд точек, который мы использовали ранее) и закончила цветом фона.

Параллакс-скроллинг с множественными фоновыми изображениями

Термин *параллакс движения* обозначает визуальный эффект, когда кажется, что объекты, расположенные вблизи движутся быстрее, чем объекты, находящиеся на большем расстоянии. В анимации или мультипликации, варьируя скорость объектов, находящихся на переднем, среднем и заднем планах, можно придать сцене подобие трехмерности. Некоторые дизайнеры используют множественные фоновые изображения для создания эффектов параллакс-скроллинга. При изменении размера окна браузера или перемещении горизонтальной полосы прокрутки, неравномерное движение фонов создает параллакс и трехмерный эффект. Поскольку вы не можете изменить размеры или прокручивать окно мобильного браузера по горизонтали, это не будет работать на смартфонах и планшетах.

Для получения более подробной информации посетите сайты:

- **habrahabr.ru/post/141687/**
- **www.coolwebmasters.com/animation/2666-parallax-scrolling-web-design.html**
- **ruseller.com/lessons.php?rub=29&id=968**

```
#header {
…
background-image: url(images/purpledot.png) center top repeat-x;
background:
url(images/purpledot.png) left top repeat-y,
url(images/purpledot.png) right top repeat-y,
url(images/gooseshadow.png) 90% bottom no-repeat;
background-color: rgba(255,255,255,.5);
}
```

Результат показан на рис. 13.24.

Рис. 13.24. Заголовок меню бистро с двумя рядами точек и небольшим изображением гуся в элементе **div#header**

Градиенты

Градиент — это переход от одного цвета к другому, иногда через несколько цветов. В прошлом единственным способом добавить градиент на веб-страницу было создать его в графическом редакторе и с помощью CSS добавить полученное изображение.

В CSS3 появилась возможность указать цвет градиента, используя только свойства стилей, оставив задачу визуализации сочетаний цветов бра-

узеру. Градиенты можно применять везде, можно добавить изображение: **background-image**, **border-image**, and **list-style-image**. В этой главе мы будем придерживаться примеров со свойством **background-image**.

Существует два типа градиентов:

- Линейные градиенты изменяют цвета вдоль линии от одного края элемента к другому.
- Радиальные градиенты начинаются с точки и распространяются наружу по кругу или эллипсу.

Линейные градиенты

Обозначение **linear-gradient()** обеспечивает угол линии и одну или несколько точек вдоль этой линии, где находится чистый цвет (*цветовые опорные точки* или *цветовые узлы*). Вы можете использовать имена цветов или любые численные значения цветов, обсуждавшиеся ранее в этой главе. Угол наклона линии указывается в градусах **(ndeg)** или с помощью зарезервированных слов. При использовании обозначений в градусах **0deg** указывает вверх, а положительный угол откладывается по часовой стрелке так, что значение **90deg** указывает вправо. Поэтому если вы хотите перейти от желтого цвета на верхней границе к зеленому на нижней, задайте угол вращения — **180deg**.

```
#banner
background-image: linear-gradient(180deg, yellow, green);
}
```

Зарезервированные слова описывают направление с шагом в 90° (**to top**, **to right**, **to bottom**, **to left**). Наш градиент на **180deg** также можно определить зарезервированным словом **to bottom**. Результат показан на рис. 13.25 (вверху).

```
#banner {
background-image: linear-gradient(to bottom, yellow, green);
}
```

В этом примере, градиент сейчас идет слева направо и включает в себя третий цвет, оранжевый, который появляется через 25% пути по линии градиента (рис. 13.25, в середине). Вы можете видеть, что размещение цветового узла указано после значения цвета. Позиции со значениями 0% и 100% можно опустить.

```
#banner {
background-image: linear-gradient(90deg, yellow, orange 25%,
blue);
}
```

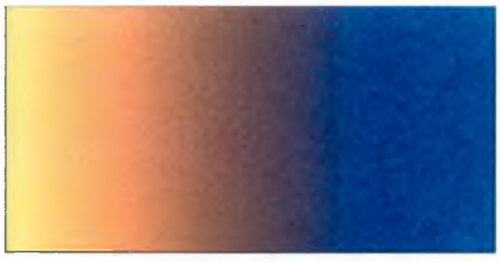
Эти примеры довольно яркие, но если подобрать цвета и цветовые узлы правильно, градиенты становятся хорошим способом придать элементам тонкие оттенки и даже трехмерность. Для кнопки фоновый гра-

Градиенты — это изображения, которые генерируются браузерами в реальном времени. Используйте их так же, как фоновое изображение.

диент используется, чтобы получить трехмерный эффект без помощи графики (рис. 13.25, внизу).

```
a.button-like {
background: linear-gradient(to bottom, #e2e2e2 0%, #dbdbdb 50%,
#d1d1d1 51%, #fefefe 100%);
}
```

linear-gradient(to bottom, yellow, green);



linear-gradient(90deg, yellow, orange 25%, blue);

ПРИМЕЧАНИЕ

Для получения более подробной информации о радиальных градиентах, см. сайт **htmlbook.ru/css3-na-primerakh/radialnyi-gradient**.

linear-gradient(to bottom, #e2e2e2 0%, #dbdbdb 50%, #d1d1d1 51%, #fefefe 100%);

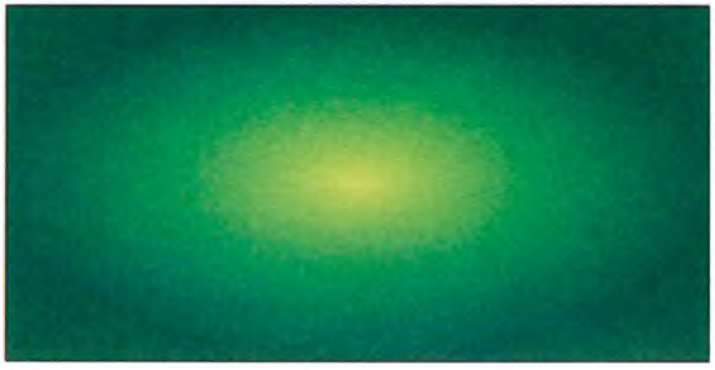
Рис. 13.25. *Примеры линейных градиентов*

Радиальные градиенты

Радиальные градиенты, как следует из названия, распространяются вовне из точки по кругу. В спецификации CSS3 обозначение **radial-gradient()** описывает фигуру (**circle** или **ellipse**; если фигура не указана, по умолчанию задается **circle**), положение центра градиента (согласно тому же синтаксису, что и **background-position**), и размеры, указанные как длина радиуса или зарезервированное слово, описывающее сторону или угол, где должен остановиться последний цвет (**closest-side**, **farthest-side**, **closest-corner**, **farthest-corner**, **cover**, **contain**).

Это правило заполняет блок радиальным градиентом, который будет находиться в пределах блока элемента (рис. 13.26).

```
#banner {
background-image: radial-gradient(center contain yellow
green);
}
```



radial-gradient(center, contain, yellow, green);

Рис. 13.26. Примеры радиальных градиентов

Вендорные префиксы

В примерах градиентов CSS, рассмотренных до сих пор, используется синтаксис, описанный в спецификации CSS3. Но браузеры принялись самостоятельно мастерить градиенты еще до того, как спецификация была полностью согласована. Для новейших функций типично, когда производители начинают экспериментировать с собственными решениями и реализациями в поставляемых браузерах до того, как спецификации оказываются полностью одобрены и закреплены.

Это очень похоже на то, что разработчики браузеров проделали в 90-х годах, вызвав множество проблем с совместимостью, но на этот раз у них хватило здравого смысла маркировать собственные свойства с помощью префиксов, которые четко указывают на то, что это собственные решения. В табл. 13.1 перечислены префиксы браузеров, используемые в настоящее время.

Табл. 13.1. Вендорные префиксы.

Префикс	Организация	Наиболее популярные браузеры
`-ms-`	Microsoft	Internet Explorer
`-moz-`	Mozilla Foundation	Firefox, Camino, Seamonkey
`-o-`	Opera Software	Opera, Opera Mini, Opera Mobile
`-webkit-`	Изначально Apple; теперь — с открытым исходным кодом	Safari, Chrome, Android, Silk, Blackberry, WebOS и многие другие
`-khtml-`	Konqueror	Konqueror

Для дизайнеров и разработчиков это означает, что при использовании некоторых новых особенностей CSS3, для размещения их в различных реализациях браузеров, необходимо перечислять свойства для каждого браузера с префиксами. Хотя приходится выполнять дополнительную работу и писать дополнительный код, это не плохо. Таким образом производители браузеров могут вводить инновации, которые не мешают процессу создания стандартов. Возвращаясь к градиентам, следующий пример показывает линейный градиент переходящий из желтого в зеленый, как он должен быть написан для любого современного браузера (добавим сюда еще эквивалент **filter** браузера Internet Explorer для полноты картины). Обратите внимание, что синтаксис немного отличается. Там, где в спецификации CSS3 спецификации используется зарезервированное слово **"to bottom"**, большинство других браузеров используют **"top"**, а Webkit — **"left top, left bottom"**.

```
background: #ffff00; /* Old browsers */

background: -moz-linear-gradient(top, #ffff00 0%, #00ff00
100%);

/* FF3.6+ */

background: -webkit-gradient(linear, left top, left bottom,
colorstop(

0%,#ffff00), color-stop(100%,#00ff00));

/* Chrome,Safari4+ */

background: -webkit-linear-gradient(top, #ffff00 0%,#00ff00
100%);

/* Chrome10+,Safari5.1+ */

background: -o-linear-gradient(top, #ffff00 0%,#00ff00 100%);

/* Opera 11.10+ */

background: -ms-linear-gradient(top, #ffff00 0%,#00ff00 100%);

/* IE10+ */

background: linear-gradient(to bottom, #ffff00 0%,#00ff00
100%);

/* W3C */

filter: progid:DXImageTransform.Microsoft.gradient(

startColorstr='#ffff00', endColorstr='#00ff00',GradientType=
0 );

/* IE6-9 */
```

Хорошая новость в том, что появляются новые версии браузеров, совместимые со стандартами, а старые уходят в прошлое, и некоторые свойства, такие как **text-shadow**, для которых когда-то были необходимы префиксы, теперь используются без них. Вполне возможно, что к тому времени, когда вы начнете читать эту книгу, префиксы браузеров останутся в прошлом. Но более вероятно, что все равно будет полезно знать вендорные префиксы и представлять, для каких именно свойств они нужны.

В следующих главах, когда для свойства будет требоваться префикс, я обязательно это отмечу. В противном случае можете предполагать, что стандарт CSS — это все, что вам нужно.

Дизайн градиентов

Последний пример кода был сногсшибателен! Даже без префиксов задача описания градиентов бывает сложной. И хотя можно написать код вручную, я рекомендую вам делать то же, что и я — использовать онлайн-инструмент создания градиентов!

Мой любимый — The Ultimate Градиент CSS Generator от компании ColorZilla (**www.colorzilla.com/gradient-editor/**), показанный на рис. 13.27. Просто укажите, сколько цветовых узлов вам нужно, перемещайте ползунки, пока не добьетесь нужного цвета, а затем скопируйте код. Именно это я и сделала, чтобы получить пример, который мы только что видели.

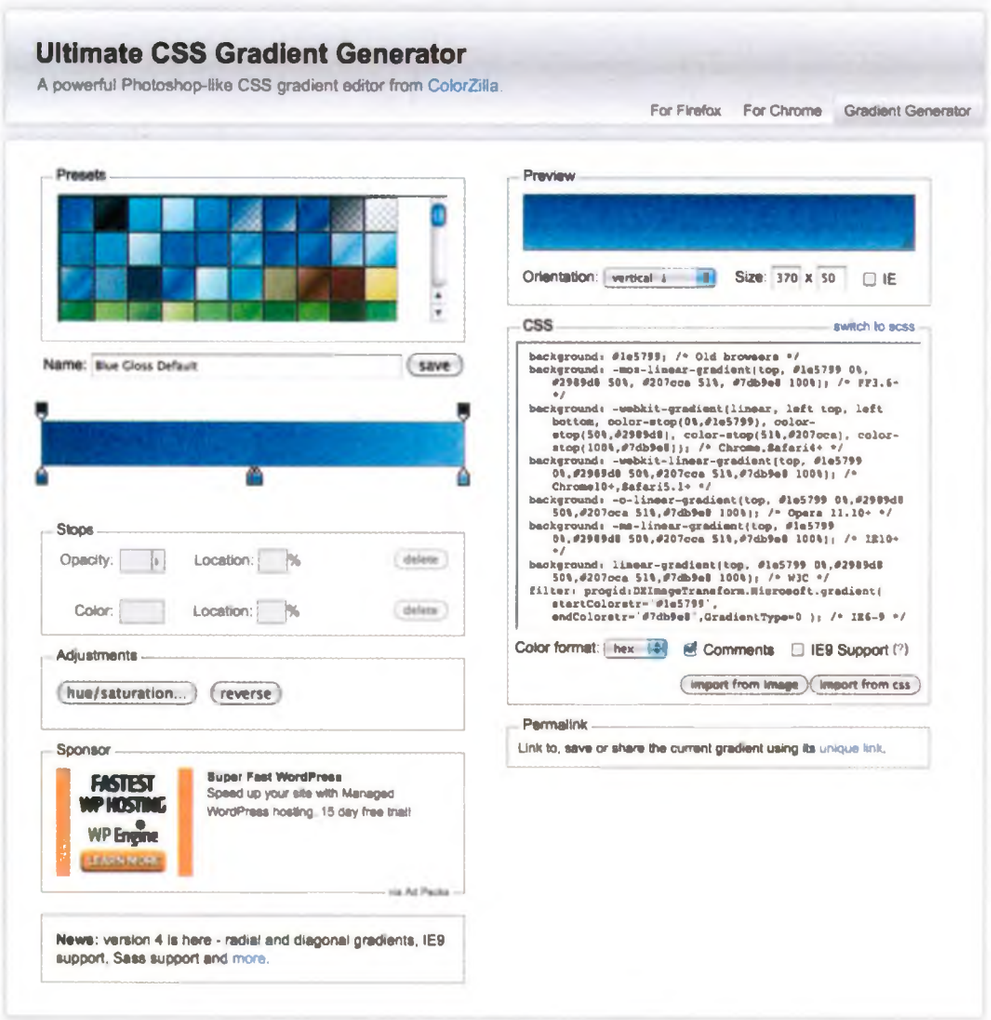


*Рис. 13.27. Создание градиента на сервисе Ultimate CSS Gradient Generator, **colorzilla.com/gradienteditor***

Для дополнительного чтения

Семь решений для преодоления кризиса с вендорными префиксами представлено в статье моего коллеги Крейга Баклера (**htmlbook.ru/blog/7-reshenii-dlya-preodoleniya-krizisa-s-vendornymi-prefi**).

Кроме того, прочитайте статьи по адресам **htmlbook.ru/blog/blizitsya-katastrofa-s-vendornymi-prefiksami** и **htmlbook.ru/blog/dva-novykh-predlozheniya-po-preodoleniyu-krizisa-s-vend**.

ПРИМЕЧАНИЕ

Более подробную информацию о синтаксисе градиентов для различных браузеров, а также о преимуществах градиентов над графикой читайте в статье Криса Койера (**css-tricks.com/css3-gradients/**).

Внешние таблицы стилей

Ранее, в главе 11, я сказала вам, что существует три способа присоединения таблиц стилей к HTML-документу: встроенные при помощи атрибута **style**, глобальные (определяются при помощи элемента **style**) и как внешний документ *.css*, связанный или импортированный в документ. В этом разделе мы наконец добрались до третьего варианта.

Внешние таблицы стилей — безусловно, самый мощный способ использования CSS, потому что вы можете вносить изменения стилей по всему сайту, просто редактируя единственный документ таблицы стилей. Это огромное преимущество — держать всю информацию о стиле в одном месте и не смешивать ее с исходным документом.

Для начала немного о самом документе *.css*. Внешняя таблица стилей — это простой текстовый файл, содержащий по крайней мере одно правило таблицы стилей. Он может *не* включать HTML-теги (в любом случае в этом нет смысла). Он может содержать комментарии, но они обязаны использовать синтаксис комментариев CSS, который вы уже видели:

```
/* Это конец раздела */
```

Файл таблицы стилей должен иметь имя с расширением *.css* (есть некоторые исключения из этого правила, но вы вряд ли встретитесь с ними так быстро). На рис. 13.28 показано, как небольшой документ таблицы стилей выглядит в моем текстовом редакторе.

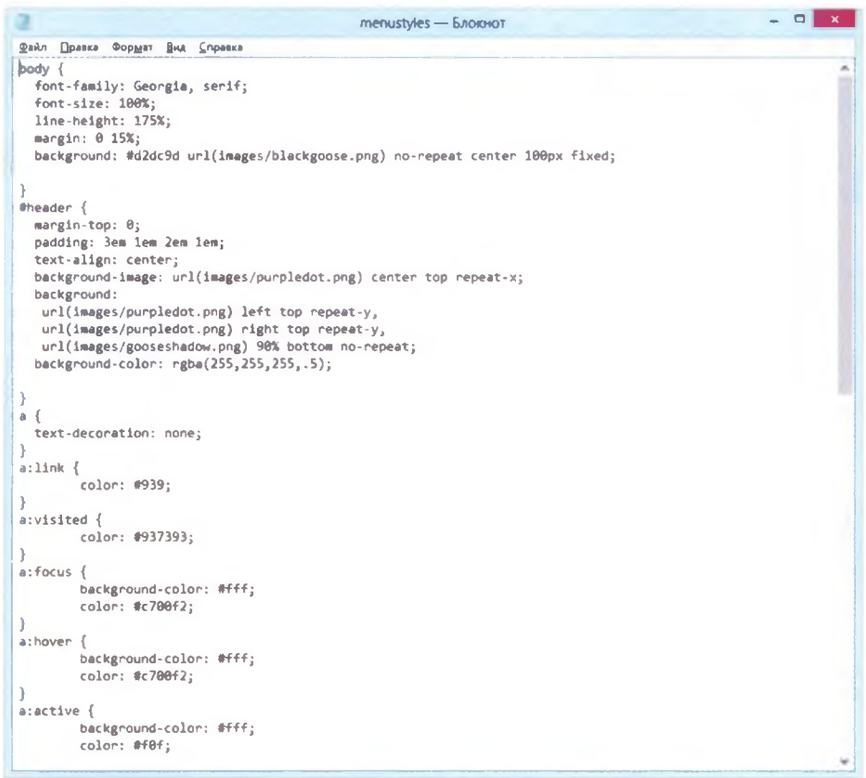


Рис. 13.28. *Внешние таблицы стилей содержат только правила CSS и комментарии в простом текстовом документе*

Существует два способа обращения к внешней таблице стилей изнутри HTML-документа: элемент **link** и правило **@import**. Мы рассмотрим оба этих метода.

Использование элемента link

Наиболее поддерживаемый метод — создать ссылку на *.css* документ, используя элемент **link** в заголовке **head** документа, как показано здесь:

```
<head>
<title>Название документа</title>
<link rel="stylesheet" href="/path/stylesheet.css">
</head>
```

Вам нужно включить два атрибута в элемент **link**:

rel="stylesheet"

Определяет отношения связываемого документа с текущим документом. Значением атрибута **rel** всегда является **stylesheet** при связи с таблицей стилей.

href="*url***"**

Задает расположение *.css* файла.

Вы можете включить множество элементов **link** для связи с различными таблицами стилей, и все они будут применены. Если возникнут конфликты, тот, который перечислен последним, заменит предыдущие вследствие порядка правил и каскадирования.

ПРИМЕЧАНИЕ

Элемент **link** пустой, так что вам нужно закрыть его при помощи завершающего слеша в XHTML-документах *(<**link** />)*. Не указывайте завершающий слеш в HTML-документах.

ПРИМЕЧАНИЕ

В HTML4.01 и XHTML1.0 элемент **link** должен содержать атрибут **type** со значением **text/css**:

type="text/css"

Атрибут **type** больше не требуется в спецификации HTML5.

Импортирование с использованием правила @import

Другой метод подключения внешних таблиц стилей к документу — импортирование их при помощи правила **@import.** Правило **@import** — это еще один тип правил, который необходимо добавить либо во внешний документ *.css* таблицы стилей, либо сразу в элемент **style**, как показано в примере ниже:

```
<head>
<style type="text/css">
@import url("path/stylesheet.css");
p { font-face: Verdana;}
</style>
<title>Название документа</title>
</head>
```

Здесь продемонстрирован относительный URL-адрес, но это мог быть и абсолютный URL (начинающийся с http://). Правило **@import** должно появляться в начале таблицы стилей и *указываться перед любыми селекторами*. Напоминаю: вы можете импортировать множество таблиц стилей, и они все будут применяться, но правила стилей из последнего перечисленного файла имеют преимущество над теми, что перечислены ранее.

Опробуйте оба метода, **link** и **@import**, в упражнении 13.3.

ПРИМЕЧАНИЕ

Вы можете также предоставить URL-адрес без определения **url()**:

```
@import "/path/style.css";
```

Абсолютные пути к файлу, начинающиеся в корне, гарантируют, что документ .css всегда будет найден.

УПРАЖНЕНИЕ 13.8. СОЗДАНИЕ ВНЕШНЕЙ ТАБЛИЦЫ СТИЛЕЙ

Создание глобальной таблицы стилей — распространенная практика при проектировании сайта. Но как только процесс завершен, лучше перенести данные во внешнюю таблицу стилей, чтобы она могла использоваться несколькими документами сайта.

Мы так и сделаем для таблицы стилей меню бистро.

1. Откройте последнюю версию файла *bistro.html*. Выделите и вырежьте все правила внутри элемента **style**, но оставьте теги **<style>...</style>**, потому что мы их еще будем использовать.
2. Создайте новый текстовый документ и вставьте все правила стилей. Убедитесь, что туда случайно не попали теги разметки.
3. Сохраните документ под именем *menustyles.css* в тот же каталог, что и файл *bistro.html*.
4. Теперь вернитесь к файлу *bistro.html*, в который добавьте правило **@import**, чтобы подключить внешнюю таблицу стилей, как показано ниже:

```
<style type="text/css">
@import url(menustyles.css);
</style>
```

Сохраните файл и обновите его в браузере. Он должен выглядеть точно так же, как выглядел, когда таблица стилей была глобальной. Если нет, убедитесь, что все шаги выполнены точно.

5. Удалите весь элемент **style**. На этот раз мы добавим таблицу стилей при помощи элемента **link** в заголовке документа — разделе **head**.

```
<link rel="stylesheet"
href="menustyles.css">
```

Снова проверьте свою работу, сохранив документ и обновив его в браузере.

Модульные таблицы стилей

Модульные таблицы стилей стали популярными из-за того, позволяют собрать информацию из множества внешних таблиц. Многие разработчики сохраняют часто используемые стили, такие как настройки форматирования текста, правила создания макета или веб-форм в отдельных таблицах стилей, затем комбинируют их, смешивая и подгоняя с помощью правил **`@import`**. Последние должны указываться перед правилами, которые используют селекторы.

Содержимое файла clientsite.css:

```
/* настройки форматирования текста */
@import url("type.css");
/* обработка форм */
@import url("forms.css");
/* навигация */
@import url("list-nav.css");
/* конкретные стили сайта */
body { background: orange; }
... другие правила стилей...
```

Это хорошая техника, о которой следует помнить, приобретая опыт создания сайтов. Вы узнаете, что существуют решения, работающие в вашем случае, и что не нужно «изобретать колесо» для каждого нового сайта. Модульные таблицы стилей — эффективный инструмент, экономящий время и облегчающий организацию (см. пояснение в примечании).

ПРИМЕЧАНИЕ

Несмотря на то что модульные таблицы стилей полезны, они могут стать проблемой для производительности компьютера и кеш-памяти. Если вы используете этот метод, рекомендуется объединить все таблицы стилей в один документ перед их загрузкой в браузер. Не волнуйтесь, вам не придется делать это вручную, существуют инструменты, которые выполнят работу за вас. Конструкции CSS LESS и SASS (которые официально будут представлены в главе 18 — два инструмента, предлагающие функции компилирования.

Резюме

Ниже приведена сводка свойств, охваченных в этой главе, в алфавитном порядке.

Свойство	Описание
`background`	Сокращенное свойство, которое объединяет свойства фона
`background-attachment`	Задает, прокручивается ли фоновое изображение или оно зафиксировано
`background-clip`	Определяет, насколько далеко должно простираться фоновое изображение
`background-color`	Задает цвет фона для элемента
`background-image`	Предоставляет путь к файлу изображения для использования его в качестве фона

Свойство	Описание
`background-origin`	Определяет, как рассчитывается свойство `background-position` (от края границы, отступа или блока контента)
`background-position`	Задает расположение начального фонового изображения
`background-repeat`	Повторяется ли фоновое изображение и как оно повторяется (мозаикой)
`background-size`	Задает размер фонового изображения
`color`	Задает основной цвет (текста и границ)
`opacity`	Задает уровень прозрачности переднего плана и фона

ГЛАВА 14

БЛОЧНАЯ МОДЕЛЬ CSS (ОТСТУПЫ, ГРАНИЦЫ И ПОЛЯ)

В главе 11 я ввела понятие *блочной модели* как одного из фундаментальных принципов CSS. В соответствии с ней каждый элемент в документе порождает *блок*, к которому можно применить такие свойства, как ширина, высота, отступы, границы и поля. Добавляя фон элементам, вы, вероятно, уже уяснили, как работают блоки. Эта глава охватывает все свойства, связанные с ними. Когда изучите основы, вы будете готовы перемещать блоки в главе 15.

Мы начнем с обзора компонентов блока элемента, затем рассмотрим его свойства, двигаясь изнутри наружу: размеры области контента, отступы, границы и поля.

В этой главе

- Компоненты блока элемента
- Задавание размеров блока
- Добавление отступов вокруг контента
- Добавление границ
- Добавление полей
- Назначение типов отображения
- Добавление тени

Блок элемента

Как мы уже видели, каждый элемент в документе, и блочный, и встроенный, порождает прямоугольный *блок элемента*. Его компоненты схематически изображены на рис. 14.1. Обратите внимание на новую терминологию — она пригодится далее для правильного понимания.

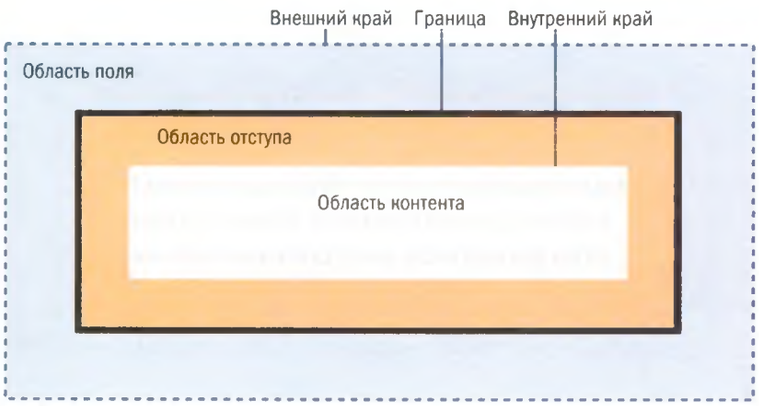


Рис. 14.1. *Компоненты блока элемента в соответствии с блочной моделью CSS*

Область контента

В центре блока элемента располагается контент. На рис. 14.1 область контента обозначена текстом в белом прямоугольнике.

Внутренние края

Границы области контента называются внутренними краями блока элемента. Хотя внутренние края отчетливо просматриваются благодаря видимому переходу между разными цветами (рис. 14.1), на реальных страницах они будут незаметны.

Отступы

Отступы— это пространство между областью контента и необязательной границей. На схеме область отступа выделена оранжевым цветом. Отступ не обязателен.

Граница

Граница — это линия (возможно, стилизованная), которая окружает элемент и его отступ. Границы также необязательны.

Поле

Поле (или внешний отступ) — это дополнительное пространство, добавленное *за пределами* границы. На схеме оно обозначено светло-голубым тоном, но в реальности поля всегда прозрачны, и сквозь них просвечивает фон родительского элемента.

Внешние края

Границы за пределами области поля составляют внешние края блока элемента. Это общая область элемента, которую он занимает на странице, и она включает в себя ширину области контента плюс общую сумму отступов, границ и полей, примененных к элементу. Внешние края на схеме обозначены пунктирными линиями, но на реальных веб-страницах они невидимы.

Общий размер блока элемента включает ширину контента плюс общую сумму отступов, границ и полей, примененных к элементу.

Все элементы имеют данные блочные компоненты; тем не менее, как вы увидите далее, некоторые свойства ведут себя по-разному в зависимости от того, является ли элемент блочным или встроенным. На самом деле мы увидим некоторые из этих отличий сразу же при рассмотрении размеров блока.

Задавание размеров блока

По умолчанию ширина и высота блока элемента рассчитываются автоматически (отсюда значение **auto**). Его ширина равняется ширине окна браузера или другого элемента, вмещающего данный блок, а высота зависит от размера контента. Однако вы можете задать области контента элемента определенную ширину и высоту с помощью свойств **width** и **height**

К сожалению, настройка размеров блока не ограничивается простым упоминанием этих свойств в таблице стилей. Вы должны точно знать, размеры какой части блока элемента указываете.

CSS3 предоставляет два способа определения размера элемента. По умолчанию используется метод, введенный еще в CSS1, который применяет значения ширины и высоты к области контента. Это означает, что в результате размер элемента будет складываться из указанных вами параметров *плюс* сумма отступов и границ, добавленных к элементу. Другой метод, введенный в спецификации CSS3 как часть нового свойства **box-sizing**, применяет значения ширины и высоты к *блоку границ*, который включает в себя контент, отступы и границы. При использовании этого метода получившийся *видимый блок элемента* с учетом отступов и границ будет точно такого размера, как вы укажете. Некоторые считают этот способ интуитивно более понятным. В данном разделе мы познакомимся с обоими методами.

Независимо от выбранного способа, вы можете указать ширину и высоту только блочных и нетекстовых встроенных элементов, таких как изображения.

Свойства **width** и **height** не применимы ко встроенным текстовым элементам (*незамещаемым*) и будут игнорироваться браузером. Другими словами, вы не можете указать ширину и высоту якорного элемента (**a**) или элемента **strong** (см. примечание).

width

Принимаемые значения: *значение длины* | *проценты* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: к блочным элементам и замещаемым встроенным элементам (таким как изображения)

Наследование: нет

height

Принимаемые значения: *значение длины* | *проценты* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: к блочным элементам и замещаемым встроенным элементам (таким как изображения)

Наследование: нет

ПРИМЕЧАНИЕ

На самом деле существует способ применения свойств **width** и **height** к якорям. Можно заставить их вести себя как блочные элементы при помощи свойства **display**, рассматриваемого в конце этой главы.

Задавание размеров области контента

По умолчанию свойства **width** и **height** применяются к блоку контента. Таким образом современные браузеры интерпретируют значения высоты и ширины, но вы можете четко указать их поведение, задав свойство **box-sizing: content-box**.

В следующем примере и на рис. 14.2 простому блоку задается ширина 500 пикселов, высота 150 пикселов с 20 пикселами отступа, границей в 2 и полями в 20 пикселов со всех сторон. При использовании модели области контента значения ширины и высоты применяются *только к области контента*.

```
{
background: #c2f670;
width: 500px;
height: 150px;
padding: 20px;
border: 2px solid gray;
margin: 20px;
}
```

В результате ширина *видимого* блока элемента будет 544 пиксела (контент плюс 40px отступа и 4px границы). Когда вы добавите 40 пикселов полей, общая ширина блока элемента станет 584 пиксела. Знать конечную ширину ваших элементов важно для создания макетов, которые ведут себя предсказуемо.

20px + 2px + 20px + 500px ширины + 20px + 2px + 20px = 584 пиксела

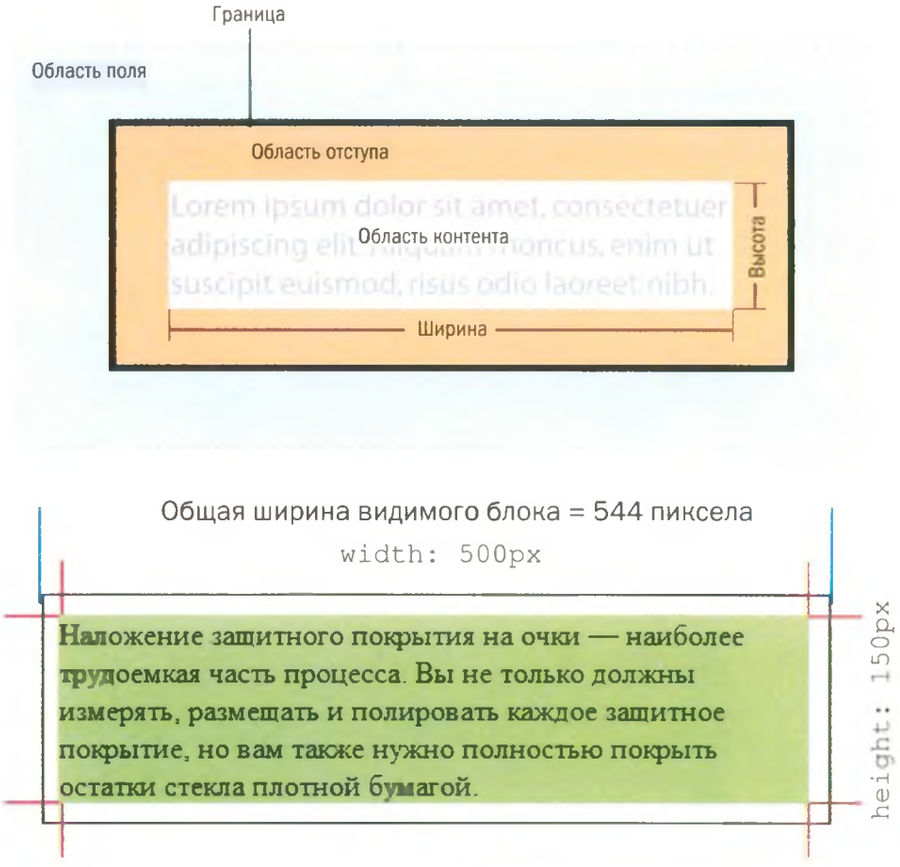


Рис. 14.2.** Задавание свойств **width** и **height** модели **content-box

Модель border-box

Другой способ указать размер элемента — обозначить ширину и высоту ко всему видимому блоку, включая отступы и границы. Так как это поведение не задано в браузере по умолчанию, вам нужно четко указать значение свойства **box-sizing: border-box** в таблице стилей.

Давайте рассмотрим тот же пример из предыдущего раздела и выясним, что произойдет, когда мы зададим значение 500 пикселов, используя метод **border-box** (рис. 14.3). Обратите внимание, что на момент написания книги необходимо было добавлять вендорные префиксы **-webkit** и **-moz**, чтобы заставить его работать в браузерах Safari, Chrome и Firefox. Браузеры Opera и Internet Explorer 8, а также более новые их версии поддерживают эту модель без префиксов (см. примечание).

```
p {
...
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
width: 500px;
height: 150px;
}
```

Многие разработчики считают модель **border-box** интуитивно более понятным способом задать размеры элементов. Она особенно полезна для определения ширины в процентах, что является краеугольным камнем адаптивного дизайна. Например можно сделать два столбца шириной 50% и знать, что они уместятся рядом друг с другом и не придется возиться с добавлением рассчитанной ширины отступов и границы к общей сумме. В самом деле, некоторые дизайнеры просто задают для

ПРЕДУПРЕЖДЕНИЕ

Старайтесь избегать использования значений **max-** и **min-** для ширины и высоты в модели **border-box**. Известно, что они вызывают ошибки в работе браузеров.

Максимальный и минимальный размеры

В спецификации CSS2 были введены свойства, чтобы задавать минимальную и максимальную высоту и ширину для блочных элементов. Они могут быть полезны, если вы хотите наложить ограничения на размер элемента.

max-height, **max-width**,

min-height, **min-width**

Принимаемые значения: *проценты* | *значение длины* | **none** | **inherit**

Эти свойства работают только с блочными и замещаемыми элементами, такими как изображения. При использовании модели **content-box** значение применяется только к области контента, поэтому если вы добавляете отступы, границы или поля, общий блок элемента станет больше, даже если были определены свойства **max-width** или **max-height**.

всех элементов в документе использование модели **border-box** с помощью универсального селектора:

```
* {box-sizing: border-box;}
```

Подробнее об этой технике читайте в статье habrahabr.ru/post/149441/.

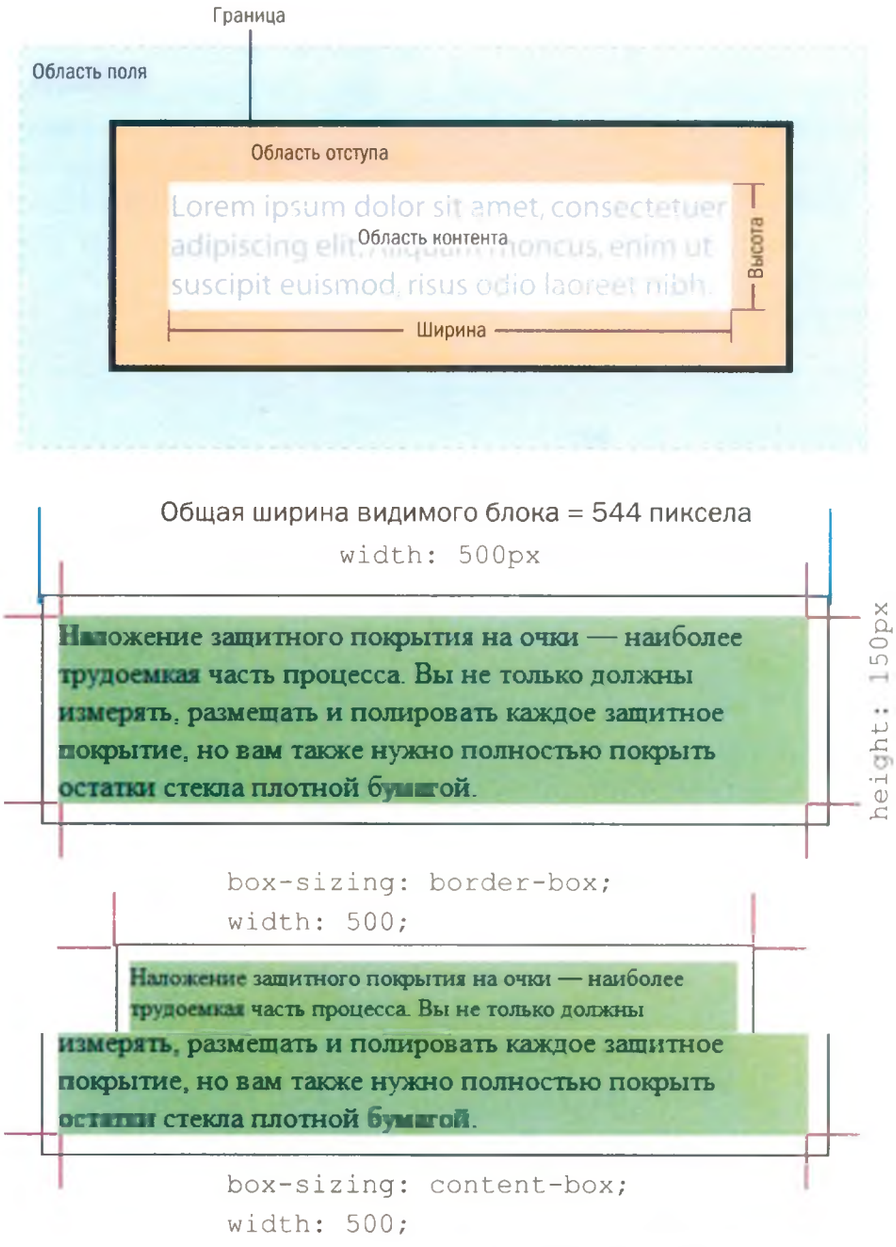


Рис. 14.3. *Изменение размеров элемента с использоаванием модели* ***border-box****. На рисунке внизу сравниваются блоки, получившиеся в результате применения обеих моделей*

Определение высоты

На практике у веб-дизайнеров не так часто возникает потребность в определении высоты элементов. В силу природы последних, высота вычисляется автоматически на основе размера текста и другого контента,

позволяя блоку элемента меняться согласно размеру шрифта, настроек пользователя или других факторов. Если вы зададите высоту элемента, содержащего текст, не забудьте также учесть, что должно происходить, если контент не соответствует размеру. К счастью, CSS предоставляет для этого несколько вариантов, как вы увидите в следующем разделе.

Выход контента за пределы блока

Когда элементу задан размер, который слишком мал для его содержимого, есть возможность определить дальнейшие действия, используя свойство **overflow**.

overflow

Принимаемые значения: **visible** | **hidden** | **scroll** | **auto** | **inherit**

Значение по умолчанию: **visible**

Применение: к блочным элементам и замещаемым встроенным элементам (таким как изображения)

Наследование: нет

Рис. 14.4 демонстрирует предопределенные значения свойства **overflow**. На рисунке показано применение различных значений к элементу, который является квадратом в 150 пикселов. Цвет фона делает края области контента видимыми.

visible

Значением по умолчанию является **visible**, которое позволяет контенту находиться поверх блока элемента, так что он весь может быть виден.

hidden

Когда свойство **overflow** установлено в значении **hidden**, содержимое, которое не помещается, обрезается и не показывается за краями области контента.

Рис. 14.4. Варианты обработки контента, выходящего за пределы свое области

ВНИМАНИЕ

Области прокрутки на мобильных устройствах

На момент написания книги свойство **overflow** славилось своей способностью вызывать проблемы на некоторых (в основном старых) мобильных устройствах, что весьма прискорбно, ведь небольшая область прокрутки на странице замечательно экономит пространство для определенного контента. Некоторые мобильные браузеры просто скрывают лишний текст, другие добавляют полосы прокрутки, но для управления ими требуется совершить движение двумя пальцами, о чем нелегко догадаться.

Одним из решений является использование сценария «Overthrow», написанного Скоттом Джелом для имитации поддержки в проблемных браузерах. См. веб-страницу **filamentgroup.com/lab/overthrow**.

scroll

Когда задано значение **scroll**, к блоку элемента добавляются полосы прокрутки, чтобы позволить пользователям просматривать все содержимое. Имейте в виду, что когда вы устанавливаете значение **scroll**, полосы прокрутки всегда будут отображены, даже если контент вмещается в блок.

auto

Значение **auto** позволяет браузеру решать, как обрабатывать выход за пределы. В большинстве случаев полосы прокрутки добавляются, только когда содержимое не помещается, и они необходимы.

Отступы

Отступ — это пространство между областью контента и границей (или позицией, где будет находиться граница, если отступ не определен). Я считаю, что полезно добавлять небольшой отступ к элементам при использовании цвета фона или границы. Это дает контенту немного свободного пространства и предотвращает наложение границы или края фона на текст.

Вы можете добавлять отступ к отдельным сторонам любого элемента (блочного или встроенного). Также существует сокращенное свойство **padding**, которое позволяет добавлять отступы ко всем сторонам одновременно.

padding-top, **padding-right**, **padding-bottom**, **padding-left**

Принимаемые значения: *значение длины* | *проценты* | **inherit**

Значение по умолчанию: **0**

Применение: ко всем элементам, за исключением **table-row**, **table-row-group**, **table-header-group**, **table-footer-group**, **table-column** и **table-column-group**

Наследование: нет

padding

Принимаемые значения: *значение длины* | *проценты* | **inherit**

Значение по умолчанию: **0**

Применение: ко всем элементам

Наследование: нет

При помощи свойств **padding-top**, **padding-right**, **padding-bottom**, **padding-left** вы можете задавать величину отступа для каждой стороны элемента, как показано в этом примере и на рис. 14.5 (обратите внимание, что я также добавила цвет фона, чтобы сделать края области отступа видимыми).

```
blockquote {
```

```
padding-top: 1em;
padding-right: 3em;
padding-bottom: 1em;
padding-left: 3em;
background-color: #D098D4;
}
```

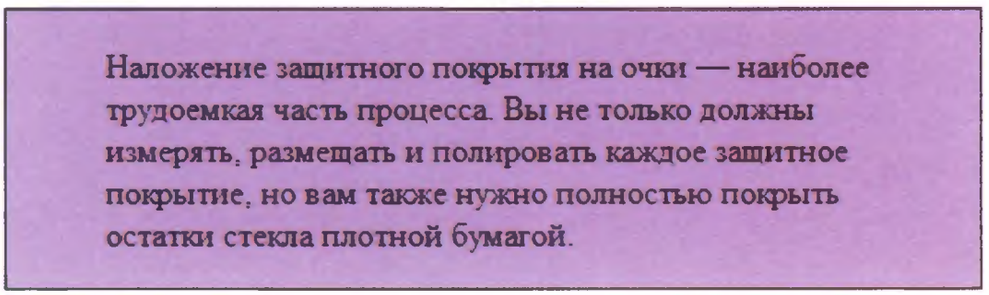


Рис. 14.5. Добавление отступов вокруг элемента

Вы можете задавать отступ в любых единицах измерения, используемых в CSS (единицы em и пикселы — наиболее часто применяемые) или в процентах от *ширины* родительского элемента. Да, ширина родительского элемента используется как основа даже для верхнего и нижнего отступов. Если она меняется, то значения отступов со всех сторон дочернего элемента также будут меняться, из-за чего значениями в процентах довольно сложно управлять.

Сокращенное свойство задавания отступов

Вместо того, чтобы задавать отступ с каждой стороны, вы можете использовать сокращенное свойство **padding** для одновременного добавления отступов со всех сторон элемента. Интересен синтаксис: вы можете определять четыре, три, два или одно значение для единственного свойства **padding**. Давайте посмотрим, как это работает, начиная с четырех значений.

Когда вы задаете четыре значения для свойства **padding**, они применяются по часовой стрелке к каждой стороне, начиная с верхней.

```
{ padding: сверху справа снизу слева; }
```

Используя свойство **padding**, мы могли бы воспроизвести отступы, заданные при помощи четырех отдельных свойств в предыдущем примере, как здесь:

```
blockquote {
padding: 1em 3em 1em 3em;
background-color: #D098D4;
}
```

НА ЗАМЕТКУ

Сокращенная запись значений

1 значение

```
padding: 10px;
```

Применяется ко всем сторонам

2 значения

```
padding: 10px 6px;
```

Первое значение для верхней и нижней сторон; второе — для левой и правой.

3 значения

```
padding: 10px 6px 4px;
```

Первое — для верхней стороны; второе — для левой и правой; третье — для нижней.

4 значения

```
padding: 10px 6px 4px 10px;
```

Значения применяются последовательно по часовой стрелке к верхнему, правому, нижнему и левому краю.

Если левый и правый отступы одинаковы, вы можете сократить свойство, задав только три значения. Значение «справа» (второе значение в строке) будет отражено и так же использовано для левой стороны. Как будто браузер считает значение «слева» пропущенным, поэтому использует значение «справа» для обеих сторон. Синтаксис для трех значений следующий:

```
{ padding: сверху справа/слева снизу; }
```

Это правило будет эквивалентно предыдущему примеру, потому что отступы на левом и правом краях элемента должны быть установлены равными 3em.

```
blockquote {
padding: 1em 3em 1em;
background-color: #D098D4;
}
```

Продолжая по этому шаблону, если вы задаете только два значения, первое используется для верхнего и нижнего краев, а второе — для левого и правого:

```
{ padding: сверху/снизу справа/слева; }
```

Такой же результат, какой был достигнут предыдущими двумя примерами, можно получить с помощью этого правила.

```
blockquote {
padding: 1em 3em;
background-color: #D098D4;
}
```

Обратите внимание, что все предыдущие примеры имеют одинаковый видимый результат, как показано на рис. 14.5.

Наконец, если вы зададите только одно значение, оно будет применено ко всем сторонам элемента. Например это определение применяет отступ в 15 пикселов ко всем сторонам элемента **div**:

```
div#announcement {
padding: 15px;
border: 1px solid;
}
```

УПРАЖНЕНИЕ 14.1. ДОБАВЛЕНИЕ НЕБОЛЬШОГО ОТСТУПА

В этом упражнении мы будем использовать основные свойства блока для улучшения внешнего вида страницы вымышленного интернет-магазина «Малышок». Я сильно облегчила вам задачу на начальном этапе, сверстав разметку исходного документа и создав таблицу стилей, которая управляет форматированием текста и фоном. Документ *malishok.html* доступен в папке *Глава 14* на диске, прилагающемся к книге.

На рис. 14.6 показаны снимки главной страницы магазина «до» и «после». Необходимо предпринять несколько шагов для того, чтобы эта страница приобрела презентабельный вид, и отступы — это только начало.

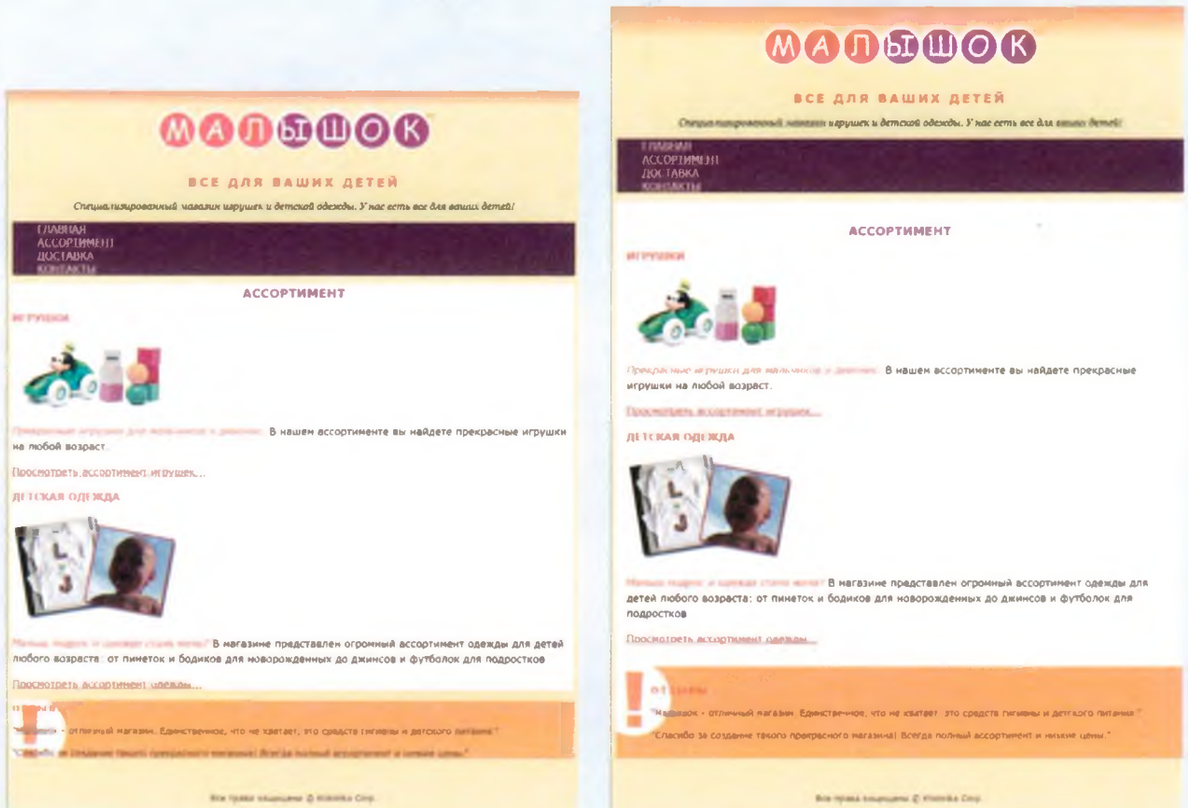


Рис. 14.6. *Снимки главной страницы Интернет-магазина Малышок «до» и «после»*

Область навигации просто ужасна! Но не волнуйтесь, мы превратим ее в симпатичное горизонтальное меню в главе 15.

Начните с открытия файла *malishok.html* в браузере и текстовом редакторе, чтобы видеть, с чем вы должны работать. Документ разделен на два элемента **div** («intro», и «content»), а элемент **#content div** поделен на разделы «products» и «testimonials». Цвета фона были добавлены к разделам **body**, **#nav**, **#products** и **#testimonials**. Также есть градиент в верхней части страницы и изображение восклицательного знака в верхнем левом углу раздела отзывов. Оставшиеся правила — для форматирования текста.

1. Первое, что мы сделаем, это добавим отступ к элементу **div** «products». Отступ в 1em со всех сторон должен подойти идеально. Найдите селектор **#products** и добавьте определение **padding**.

```
#products {
background-color: #FFF;
line-height: 1.5em;
padding: 1em;
}
```

2. Далее, мы немного пофантазируем при работе с разделом «testimonials». Я хочу очистить некоторое пространство в левой стороне элемента **div** так, чтобы мое стильное фоновое изображение восклицательного знака стало видимым. Существует несколько подходов к тому, как применять различные величины отступов к каждой стороне, но я собираюсь сделать это способом, который позволит вам попрактиковаться в сознательной замене более ранних определений.

Используйте сокращенное свойство **padding** для добавления отступа в 1em с каждой стороны раздела отзывов элемента **div**. Затем напишите второе определение, которое добавляет 55 пикселов отступа только к левой стороне. Из-за того, что определение **padding-left** указано вторым, оно заменит установленное значение 1em, примененное при помощи свойства **padding**.

```
#testimonials {
background: #FFBC53 url(images/ex-circle-corner.gif) no-repeat
left top;
color: #633;
font-size: .875em;
line-height: 1.5em;
padding: 1em;
padding-left: 55px;
}
```

3. Сохраните документ и просмотрите результат в браузере. Отзывы и описания продукции должны выглядеть привлекательнее в своих блоках. На рис. 14.7 выделены добавленные отступы.

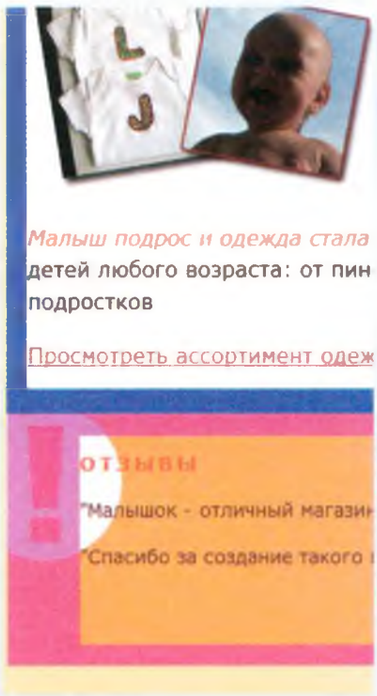


Рис. 14.7. *Область, выделенная розовым цветом, обозначает отступы, добавленные к разделу отзывов. Область, помеченная синим цветом, обозначает отступы, добавленные к разделу продукции*

Границы

Граница — это обычная линия, нарисованная вокруг области контента и ее отступов (при наличии таковых). Вы можете выбирать любые из восьми стилей границ и делать их какой угодно ширины и цвета. Можете обозначить границу вокруг всего элемента или только с одной или нескольких сторон. В CSS3 введены свойства для скругления углов или применения изображений к границам. Мы начнем с различных стилей границ.

Стиль границы

Стиль — это самое важное из свойств границы, потому что в соответствии со спецификацией CSS, если не задан стиль границы, она не существует. Другими словами, прежде всего вы должны объявить стиль границы, иначе другие свойства будут игнорироваться.

Стили границы могут быть применены к одной стороне за один раз или с использованием сокращенного свойства **border-style**.

border-top-style, **border-right-style**, **border-bottom-style**, **border-left-style**

Принимаемые значения: **none** | **dotted** | **dashed** | **solid** | **double** | **groove** | **ridge** | **inset** | **outset** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: нет

border-style

Принимаемые значения: **none** | **dotted** | **dashed** | **solid** | **double** | **groove** | **ridge** | **inset** | **outset** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: нет

Значением свойств **border-style** является одно из десяти зарезервированных слов, описывающих доступные стили границ, как показано на рис. 14.8.

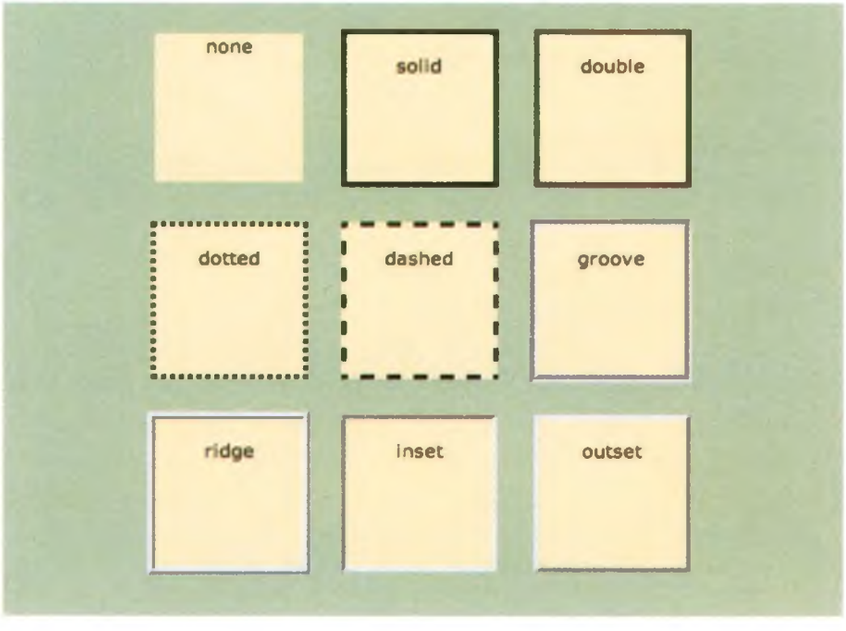


Рис. 14.8. *Доступные стили границ (показаны со средним значением ширины по умолчанию)*

Вы можете использовать свойства стилей границ для конкретной стороны (**border-top-style**, **border-right-style**, **border-bottom-style** и **border-left-style**), чтобы применить стиль к одной стороне элемента. Если вы не задаете ширину, будет взято среднее значение по умолчанию. Если не задан оттенок, граница использует основной цвет элемента (такой же, как цвет текста).

СОВЕТ ДИЗАЙНЕРА

Нижние границы вместо подчеркивания

Отключение подчеркивания ссылок и замена его пользовательской нижней границей — распространенный прием в современном веб-дизайне. Это улучшает внешний вид ссылок, в то же время выделяя их на фоне обычного текста.

В следующем примере я применила разный стиль к каждой стороне элемента, чтобы показать в действии свойства границ, определенные для одной стороны (рис. 14.9).

```
div#silly {
border-top-style: solid;
border-right-style: dashed;
border-bottom-style: double;
border-left-style: dotted;
width: 300px;
height: 100px;}
```

Сокращенное свойство **border-style** действует по часовой стрелке, как описывалось ранее для свойства **padding**. Вы можете задать четыре значения для всех сторон или меньше, если левая/правая и верхняя/нижняя границы одинаковые. Нелепый эффект границы в предыдущем примере мог быть также определен при помощи свойства **border-style**, как показано здесь, и результат был бы таким же, как на рис. 14.9.

```
border-style: solid dashed double dotted;
```

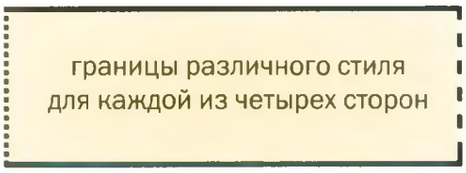


Рис. 14.9. *Стили границ, примененные к отдельным сторонам элемента*

Ширина границы (толщина)

Используйте одно из свойств ширины границы, чтобы указать ее толщину. Напоминаю: вы можете задать ширину для каждой стороны элемента при помощи свойства для конкретной стороны или сделать это сразу для нескольких сторон при помощи сокращенного свойства **border-width**.

border-top-width, **border-right-width**, **border-bottom-width**, **border-left-width**

Принимаемые значения: *значение длины* | **thin** | **medium** | **thick** | **inherit**

Значения по умолчанию: **medium**

Применение: ко всем элементам

Наследование: нет

border-width

Принимаемые значения: *значение длины* | **thin** | **medium** | **thick** | **inherit**

Значения по умолчанию: **medium**

Применение: ко всем элементам

Наследование: нет

Наиболее распространенный способ определения ширины границы — использование измерений в пикселах или единицах em; однако вы также можете задать одно из зарезервированных слов (**thin**, **medium** или **thick**) и предоставить представление браузеру.

В этом примере использованы различные варианты (рис. 14.10). Обратите внимание, что я также включила свойство **border-style**, ведь в противном случае граница не отобразилась бы совсем.

```
div#help {
border-top-width: thin;
border-right-width: medium;
border-bottom-width: thick;
border-left-width: 12px;
border-style: solid;
width: 300px;
height: 100px; }
```

или

```
div#help {
border-width: thin medium thick 12px;
border-style: solid;
width: 300px;
height: 100px; }
```

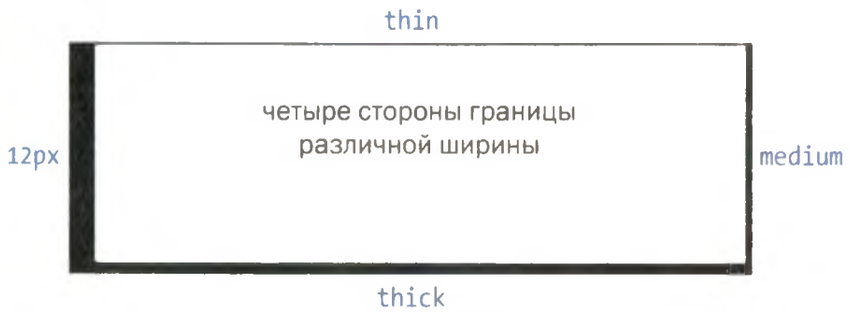


Рис. 14.10. Определение ширины границы

Цвет границы

Цвета границы задаются аналогичным способом — с использованием свойств для конкретных сторон или сокращенного свойства **border-color**. Когда вы задаете цвет границы, он заменяет основной цвет, который установлен свойством **color** для элемента.

СОВЕТ ДИЗАЙНЕРА

Почти незаметные контуры

Задавая цвет правила близкий фоновому и устанавливая достаточную ширину, мы можем добиться очень мягкого эффекта, создавая скорее текстуру, чем контур.

`border-top-color`, `border-right-color`, `border-bottom-color`, `border-left-color`

Принимаемые значения: *имя цвета или значение по модели RGB* | **`transparent`** | **`inherit`**

Значение по умолчанию: значение свойства **`color`** для элемента

Применение: ко всем элементам

Наследование: нет

`border-color`

Принимаемые значения: *имя цвета или значение по модели RGB* | **`transparent`** | **`inherit`**

Значение по умолчанию: значение свойства **`color`** для элемента

Применение: ко всем элементам

Наследование: нет

Теперь вы знаете все о том, как задавать значения цветов, и должны привыкать к сокращенным свойствам, поэтому следующий пример я сделаю коротким и удобным (рис. 14.11). Здесь заданы два значения для сокращенного свойства **`border-color`**, чтобы окрасить верхнюю и нижнюю стороны границы элемента **`div`** темно-бордовым цветом, а левую и правую —цветом морской волны.

```
div#special {
border-color: maroon aqua;
border-style: solid;
border-width: 6px;
width: 300px;
height: 100px;
}
```

верхняя и нижняя стороны темно-бордового цвета, а левая и правая —цвета морской волны

Рис. 14.11. *Определение цвета границ*

ПРИМЕЧАНИЕ

В спецификации CSS2 было добавлено зарезервированное значение **`transparent`** для цветов границ, которое позволяет фону родительского элемента просвечивать сквозь них, а также сохраняет заданную ширину границы. Это может быть полезно для создания при помощи CSS эффектов, возникающих при наведении мыши на элемент (**`:hover`**), потому что пространство, где появится граница, сохраняется, даже когда курсор не указывает на элемент.

Комбинирование стилей, ширины и цвета

Разработчики CSS не скупились, когда дело дошло до сокращенных записей свойств границ. Они также создали свойства, чтобы задавать значения стилей, ширины и цвета в одном определении. Напомню, вы можете задавать вид конкретных сторон или использовать свойство **`border`** для изменения всех четырех сторон за один раз.

border-top, **border-right**, **border-bottom**, **border-left**

Принимаемые значения: ***border-style border-width border-color*** | **inherit**

Значение по умолчанию: значения по умолчанию для каждого свойства

Применение: ко всем элементам

Наследование: нет

border

Принимаемые значения: ***border-style border-width border-color*** | **inherit** ***Значение по умолчанию***: значения по умолчанию для каждого свойства

Применение: ко всем элементам

Наследование: нет

Значения для **border** и свойств для конкретных сторон границы могут включать в себя значения стиля, ширины и цвета в любом порядке. Вам не надо объявлять все три, но имейте в виду, что если значение стиля границы пропущено, она не будет отображаться.

Свойство **border** работает немного иначе, чем другие рассмотренные нами сокращенные свойства. Оно обрабатывает один набор значений и всегда применяет его ко всем четырем сторонам элемента, то есть не использует систему «по часовой стрелке».

Здесь приведено несколько правильных примеров сокращенных свойств границы, чтобы вы получили представление о том, как они работают.

```
h1 { border-left: red .5em solid; }      /* только левая граница */
h2 { border-bottom: 1px solid; }         /* только нижняя граница */
p.example { border: 2px dotted #663; }   /* все четыре стороны границы */
```

Скругление углов при помощи свойства border-radius

Блоки со скругленными углами стали модным элементом стиля в последние годы. Первоначально скругленные углы на веб-страницах можно было сделать только с помощью изображений и дополнительной разметки. К счастью, на сегодняшний день все актуальные версии браузеров способны добавлять к элементам скругленные углы с помощью одного только свойства CSS **border-radius**. Это означает меньше обращений к серверу для скачивания графики и меньше работы в Photoshop для дизайнеров. В данном разделе мы начнем с кода, как он прописан в спецификации CSS3, затем рассмотрим примеры, а в завершение я скажу несколько слов о поддержке браузеров.

Как мы уже видели, существуют отдельные свойства для каждого угла, а также сокращенное свойство **border-radius**.

border-top-left-radius, border-top-right-radius,border-bottom-right-radius, border-bottom-left-radius

Новое в CSS3

Принимаемые значения: *значение длины | проценты*

Значение по умолчанию: 0

Применение: ко всем элементам

Наследование: нет

border-radius

Новое в CSS3

Принимаемые значения: *1, 2, 3 или 4 значения длины или процентных значения*

Значение по умолчанию: 0

Применение: ко всем элементам

Наследование: нет

Чтобы закруглить угол элемента, просто примените одно из свойств **border-radius**, но имейте в виду, что вы увидите результат, только если у элемента есть граница или цвет его фона отличается от фона страницы. Значения, как правило, представлены в единицах em или пикселах. Проценты также можно использовать, и они хорошо подходят для сохранения пропорций скругленных углов по отношению к блоку при изменении его размера, но вы можете столкнуться с несоответствиями в браузерах.

Рис. 14.12. Закруглите углы блоков элементов с помощью свойств ***border-radius***

Изменяйте углы по отдельности или используйте сокращенное свойство **border-radius**. Если вы предоставите одно значение для свойства **border-radius**, оно будет применено ко всем углам. Четыре значения применяются по часовой стрелке, начиная с верхнего левого угла (верхний левый, верхний правый, нижний правый, нижний левый). Когда

вы указываете два значения, первое применяется к верхнему левому и нижнему правому углам, а второе — к двум оставшимся.

Сравните различные значения свойства **border-radius** получившихся блоков на рис. 14.12. Вы можете добиться разнообразных эффектов, от слабого смягчения углов до длинной ромбовидной фигуры, в зависимости от того, как зададите значения.

Овальные углы

До сих пор наши углы были фрагментами идеальных окружностей, но вы можете также сделать угол овальным, указав два значения: первое — для горизонтального радиуса, а второе — для вертикального(см. рис. 14.13, А и Б).

```
А. border-top-right-radius: 100px 50px;
Б. border-top-right-radius: 50px 20px;
border-top-left-radius: 50px 20px;
```

При использовании сокращенного свойства горизонтальный и вертикальный радиусы разделяются слешем (в противном случае браузер примет их за значения разных углов). Следующий пример задает на всех углах горизонтальный радиус 60px и вертикальный радиус 40px (рис. 14.13, В).

```
В. border-radius: 60px / 40px;
```

Если вы хотите увидеть что-то действительно увлекательное, взгляните на сокращенное свойство **border-radius**, которое определяет различные овалы для каждого из четырех углов. Все горизонтальные значения перечисляются в левой стороне от слеша по часовой стрелке (верхний левый, верхний правый, нижний правый, нижний левый), а все соответствующие вертикальные значения — в правой (рис. 14.13, Г).

```
Г. border-radius: 36px 40px 60px 20px / 12px 10px 30px 36px;
```

Как я упоминала ранее, текущие версии всех основных браузеров теперь поддерживают свойство **border-radius**, используя синтаксис спецификации CSS3. Это хорошая новость!

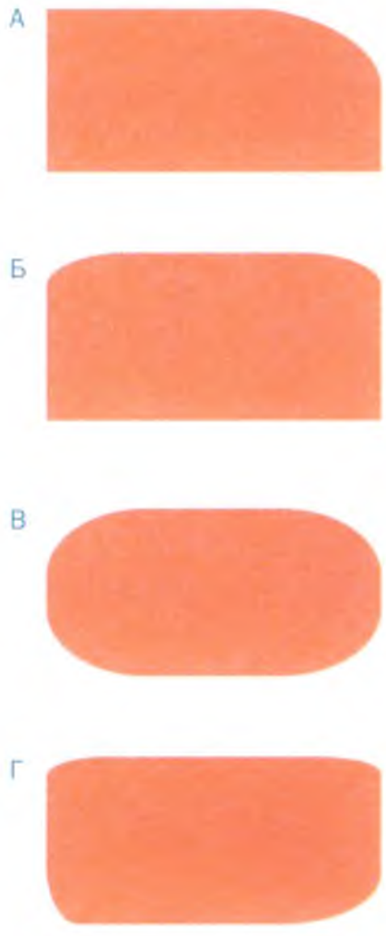


Рис. 14.13. Применение к блокам углов овальной формы

Красивые границы

Мы уже несколько страниц обсуждаем границы в CSS. Я оставила самый интересный и сложный прием их создания напоследок. В этом разделе мы рассмотрим использование свойства **border-image** для заполнения сторон и углов границы блока изображением по вашему выбору. Данное свойство избавляет от необходимости нарезать отдельные файлы изображений и добавлять кучу бесполезной разметки, чтобы вместить их. Теперь ко всему элементу можно применить одно изображение с помощью CSS.

Следует отметить, что браузер Internet Explorer поддерживает изображения границ только начиная с версии 11.

`border-image`

Новое в CSS3

Принимаемые значения: `border-image-source border-image-slice border-image-width border-image-outset border-image-repeat`

Значение по умолчанию: значения по умолчанию, предустановленные для каждого свойства

Применение: ко всем элементам, кроме табличных, где свойство `border-collapse` означает `collapse`

Наследование: нет

Давайте начнем обсуждение с наглядного примера, чтобы дать вам представление, о чем я говорю. На рис. 14.14 показаны два элемента и соответствующие изображения, используемые для заполнения их границ. Обратите внимание, что углы изображения точно заполняют углы элемента. Для сторон элемента можно задать растягивание (как показано на рисунке вверху) или мозаичный узор (внизу).

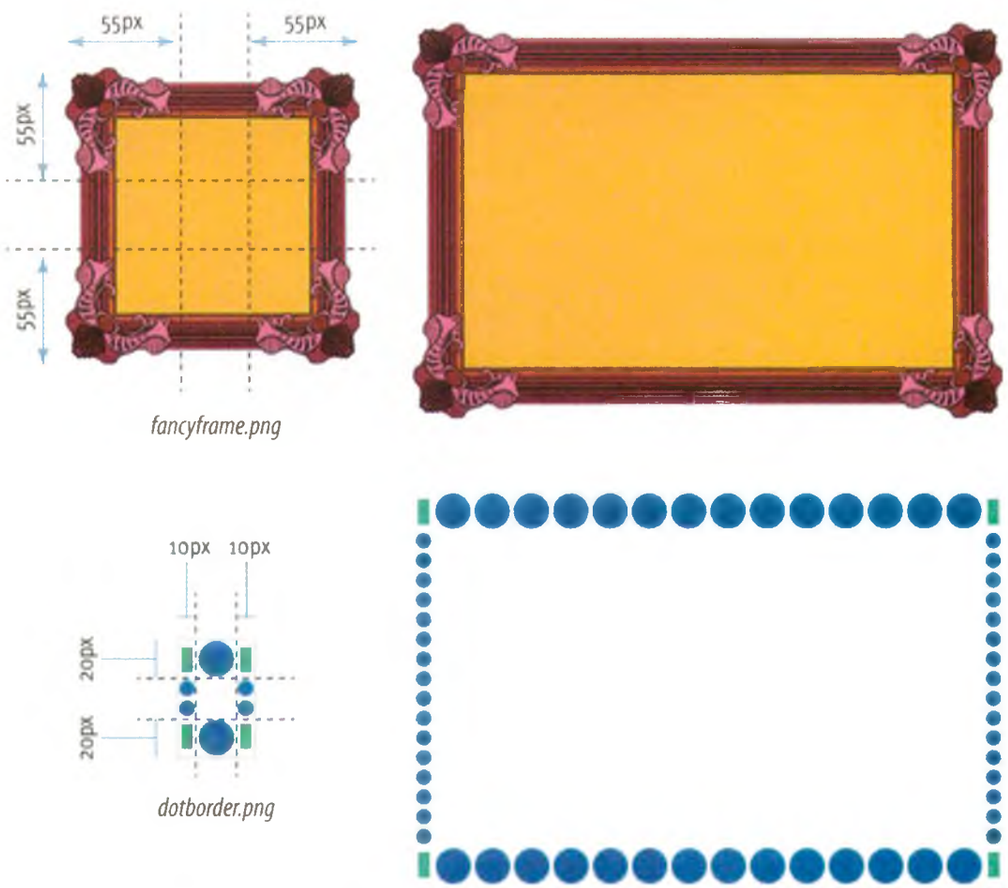


Рис. 14.14. *Примеры изображений границы с растянутыми и повторяющимися изображениями по сторонам*

Теперь код. Сокращенное свойство `border-image` в том виде, в каком оно поддерживалось на момент написания книги, состоит из трех частей (рис. 14.15). Более подробную информацию читайте во врезке «Спецификация свойства `border-image`».

Рис. 14.15. Части правила для свойства **`border-image`**

URL-адрес (**А**) указывает на расположение файла изображения для границы. Следующее значение, сообщает положение линий разреза, разделяющих изображение на девять частей (**Б**). Значения представляют собой смещения от каждого края изображения, перечисленные по часовой стрелке (сверху, справа, снизу, слева). Можно использовать сокращенные значения, например предоставить одно значение для смещения линий разреза на одинаковое расстояние от всех четырех краев. При указании значений в пикселах вы можете опустить обозначение **`px`**. Также допустимо указывать процентные значения.

Зарезервированное слово в конце описывает, как заполнить стороны границы (**В**). Значения: **`stretch`** (растягивает изображение до нужного размера), **`repeat`** (укладывает изображение мозаикой) и **`round`** (повторяет изображение, но немного растягивает или сжимает его, чтобы уместить целиком).

Ниже приведено правило стилей, создающее интересное изображение границы в виде рамки на рис. 14.14 (вверху). Я пока не использую вендорные префиксы, чтобы не усложнять его.

```
.framed {
…
background-color: #fec227; /* яркий желто-оранжевый оттенок*/
border-color: #fec227; /* яркий желто-оранжевый оттенок */
border-style: solid;
border-width: 55px;
border-image: url(fancyframe.png) 55 stretch;
}
```

Источник изображения для границы — файл *fancyframe.png*. Так как точки разреза одинаковы по всем четырем сторонам (55 пикселов), указать значение 55 требуется только один раз (помните, что не нужно указывать единицу измерения для пикселов). Наконец, зарезервированное слово **`stretch`** сообщает, что стороны блока будут заполнены путем растягивания сторон рисунка. Как запасной вариант я указала цвет фона и границы такого же яркого желто-оранжевого оттенка, как и центральная часть изображения для границы.

В браузере Internet Explorer у изображений будет блок такого же размера и цвета, но без обрамляющего рисунка (см. примечание).

ПРИМЕЧАНИЕ

Различные типы изображений для границы могут предполагать иные запасные решения, но так как конкретно это было широким, мне показалось, что лучше всего применить к нему заливку сплошным цветом.

Спецификация свойства border-image

Согласно спецификации CSS3, **border-image** — это сокращенное свойство, включающее в себя следующие пять отдельных свойств:

border-image-source

Указывает URL-адрес изображения, которое будет использоваться.

border-image-slice

Обеспечивает значения для четырех линий разреза, перечисленных по часовой стрелке.

border-image-width

Определяет ширину границы на основе значений, указанных по часовой стрелке.

border-image-outset

Указывает расстояние, на которое изображение выступает за границу.

border-image-repeat

Сообщает, каким образом следует заполнить стороны (**stretch**, **repeat** или **round**)

В настоящее время браузеры не поддерживают эти свойства по отдельности, поэтому для добавления изображений на границы вам всегда нужно использовать сокращенное свойство **border-image.**

Кроме того, поскольку свойство **border-image-width** вызывает ошибки в браузерах, а **border-image-outset** не поддерживается ни одним из них, я не стала обсуждать эти свойства, говоря о **border-image**, и оставила только три части, приведенные на рис. 14.15.

Ниже показано, как выглядит правило со всеми префиксами браузера. Если решитесь добавить изображение в качестве границы, ваши правила будут выглядеть так же. Не забудьте стандартное свойство без префиксов указать последним.

```
.framed {
…
background-color: #fec227; /* яркий желто-оранжевый оттенок*/
border-color: #fec227; /* яркий желто-оранжевый оттенок */
border-style: solid;
border-width: 55px;
-moz-border-image: url(fancyframe.png) 55 stretch;
-webkit-border-image: url(fancyframe.png) 55 stretch;
-o-border-image: url(fancyframe.png) 55 stretch;
border-image: url(fancyframe.png) 55 stretch;
}
```

Ниже приведено правило стилей для отображения границы, состоящей из точек. Оно отличается тем, что значения ширины верхней и боковых сторон различны (отсюда два значения свойств **border-image-slice** и **border-width**), я также задала значение повтора **round**, чтобы заполнить пространство повторяющимся мозаичным узором, размер которого изменен для точного соответствия. Обратите внимание, что браузеры Webkit в настоящее время отображают значение **round** как простое повторение (**repeat**).

```
.dotted {
background-color: white;
border-color: #0063a4;
border-style: dotted;
border-width: 20px 10px;
-moz-border-image: url(dotborder.png) 20 10 round;
-webkit-border-image: url(dotborder.png) 20 10 round;
-o-border-image: url(dotborder.png) 20 10 round;
border-image: url(dotborder.png) 20 10 round;
}
```

Теперь пришло время поработать с границами. Упражнение 14.2 не только позволит вам попрактиковаться, но и подскажет некоторые идеи того, как придать привлекательность дизайну страниц.

ПРИМЕЧАНИЕ

Согласно спецификации CSS3, браузеры не должны визуализировать центр изображения по умолчанию, но на сегодняшний день все они визуализируют центр изображения в центре элемента.

Центральная область растягивается или повторяется так же, как указано для границ. Если вы хотите предоставить в блоке контента другой цвет фона или фоновое изображение, создайте изображение для границы с прозрачной центральной областью, чтобы сквозь нее был виден фон.

УПРАЖНЕНИЕ 14.2. ПРИЕМЫ РАБОТЫ С ГРАНИЦАМИ

В этом упражнении мы поработаем с границами на главной странице интернет-магазина «Малышок». Помимо добавления тонких границ вокруг разделов с контентом, мы используем границы для усиления заголовков продукции и как альтернативу подчеркиваниям ссылок.

1. Откройте файл *malishok.html* в текстовом редакторе, если он еще не открыт. Начнем с использования сокращенного свойства **border**, чтобы получить пунктирную светло-оранжевую (**#FFBC53**) линию вокруг раздела «products». Добавьте новое определение к правилу для элемента **div** «products».

```
#products {
…
border: double #FFBC53;
}
```

2. Далее придадим разделу «testimonials» скругленные углы. Обратите внимание, они не будут видны в браузере Internet Explorer версии 8 и ниже.

```
#testimonials {
…
border-radius: 20px;
}
```

3. Ради практики мы поместим декоративные границы с двух сторон заголовков в разделе продукции (**h3**). Я хочу, чтобы границы были того же цвета, что и текст, поэтому не надо определять свойство **border-color**. Найдите существующее правило для элементов **h3** в элементе **div** «products» и укажите определение, которое добавляет 1-пиксельную сплошную линию в верхнюю часть заголовка. Напишите другое определение, которое добавляет более толстую, 3-пиксельную сплошную линию к левой стороне и небольшой (1em) отступ слева от контента заголовка.

```
#products h3 {
font-size: 1em;
text-transform: uppercase;
color: #F26521;
```

```
border-top: 1px solid;
border-left: 3px solid;
padding-left: 1em;
}
```

4. Последнее, что мы сделаем, это заменим стандартное подчеркивание текста под ссылками декоративной границей снизу. Начните с отключения подчеркивания для всех состояний ссылок, установив свойство **text-decoration** в **none** для элемента **a**. Добавьте правило в таблицу стилей.

```
a {
text-decoration: none;
}
```

5. Далее создайте 1-пиксельную точечную границу по нижнему краю ссылок, добавив это определение к каждому правилу для них:

```
a {
text-decoration: none;
border-bottom: 1px dotted;
}
```

Обратите внимание, что если мы хотим, чтобы граница имела такой же цвет, как и ссылки, нам не надо задавать цвет.

Однако если вы пробуете это на ваших собственных страницах, то легко сможете изменить цвет и стиль нижней границы.

Когда вы добавляете границу к элементу, неплохо также добавить небольшой отступ, чтобы защитить элементы от наложения друг на друга. Добавьте некоторый отступ только к нижнему краю, как показано в примере:

```
a {
text-decoration: none;
border-bottom: 1px dotted;
padding-bottom: .1em;
}
```

См. рис. 14.16 для того, чтобы увидеть, как выглядит страница.

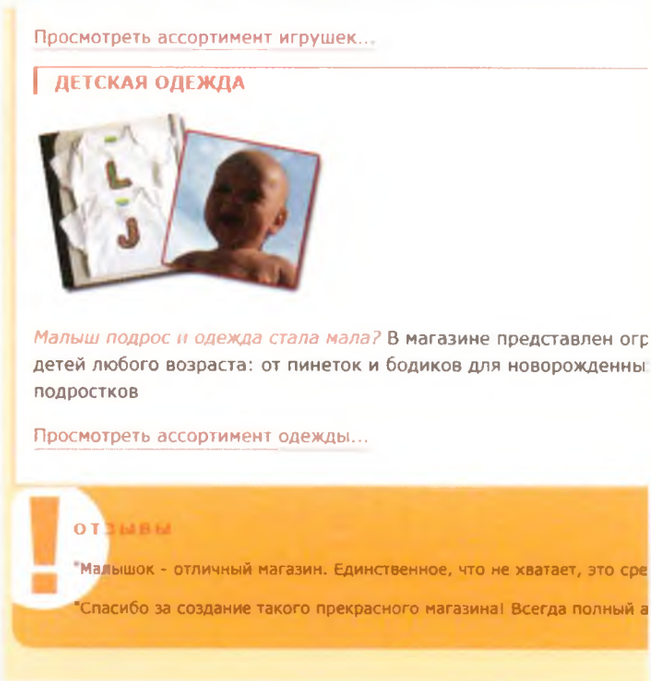


Рис. 14.16. *Результаты добавления границ*

Поля

Последний оставшийся компонент блока элемента — это его поле, то есть пространство, которое вы можете добавлять к внешней стороне границы. Поля защищают элементы от наложения друг на друга или выхода за край окна браузера. Их можно использовать, даже чтобы создать пространство для другой колонки с контентом (мы увидим, как это работает в главе 16). Таким образом, поля являются важным инструментом в верстке страниц на основе CSS.

Свойства для конкретных сторон и сокращенное свойство **margin** работают очень схоже со свойствами задавания отступов, которые мы уже рассмотрели, однако, знайте: поля имеют некоторое особое поведение.

margin-top, **margin-right**, **margin-bottom**, **margin-left**

Принимаемые значения: *значение длины* | *проценты* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: ко всем элементам

Наследование: нет

margin

Принимаемые значения: *значение длины* | *проценты* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: ко всем элементам, кроме элементов типов отображения таблицы, отличных от **table-caption**, **table** и **inline-table**

Наследование: нет

Свойства задавания полей используются прямо. Вы можете задать величину поля для каждой стороны элемента или использовать свойство **margin**, чтобы задать поля со всех сторон одновременно.

Свойство **margin** работает так же, как и сокращенное свойство **padding**. Когда вы задаете четыре значения, они применяются по часовой стрелке (верх, право, низ, лево) к сторонам элемента. Если вы задаете три значения, центральное применяется и к левой, и к правой сторонам. Когда задаются два значения, первое используется для верхней и нижней сторон, а второе применяется к левой и правой. Наконец, одно значение будет применяться ко всем четырем сторонам элемента.

Как и для большинства размеров во Всемирной паутине, наиболее распространенные единицы измерения для этого — em, пикселы и проценты. Имейте в виду, однако, что если вы задаете процентное значение, оно вычисляется на основе *ширины* родительского элемента. Если ширина родительского элемента меняется, также будут меняться поля со всех четырех сторон дочернего элемента (такое же поведение демонстрируют и отступы). Зарезервированное слово **auto** позволяет браузеру предоставить величину поля, необходимую для того, чтобы вместиться или заполнить доступное пространство.

На рис. 14.17 показаны результаты следующих примеров применения полей. Обратите внимание, что я добавила светлую точечную линию, чтобы обозначить внешний край поля, только для пояснения смысла, но она бы не появилась на реальной веб-странице.

А.
```
p#A {
    margin: 4em;
    border: 1px solid red;
    background: #FCF2BE;
}
```

Б.
```
p#B {
    margin-top: 2em;
```

СОВЕТ ПО ТАБЛИЦАМ CSS

Поля браузера по умолчанию

Вы могли заметить, что пространство добавляется автоматически вокруг заголовков, абзацев и других блочных элементов. Это работа таблицы стилей браузера по умолчанию, добавляющей величины полей выше и ниже тех элементов.

Хорошо бы иметь в виду, что браузер применяет свои собственные значения для полей и отступов «за кадром». Они будут использоваться, пока вы специально не замените их вашими собственными правилами стилей.

Если вы работаете над дизайном и встречаете загадочное количество пространства, которое вы не добавляли, виновниками могут быть стили браузера по умолчанию.

Одним из решений будет установить значение всех отступов и полей для всех элементов равное 0, что обсуждается в главе 18.

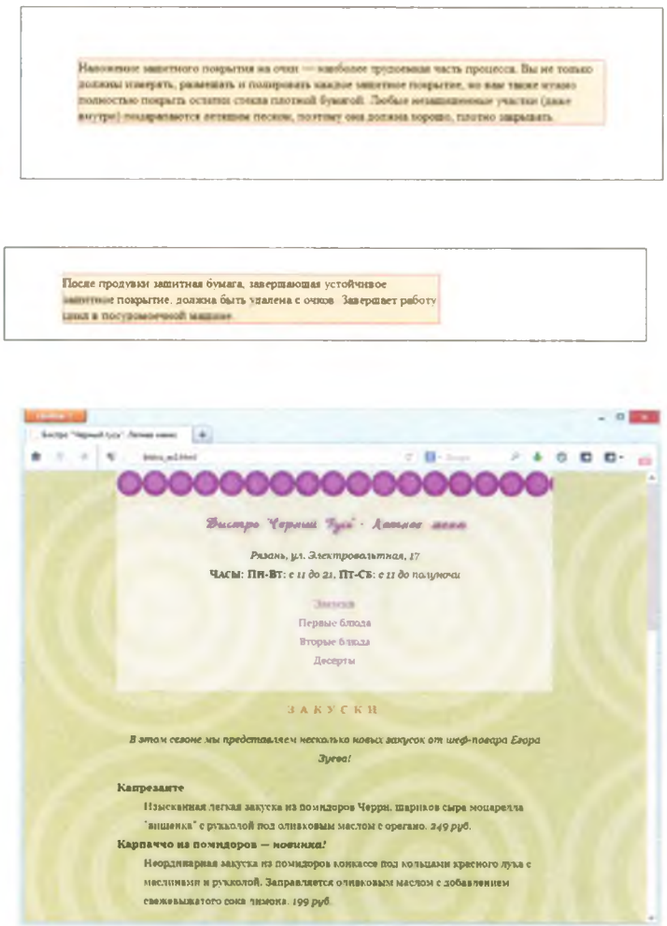
ПРИМЕЧАНИЕ

Добавление поля к элементу **body** добавляет пространство между контентом страницы и краями окна браузера.

```
    margin-right: 250px;
    margin-bottom: 1em;
    margin-left: 4em;
    border: 1px solid red;
    background: #FCF2BE;
}
```

В.

```
body {
    margin: 0 15%;
    border: 1px solid red;
    background-color: #FCF2BE;
}
```



А

`margin: 4em;`

Б

`margin-top: 2em;`
`margin-right: 250px;`
`margin-bottom: 1em;`
`margin-left: 4em;`

В

`body {margin: 0 15%}`

Добавление полей к **body** размещает пространство между элементом и краями видимой области окна браузера. Граница показывает пределы элемента **body** (отступы не применялись).

Рис. 14.17. Применение полей к **body** *и к отдельным элементам*

Поведение полей

Писать правила, которые добавляют поля вокруг HTML-элементов, легко, а вот поведение полей — тема более сложная.

Схлопывание полей

Наиболее существенное поведение полей, которое надо знать, это то, что верхние и нижние поля соседних элементов *схлопываются*. Это означает, что вместо суммирования смежные поля совмещаются, и будет использоваться только самое большое значение.

Используем два абзаца из предыдущего рисунка в качестве примера. Если верхний элемент имеет нижнее поле величиной 4em, а находящийся ниже элемент — верхнее поле величиной 2em, итоговая область поля между элементами не соответствует 6em. Наоборот, поля совмещаются, и получившееся в результате поле между абзацами будет величиной 4em, как самое большое из заданных значений. Это продемонстрировано на рис. 14.18.

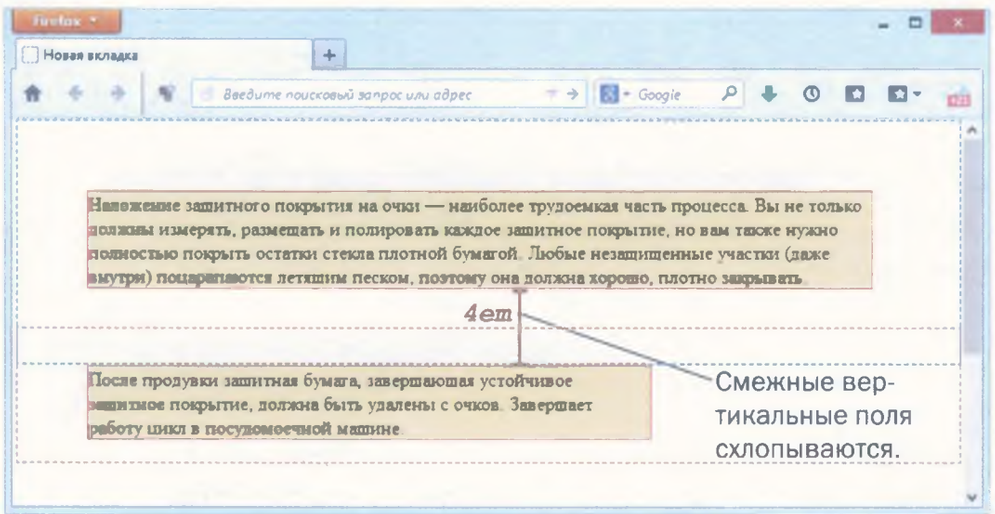


Рис. 14.18. *Вертикальные поля соседних элементов схлопываются, так что будет использоваться только наибольшее значение*

В одном только случае верхнее и нижнее поля *не* схлопываются — для обтекаемых или абсолютно позиционированных элементов (мы разберемся с обтеканием и позиционированием в главе 15). Левое и правое поля никогда не схлопываются, поэтому они предсказуемы.

Поля встроенных элементов

Вы можете применить верхнее и нижнее поле к встроенным текстовым элементам (или «не замещаемым встроенным элементам», если использовать правильную CSS терминологию), но это не добавит вертикальное пространство выше и ниже элемента, и высота строки не изменится. Однако когда вы применяете левое и правое поле к встроенным тестовым элементам, пространство поля *будет* сохраняться перед и после текста в потоке элемента, даже если этот элемент разрывает несколько строк.

Просто ради сохранения привлекательности поля на замещаемых элементах, таких как изображения, отображаются на всех сторонах, и поэ-

ДЛЯ ДАЛЬНЕЙШЕГО ЧТЕНИЯ

Схлопывание полей

Когда пространство между и вокруг элементов ведет себя непредсказуемо, всегда обвиняют схлопывающиеся поля. Ниже приведено несколько статей, которые глубже рассматривают их поведение.

- «Схлопывание margin» (**www.xiper.net/learn/css/box-model/margin-collapsing.html**)
- «Границы и отступы в потоке» (**softwaremaniacs.org/blog/2005/09/05/css-layout-flow-margins/**)
- «Обходим схлопывание margin» (**www.xiper.net/collect/html-and-css-tricks/pozitsionirovanie/margin-collapsing-trick.html**)
- «Uncollapsing Margins» (**www.complexspiral.com/publications/uncollapsing-margins/**)

тому оказывают воздействие на высоту строки. Смотрите примеры вышеописанного на рис. 14.19.

```
em { margin: 2em;}
```

Только горизонтальные поля отображаются у не замещаемых (текстовых) элементов.

Наложение защитного покрытия на очки — наиболее трудоемкая часть процесса. Вы не только должны измерять, размещать и полировать каждое защитное покрытие, но вам также нужно *полностью* покрыть остатки стекла плотной бумагой. Любые незащищенные участки (даже внутри) поцарапаются летящим песком, поэтому она должна хорошо, плотно закрывать.

```
img { margin: 2em;}
```

Поля отображаются со всех сторон замещаемых элементов, таких как изображения.

Наложение защитного покрытия на очки — наиболее трудоемкая часть процесса. Вы не только должны измерять,

размещать и полировать каждое защитное покрытие, но вам также нужно полностью покрыть остатки стекла плотной бумагой. Любые незащищенные участки (даже внутри) поцарапаются летящим песком, поэтому она должна хорошо, плотно закрывать.

Рис. 14.19. *Поля, примененные к встроенным текстовым элементам и изображениям*

Отрицательные значения полей

Стоит отметить, что возможно определять отрицательные значения для полей. Когда вы это делаете, контент, отступы и граница перемещаются в направлении, противоположном тому, которое получилось бы при положительном значении поля.

Это прояснится на примере. На рис. 14.20 показаны два соседствующих абзаца с границами разного оттенка, примененными для демонстрации их пределов. Как показано на левом изображении, я добавила нижнее поле величиной 4em к верхней стороне абзаца, и в результате расположенный ниже абзац сместился *вниз* на эту величину. Если я определю отрицательное значение (-4em), расположенный ниже элемент пере-

`p.top { margin-bottom: 4em; }`

Отодвигает расположенный ниже элемент абзаца на величину 4em.

`p.top { margin-bottom: -4em; }`

Расположенный ниже элемент перемещается вверх на величину 4em.

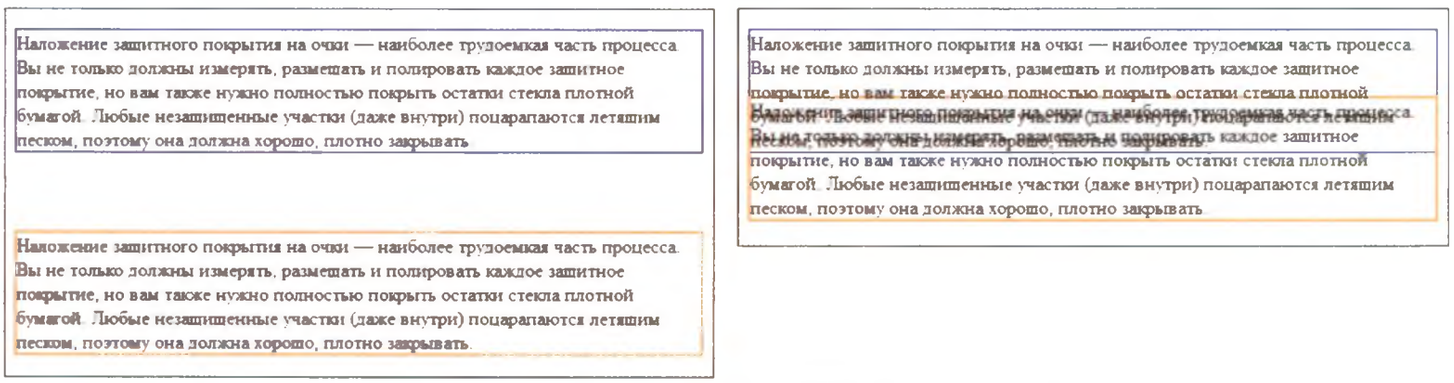


Рис. 14.20. *Использование отрицательных значений полей*

двинется *вверх* на эту величину, и наложится на элемент с отрицательным полем.

Это действие кажется странным, и на самом деле вы вряд ли станете заставлять блоки текста накладываться друг на друга, как показано. Смысл в том, что вы можете использовать поля как с положительными значениями, так и с отрицательными, для перемещения элементов по странице. Это основа многих технических приемов CSS.

Теперь давайте используем поля для добавления некоторого пространства между частями главной страницы интернет-магазина в упражнении 14.3. Понимание приемов использования отступов, границ и полей является первым шагом в овладении техникой верстки на основе CSS. В следующей главе вы узнаете о свойствах, используемых для обтекания и позиционирования элементов на странице. Мы даже превратим страницу интернет-магазина «Малышок» в макет с двумя колонками. Но перед этим рассмотрим еще одно свойство.

ПРИМЕЧАНИЕ

Когда значение равняется 0, вам необязательно указывать конкретную единицу измерения.

УПРАЖНЕНИЕ 14.3. ДОБАВЛЕНИЕ ОБЛАСТИ ПОЛЯ ВОКРУГ ЭЛЕМЕНТОВ

Откройте файл *malishok.html* в текстовом редакторе, если он еще не открыт, и мы настроим поля. Начнем с настройки полей всего документа целиком, а потом подправим каждый раздел сверху вниз.

1. Распространенная практика — установить поля элемента **body** равными нулю, убирая таким образом установленные браузером по умолчанию настройки полей и создавая стартовую точку, чтобы задать собственные поля элементов по всей странице.

```
body {
…
margin: 0;
}
```

Сохраните файл и просмотрите страницу в браузере. Мне нравится, как фиолетовая панель навигации простирается от одного края окна к другому, но, думаю, стоит настроить другие области контента.

2. Начните с элемента **#intro div** и добавьте поле величиной 2em к верхнему и 1em к нижнему краю. Я также хочу убрать пространство между логотипом и слоганом, поэтому задайте нижнее поле элемента **h1** равное нулю, а верхнее поле элемента **h2** установите равным 10px, чтобы переместить слоган вверх и поближе к логотипу. Наконец, установите поле величиной 1em вокруг вводного абзаца (**p**).

```
#intro {
…
margin: 2em 0 1em;
}
#intro h1 {
margin-bottom: 0;
}
#intro h2 {
…
margin-top: -10px;
}
#intro p {
…
margin: 1em;
}
```

3. Задайте отступ величиной 1em по всем сторонам раздела **#products**.

```
#products {
…
margin: 1em;
}
```

4. Теперь добавьте пространство шириной 2.5em над заголовками видов продукции **h3**. К этому моменту, я уверена, вы сможете написать правило без моей помощи, но для подсказки ниже приведено новое определение, добавленное к элементам **h3** в разделе «products». Вы можете попробовать добавить различные промежутки и посмотреть, какое расстояние вам понравится больше.

```
#products h3 {
…
margin-top: 2.5em;
}
```

5. Наконец, мы отделим блок отзывов, добавив пространство шириной 1em над ним и 10% с левой и правой сторон. В этот раз попробуйте сообразить сами.
6. Сохраните изменения еще раз, после чего страница должна выглядеть, как показанная на рис. 14.21. Просмотрите документ в браузере. Это не самый красивый дизайн, особенно, если окно вашего браузера широкое. Однако, сделав его очень узким, вы обнаружите, что дизайн не так уж плох в качестве версии для маленького экрана в адаптивном веб-дизайне. (Считайте это началом работы, которую мы выполним в главе 18.) Финальный вариант таблицы стилей для этой страницы доступен в приложении А.

Рис. 14.21. *Главная страница интернет-магазина «Малышок» после добавления отступов, границ и полей*

Присвоение типов отображения

Поскольку мы говорим о блоках и модели верстки CSS, наступило время для введения свойства **display**. Вы уже должны быть знакомы с характером отображения блочных и встроенных элементов. Однако не все XML-языки присваивают поведения отображения по умолчанию (или *типы отображения*) элементам, которые они содержат. По этой причине свойство **display** было создано для того, чтобы позволить верстальщикам определять, как элементы должны себя вести в макетах.

display

Принимаемые значения: **inline** | **block** | **list-item** | **inline-block** | **table** | **inline-table** | **table-row-group** | **table-header-group** | **table-footer-group** | **table-row** | **table-column-group** | **table-column** | **table-cell** | **table-caption** | **none**

Следующие значения являются новыми в CSS3: **run-in**|**compact**|**ruby**|**ruby-base**|**ruby-text**|**ruby-base-container**|**ruby-text-container**

Значения по умолчанию: **inline**

Применение: все элементы

Наследование: да

Свойство **display** определяет тип блока элемента, который порождается элементом в макете. В добавление к знакомым типам отображения **inline** и **block** вы можете также заставить элементы отображаться как пункты списка или различные части таблицы. Как видно из списка значений, существует множество типов элемента, но только несколько из них используются в повседневной практике.

Вообще, консорциум Всемирной паутины не одобряет произвольное переназначение типов отображения для элементов HTML. Однако в определенных ситуациях это неопасно и даже стало обычным явлением. Например распространенной практикой является отобразить элементы **li** (которые обычно отображаются как блочные элементы) как встроенные, чтобы превратить список в горизонтальную навигационную панель. Вы также можете отобразить встроенный элемент **a** (якорь) как блочный для того, чтобы задать ему конкретную ширину и высоту.

```
ul.navigation li { display: inline; }

ul.navigation li a { display: block; }
```

Другое полезное значение для свойства отображения — **none**, которое полностью удаляет контент из нормального потока. В отличие от **visibility: hidden**, которое просто делает элемент невидимым, но сохраняет пространство, которое бы он занимал, **display: none** удаляет и контент, и пространство.

Одним из распространенных использований **display: none** является предотвращение появления того или иного контента на определенных типах устройств, скажем, при печати страницы или отображении ее на

ПРЕДУПРЕЖДЕНИЕ

Имейте в виду, что изменение представления HTML-элемента при помощи CSS свойства **display** *не* меняет определение этого элемента как блочного или встроенного. Размещение блочного элемента внутри встроенного всегда будет ошибочным, независимо от типа его отображения.

ПРЕДУПРЕЖДЕНИЕ

Имейте в виду, что контент, для которого установлено значение **none** свойства **display**, по-прежнему загружается вместе с документом. Задание части контента свойства **display: none** для устройств с маленьким размером экрана сокращает длину страницы, но не влияет на уменьшение объема используемых данных или времени загрузки.

устройствах с маленькими экранами. Например, у вас можете быть абзац, который появляется, когда документ выводится на печать, но не является частью страницы, когда она отображается на экранах компьютеров.

Добавление теней к блокам

Мы добрались до последней станции в нашем путешествии по блокам элемента. В главе 12 вы узнали о свойстве **text-shadow**, которое добавляет к тексту тень. Свойство **box-shadow** (Новое в CSS3) применяет тень вокруг всего видимого блока элемента (без учета полей).

box-shadow

Новое в CSS3

Принимаемые значения: *'смещение по горизонтали' 'смещение по вертикали' 'расстояние размытия' 'расстояние распространения' цвет* **inset** | **none**

Значения по умолчанию: **none**

Применение: все элементы

Наследование: нет

Свойство **box-shadow** может показаться вам знакомым после работы со свойством **text-shadow**: укажите размер смещения по горизонтали и вертикали, насколько тень должна быть размытой и ее цвет. Для тени блока вы также можете указать значение *расстояния распространения*, которое увеличивает (или уменьшает, если значения отрицательные) размер тени. По умолчанию, цвет тени такой же, как цвет переднего плана элемента, но указание цвета это отменяет.

На рис. 14.22 показаны результаты следующих примеров кода. Первый (**А**) добавляет простую тень блока, смещенную на шесть пикселов вправо и вниз, без размытия или распространения. Второй (**Б**) добавляет значение размытия равное 5 пикселам, а третий (**В**) показывает эффект распространения на 10 пикселов. Тени элементов всегда применяются к области *за пределами* границ элемента (или позиции, где была бы граница, если она не указана). Если элемент имеет прозрачный или полупрозрачный фон, вы не увидите тень блока в области за элементом.

```
А.    -webkit-box-shadow: 6px 6px #666;
-moz-box-shadow: 6px 6px #666;
box-shadow: 6px 6px #666;
Б.    -webkit-box-shadow: 6px 6px 5px #666;
-moz-box-shadow: 6px 6px 5px #666;
box-shadow: 6px 6px 5px #666;/* размытие 5 пикселов */
В.    -webkit-box-shadow: 6px 6px 5px 10px #666;
-moz-box-shadow: 6px 6px 5px 10px #666;
box-shadow: 6px 6px 5px 10px #666;/* размытие 5px, распространение 10px */
```

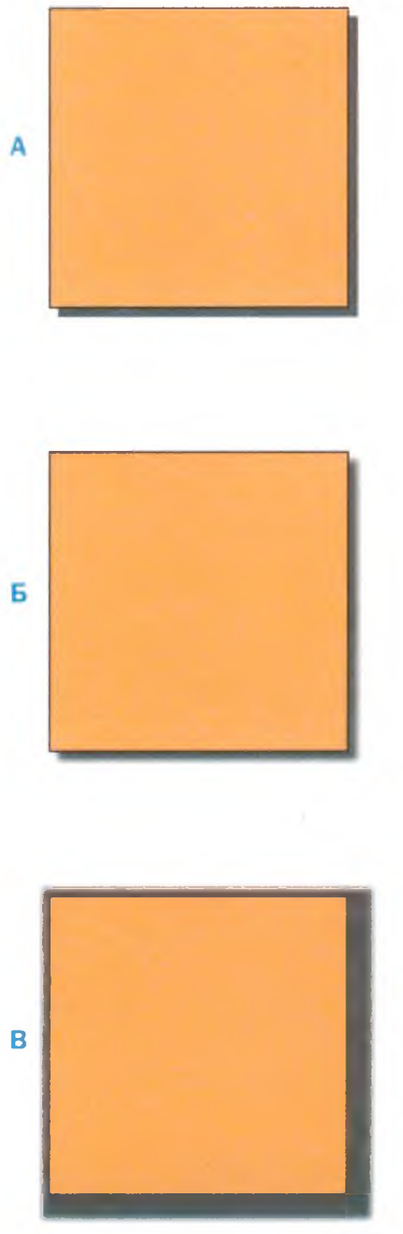


Рис. 14.22. Добавление тени вокруг элемента с помощью свойства ***box-shadow***

Рис. 14.23. Вдавленная тень элемента с помощью слова ***inset***

Вы можете визуализировать тень внутри границ видимого блока элемента, добавив в правило зарезервированное слово **inset**. Это делает его похожим на элемент, вдавленный в экран (рис. 14.23).

```
-webkit-box-shadow: inset 6px 6px 5px #666;
-moz-box-shadow: inset 6px 6px 5px #666;
box-shadow: inset 6px 6px 5px #666;
```

Что касается свойства **text-shadow**, можете указать несколько теней в блоке элемента, перечислив значения списком через запятую. Те, которые стоят на первом месте размещаются сверху, а последующие тени — ниже в том порядке, в котором они отображаются в списке.

Свойство **box-shadow** поддерживается всеми текущими версиями браузеров за исключением Opera Mini для мобильных устройств.

Резюме

Свойство	Описание
border	Сокращенное свойство, которое объединяет свойства границы
border-top, **border-right**, **border-bottom**, **border-left**	Объединяют свойства границы для каждой стороны элемента
border-color	Сокращенное свойство для задания цвета границ
border-top-color, **border-right-color**, **border-bottom-color**, **border-left-color**	Задают цвет границы для каждой стороны элемента
border-image CSS3	Добавляет изображение внутрь области границы
border-radius CSS3	Сокращенное свойство для скругления углов видимого блока элемента
border-top-left-radius, **border-top-right-radius**, **border-bottom-right-radius**, **border-bottom-left-radius**	Указывает радиус кривой для каждого угла в отдельности
border-style	Сокращенное свойство для задания стилей границ

Свойство	Описание
`border-top-style`, `border-right-style`, `border-bottom-style`, `border-left-style`	Задают стиль границы для каждой стороны элемента
`border-width`	Сокращенное свойство для задания ширины границ
`border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`	Задают ширину границы для каждой стороны элемента
`box-sizing`	Указывает, применяются ли размеры ширины и высоты к области контента или к области границы
`box-shadow` CSS3	Добавляет тень вокруг видимого блока элемента
`display`	Задает тип блока элемента, который порождается элементом
`height`	Задает высоту контентной области элемента
`margin`	Сокращенное свойство для задания области поля вокруг элемента
`margin-top`, `margin-right`, `margin-bottom`, `margin-left`	Задают величину поля для каждой стороны элемента
`max-height`	Задает максимальную высоту элемента
`max-width`	Задает максимальную ширину элемента
`min-height`	Задает минимальную высоту элемента
`min-width`	Задает минимальную ширину элемента
`overflow`	Позволяет управлять контентом, который не помещается в контентную область
`padding`	Сокращенное свойство для задания пространства между областью контента и границей
`padding-top`, `padding-right`, `padding-bottom`, `padding-left`	Задают величину отступа для каждой стороны элемента
`width`	Задает ширину контентной области элемента

ГЛАВА 15

ОБТЕКАНИЕ И ПОЗИЦИОНИРОВАНИЕ

На данный момент вы уже знаете множество свойств CSS, которые позволяют менять внешний вид текстовых элементов и сгенерированных ими блоков. Но до сих пор мы только декорировали элементы там, где они появлялись в потоке контента.

В этой главе мы рассмотрим обтекание и позиционирование, методы CSS для вывода потока и расстановки элементов на странице. При *обтекании* элемент смещается влево или вправо, а контент располагается с оставшихся сторон. *Позиционирование* — это способ задания размещения элемента в любом месте страницы с точностью до пиксела.

Мы начнем с изучения свойств, отвечающих за обтекание и позиционирование, так что вы получите хорошее представление о том, как работают инструменты верстки CSS-макетов. В главе 16 мы расширим границы и увидим, как эти свойства используются для создания распространенных многоколоночных макетов страниц.

Перед тем как мы начнем перемещать элементы, давайте убедимся, что мы хорошо знакомы с тем, как они себя ведут в нормальном потоке.

В этой главе

- Обтекание элементов слева и справа
- Запрет обтекания
- Заключение обтекаемых элементов
- Относительное позиционирование
- Абсолютное позиционирование и блоки
- Фиксированное позиционирование

Нормальный поток

Мы рассматривали нормальный поток в предыдущих главах, но стоит напомнить: в модели верстки CSS-макетов текстовые элементы расставляются сверху вниз в том порядке, в котором они появляются в исходном коде, и слева направо (в языках, в письменных системах которых слова читаются слева направо*). Блочные элементы выкладываются сверху друг за другом и заполняют доступную ширину окна браузера или другого содержащего элемента. Встроенные элементы и текстовые символы выстраиваются в линию рядом друг с другом, чтобы заполнить блочные элементы.

Когда переопределяются размеры окна или содержащего элемента, блочные элементы расширяются или сжимаются до новой ширины,

* В языках, в письменных системах которых слова читаются справа налево, таких как арабский язык и иврит, нормальный поток — сверху вниз и справа налево.

Решение ошибок браузеров

Сейчас благоприятное время обратиться к печальной теме ошибок браузеров. В этой книге демонстрируется предполагаемая работа CSS, но в реальности браузеры имеют ошибки и неустойчивую поддержку стандарта CSS, что превращает получение должного поведения макета в большую головную боль.

В прошлом основным виновником был браузер Internet Explorer 6. Эти ошибки были исправлены в более новых версиях браузера Internet Explorer и больше не являются проблемой.

Тем не менее теперешнее положение, возможно, хуже. Сейчас не только множество браузеров работает на всех видах устройств, но и ошибки, становятся более необычными и менее предсказуемыми.

Я, конечно, укажу на свойства, которые известны тем, что вызывают сбои в поведении браузера. К тому времени, как вы будете это читать, проблемные браузеры, вероятно, уже исчезнут. Лучший совет, который я могу дать вам — протестировать свой сайт в как можно большем количестве браузеров и устройств и исправить все, что работает неправильно. Поиск во Всемирной паутине по запросу, содержащему имена отдельных свойств или названия браузеров и слово «ошибка» (или «bug») обычно выявляет сообщения разработчиков, сталкивающихся с теми же проблемами или предлагающих потенциальные обходные пути. Вы также можете проверить сайт **cssdiscuss.incutio.com**, на котором размещен архив известных ошибок для всех браузеров, а также множество другой полезной информации о CSS.

а встроенный контент повторно заливается, чтобы подходить по размеру (рис. 15.1).

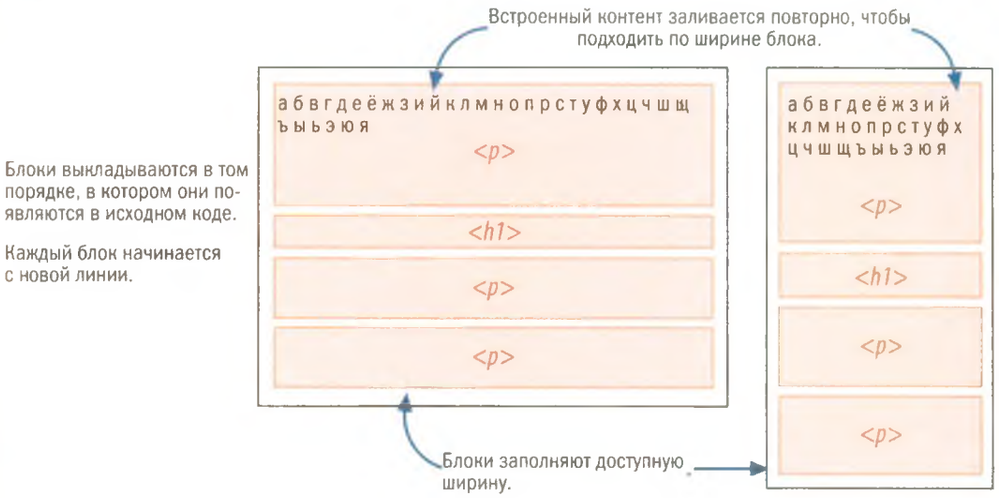


Рис. 15.1. Пример поведения нормального потока

Объекты в нормальном потоке влияют на расположение вокруг них. Это то поведение, к которому вы привыкли — на веб-страницах элементы не накладываются один на другой и не сбиваются в кучу, они создают «пространство» друг для друга.

Мы все это видели ранее, но в этой главе мы обратим внимание на то, находится ли элемент в потоке или удален из него. Обтекание и позиционирование меняет взаимоотношение элементов в нормальном потоке различными способами. Давайте сначала посмотрим на особое поведение обтекаемых (или *плавающих*) элементов.

Обтекание

Проще говоря, свойство **float** передвигает элемент максимально возможно влево или вправо, позволяя расположенному ниже контенту обтекать его. Свойство является не схемой позиционирования *как таковой*, а уникальной особенностью с некоторым интересным поведением, встроенной в CSS,. Плавающие элементы являются одним из главных инструментов современного веб-дизайна средствами CSS, использующихся для создания многоколоночных макетов, навигационных панелей на основе списков и выравнивания как с таблицами, но без таблиц. Это захватывающая вещь. Давайте начнем с самого свойства **float**.

float

Принимаемые значения: **left** | **right** | **none** | **inherit**

Значение по умолчанию: **none**

Применение: ко всем элементам

Наследование: нет

Лучше всего объяснит обтекание его демонстрация. В этом примере свойство **float** применяется к элементу **img** для перемещения его вправо. На рис. 15.2 показано, как абзац и содержащееся изображение отображаются по умолчанию (вверху) и как это выглядит, когда применено свойство **float** (внизу).

Разметка

```
<p><img src="images/lolita.png" alt="">Экспонат номер два -
записная книжечка в черном переплете...</p>
```

Таблица стилей

```
img {
float: right;
}
p {
padding: 15px;
background-color: #FFF799;
border: 2px solid #6C4788;
}
```

Рис. 15.2. Расположение изображения в нормальном потоке (вверху) и с примененным свойством **float** *(внизу)*

Отличный результат — мы избавились от большого количества бесполезного пространства на странице, но теперь текст наезжает на изображение. Как добавить немного пространства между изображением и окружающим текстом? Если вы предположили «добавить поле», вы абсолютно правы. Я добавлю 10 пикселов пространства со всех сторон изображения, используя свойство **margin** (рис. 15.3). Вы увидите, как все свойства блока работают вместе в макете страницы.

```
img {
float: right;
margin: 10px;
}
```

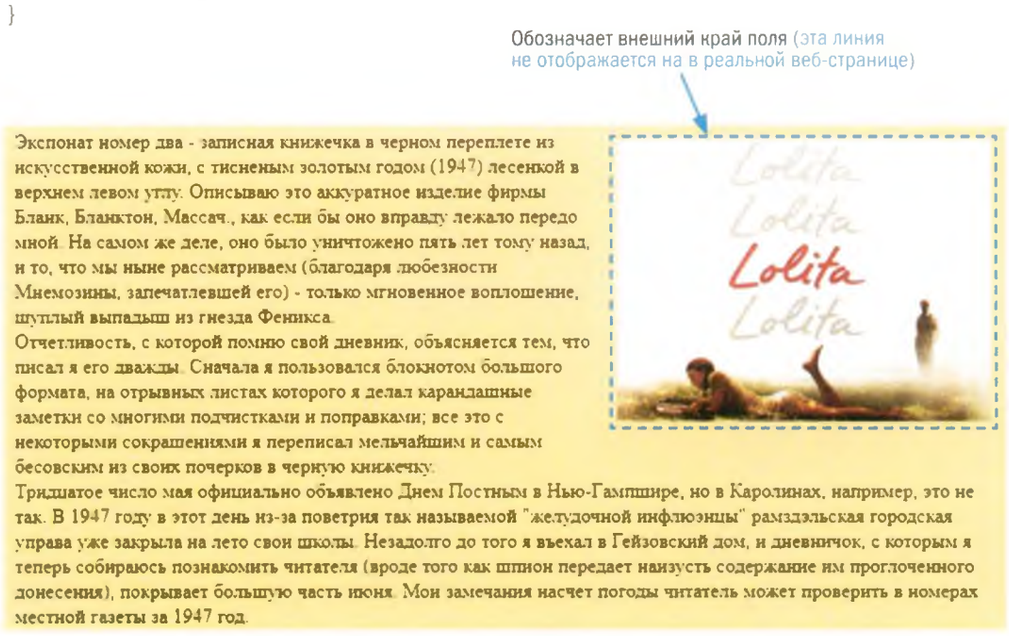


Рис. 15.3. Добавление 10-пиксельного поля вокруг обтекаемого элемента

Некоторые ключевые моменты поведения обтекаемых элементов, наблюдаемые на предыдущих двух рисунках:

Обтекаемый элемент подобен острову в потоке.

Первое и самое главное, вы можете видеть, что изображение, в обоих случаях перемещенное из своего положения в нормальном потоке, все еще продолжает влиять на окружающий контент. Последующий текст абзаца переформатируется, чтобы создать «комнату» для обтекаемого элемента **img**. Одна популярная аналогия сравнивает плавающие элементы с островами в потоке — они вне его, но он должен их обтекать. Это поведение уникально для обтекаемых элементов.

Плавающие элементы остаются в контентной области содержащего элемента.

Также важно отметить, что обтекаемое изображение размещено внутри области контента (внутренних краев) абзаца, который их содержит. Оно не продолжается в области отступов абзаца.

Поля сохраняются.

Кроме того, поле сохраняется со всех сторон обтекаемого изображения, как показано на рис. 15.3, штриховой линией темного цвета. Другими словами, весь блок элемента между внешними границами обтекается.

Обтекание встроенных и блочных элементов

Рассмотрим большее количество примеров и исследуем некоторые дополнительные варианты поведения обтекания. До изучения таблиц стилей, единственным, что вы могли перемещать, было изображение с использованием нерекомендуемого атрибута `align`. С CSS возможно перемещать любой HTML-элемент, как встроенный, так и блочный, как мы увидим в следующих примерах.

Обтекание встроенных текстовых элементов

В предыдущем примере, мы перемещали встроенный элемент изображения. На этот раз рассмотрим, что происходит, когда вы перемещаете встроенный текстовый (не замещаемый) элемент (рис. 15.4).

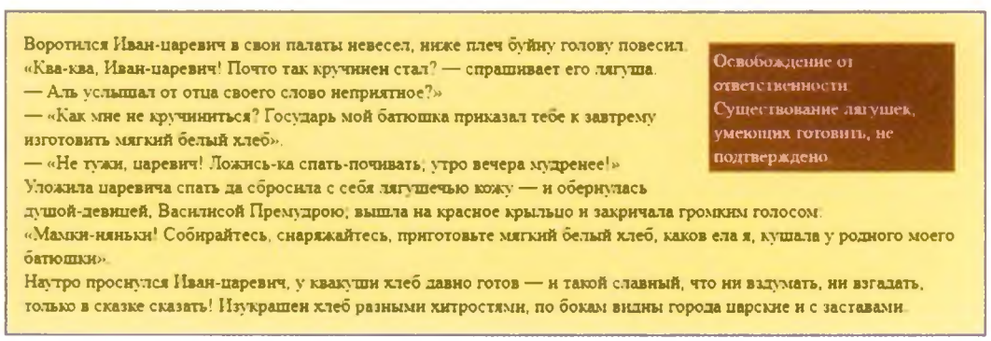


Рис. 15.4. Обтекание встроенного текстового (не замещаемого) элемента

Разметка

```
<p><span class="disclaimer">Освобождение от ответственности:
Существование лягушек, умеющих готовить, не подтверждено.
</span>Воротился Иван-царевич в свои палаты невесел... </p>
```

Таблица стилей

```
span.disclaimer {
float: right;
margin: 10px;
width: 200px;
color: #FFF;
background-color: #9D080D;
padding: 4px;
```

```
}
p {
padding: 15px;
background-color: #FFF799;
border: 2px solid #6C4788;
}
```

Необходимо задавать ширину для обтекаемых текстовых элементов.

На первый взгляд, он ведет себя точно так же, как обтекаемое изображение, что мы и ожидали. Но здесь есть некоторые небольшие различия в работе, на которые следует обратить внимание.

Всегда задавайте ширину для обтекаемых текстовых элементов.

Для начала вы заметите, что правило стилей, которое перемещает **span**, включает свойство **width**. На самом деле, необходимо задавать ширину для обтекаемых текстовых элементов, потому что без этого область контента блока расширится до его максимально возможной ширины (или, в некоторых браузерах, она может сжаться до ее минимально возможной ширины). Изображения имеют собственную ширину, так что нам не обязательно задавать ее в предыдущем примере (хотя мы, несомненно, могли бы).

Обтекаемые встроенные элементы ведут себя как блочные элементы.

Обратите внимание, что поле сохраняется со всех сторон обтекаемого текста **span**, даже несмотря на то, что верхнее и нижнее поля для встроенных элементов обычно не отображаются (см. рис. 14.18 в предыдущей главе). Это из-за того, что все обтекаемые элементы ведут себя как блочные. Задав обтекание встроенному элементу, вы заставляете его следовать правилам отображения для блочных элементов, и поля показываются со всех четырех сторон.

Поля обтекаемых элементов *не* сжимаются.

В нормальном потоке, накладывающиеся верхнее и нижнее поля сокращаются (перекрываются), но для обтекаемых элементов поля поддерживаются по всем сторонам, как указано.

Обтекание блочных элементов

Рассмотрим, что происходит, когда вы перемещаете блок внутри нормального потока. В этом примере целый элемент абзаца перемещается влево (рис. 15.5).

Разметка

```
<p>Воротился Иван-царевич в свои палаты невесел...</p>

<p id="float"> "Ква-ква, Иван-царевич! Почто так кручинен стал?...</p>

<p> Уложила царевича спать да сбросила с себя лягушечью кожу...</p>
```

Таблица стилей

```
p#float {
float: left;
width: 200px;
margin-top: 0px;
background: #A5D3DE;
}
p {
border: 1px solid red;
}
```

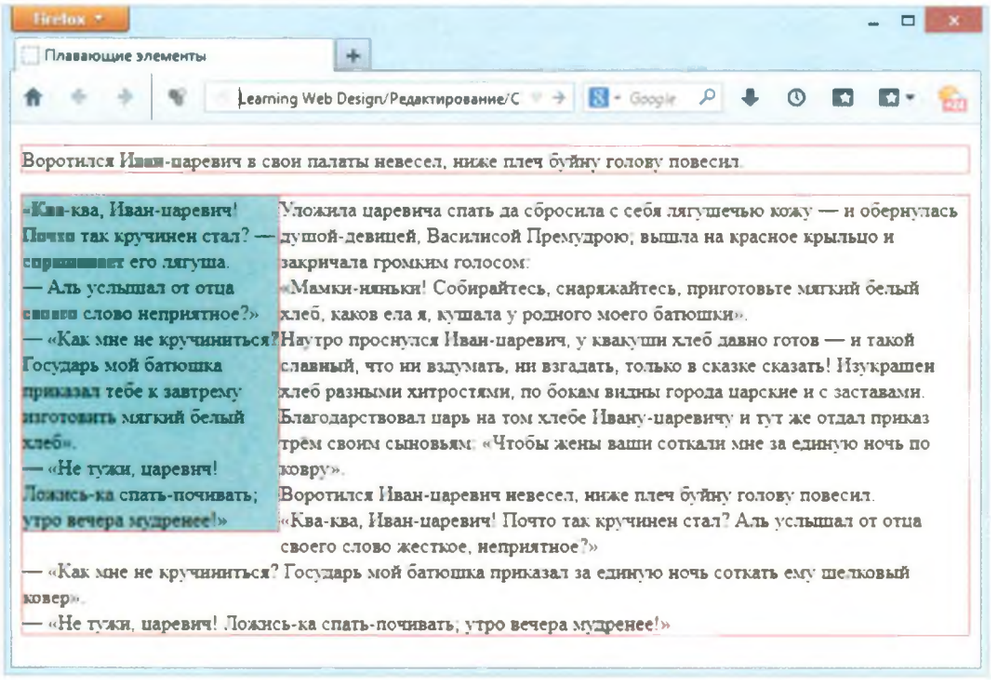


Рис. 15.5. Обтекание блочного элемента

Я добавила красную линию вокруг всех элементов **p**, чтобы показать их пределы. Более того, я установила значение верхнего поля в плавающем элементе равным 0 (нулю), чтобы заменить в браузерах настройки поля по умолчанию у абзацев. Это позволяет обтекаемому абзацу выравниваться с верхней стороной нижерасположенного абзаца. Есть еще несколько других моментов, на которые бы я хотела обратить внимание в этом примере.

Точно как мы видели в примере с изображением, абзац отодвигается в сторону (на этот раз влево), и нижерасположенный контент обтекает его, даже несмотря на то, что блоки обычно складываются сверху друг за другом. Существует два момента, на которые я хочу обратить внимание в этом примере.

ПРИМЕЧАНИЕ

Абсолютное позиционирование в CSS — метод размещения элементов на странице независимо от того, как они указаны в исходном коде. Мы приступим к абсолютному позиционированию через несколько разделов.

Вы должны задавать ширину для обтекаемых блочных элементов.

Если вы не зададите значение **width**, ширина обтекаемого блока будет установлена как **auto**, которое заполняет доступную ширину окна браузера или другого содержащего элемента. Нет больше смысла использовать обтекаемый блок полной ширины, так как лучше разместить текст рядом с плавающим элементом, а не располагать его ниже.

Элементы не перемещаются выше своих источников в исходном коде.

Обтекаемый блок будет перемещен влево или вправо относительно того, где он помещен в исходном коде, позволяя расположенным ниже элементам в потоке обступать его. Он останется ниже любых блочных элементов, которые предшествовали ему в потоке (в действительности «заблокирован» ими). Это означает, что вы не можете перемещать элемент вверх, в верхний угол страницы, даже если его ближайшим «предком» является элемент **body**. Если вы хотите, чтобы обтекаемый элемент начинался вверху страницы, он должен быть указан первым в исходном коде документа.

Запрет обтекания

Если вы планируете использовать обтекание элементов, важно знать как *отключить* обтекание текстом и вернуться к обычной верстке. Это делается при помощи *запрета обтекания* элементов, которые располагаются ниже плавающего. Применение свойства **clear** к элементу предотвращает его появление рядом с обтекаемым, и заставляет его расположиться в ближайшем доступном «пустом» пространстве под плавающим элементом.

clear

Принимаемые значения: **left** | **right** | **both** | **none** | **inherit**

Значение по умолчанию: **none**

Применение: только к блочным элементам

Наследование: нет

Имейте в виду, что вы применяете свойство **clear** не к самому обтекаемому элементу, а к тому, который хотите разместить *ниже* него. Значение **left** запрещает обтекать все плавающие элементы, которые были перемещены влево. Аналогично, значение **right** делает то же для элементов, находящихся с правого края содержащего блока. Если есть несколько обтекаемых элементов и вы хотите удостовериться, что элемент располагается ниже их всех, используйте значение **both** для запрета обтекания с обеих сторон.

В этом примере свойство **clear** было использовано, чтобы заставить элементы **h2** расположиться ниже перемещенных влево обтекаемых элементов. На рис. 15.6 показано, как заголовок **h2** располагается у ближайшего доступного края ниже плавающего элемента.

```
img {
float: left;
margin-right: 10px;
}
h2 {
clear: left;
margin-top: 2em;
}
```

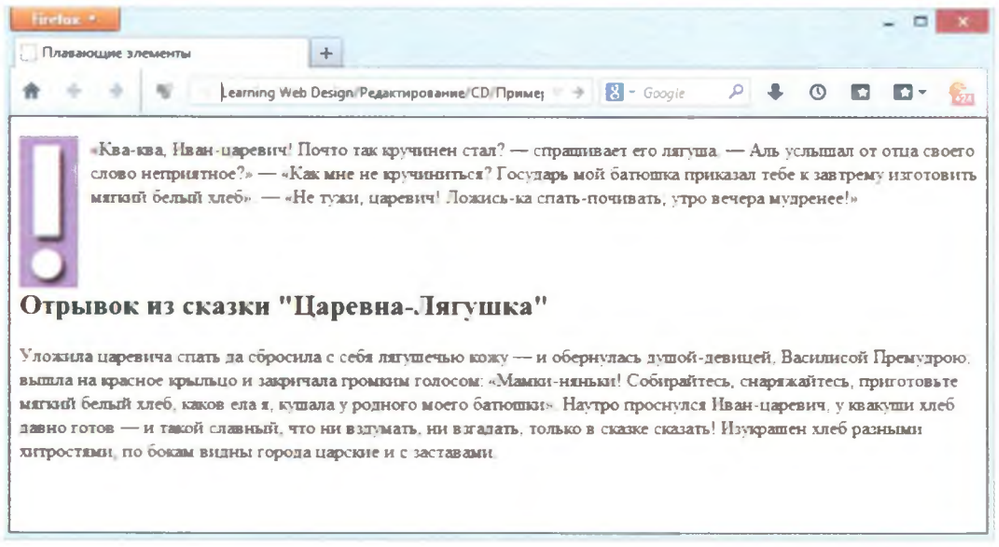


Рис. 15.6. Запрет обтекания перемещенного влево плавающего элемента

Обратите внимание (см. рис. 15.6), что несмотря на то, что к элементу **h2** было применено верхнее поле в 2em, оно не отображается между заголовком и обтекаемым изображением. Это следствие совмещающихся вертикальных полей. Если вы хотите обеспечить сохраняемое пространство между плавающим элементом и нижерасположенным текстом, примените нижнее поле к самому обтекаемому элементу.

Настала пора испытать обтекание в упражнении 15.1.

УПРАЖНЕНИЕ 15.1. ОБТЕКАНИЕ ЭЛЕМЕНТОВ

В упражнениях этой главы мы продолжим улучшать главную страницу интернет-магазина «Малышок», над которой работали в главе 14. Если вы не выполняли упражнения в предыдущей главе или просто хотите начать все заново, в папке *Глава 15* на диске, прилагающемся к книге находится последний вариант файла *malishok_ch15.html*.

1. Откройте файл главной страницы интернет-магазина «Малышок» в текстовом редакторе и браузере (она должна выглядеть так, как показано на рис. 14.21 в предыдущей главе).

Мы начнем с удаления бесполезного вертикального пространства рядом с изображениями продукции, переместив изображения влево. Используем контекстуальный селектор, чтобы обеспечить перемещение изображения только в разделе «products» на странице. Пока мы работаем над этим разделом, давайте добавим небольшое поле на нижнюю и правую стороны, используя сокращенное свойство **margin**.

```
#products img {
float: left;
margin: 0 6px 6px 0;
}
```

Сохраните документ и взгляните на него в браузере. Вы должны видеть описания продукции, обтекающие изображения справа.

2. Далее мне бы хотелось, чтобы ссылки «Просмотреть ассортимент...» всегда появлялись ниже изображений с тем, чтобы они были ясно видны с левой стороны раздела продукции. Это изменение потребует небольшой дополнительной разметки, потому что нам нужен способ выбора только тех абзацев, которые содержат ссылки «more». Добавьте имя класса «more» каждому из абзацев, которые содержат ссылки. Ниже приведен первый:

```
<p class="more"><a href="#">Просмотреть ассортимент игрушек...</a></p>
```

Теперь мы можем использовать селектор класса, чтобы запретить тем абзацам обтекать плавающие изображения.

```
#products p.more {
clear: left;
}
```

На рис. 15.7 показан новый и улучшенный раздел «products».

Рис. 15.7. Плавающие изображения и заключенный в контейнер текст экономят пространство раздела товаров

Обтекание нескольких элементов

Это прекрасно — перемещать несколько элементов на странице или даже внутри одного элемента. Фактически, это один из способов превратить список ссылок в горизонтальное меню, как вы увидите далее. Когда вы перемещаете несколько элементов, используется сложная система, незаметно осуществляющая отображение правил, которая обеспечивает отсутствие наложения обтекаемых элементов. Вы можете обратиться за деталями к спецификации CSS, но мораль в том, что обтекаемые элементы будут размещены так далеко влево или вправо (как задано) и так высоко, как позволит пространство.

На рис. 15.8 показано, что происходит, когда ряд последовательных абзацев помещается к одной и той же стороне. Первые три плавающих элемента начинают укладываться от левого края, но когда нет достаточного места для четвертого, он перемещается вниз и влево, пока не столкнется с чем-нибудь; в данном случае — с краем окна браузера. Однако если бы один из плавающих элементов, например «P2», был слишком длинным, он вместо этого наскочил бы на край длинного плавающего элемента.

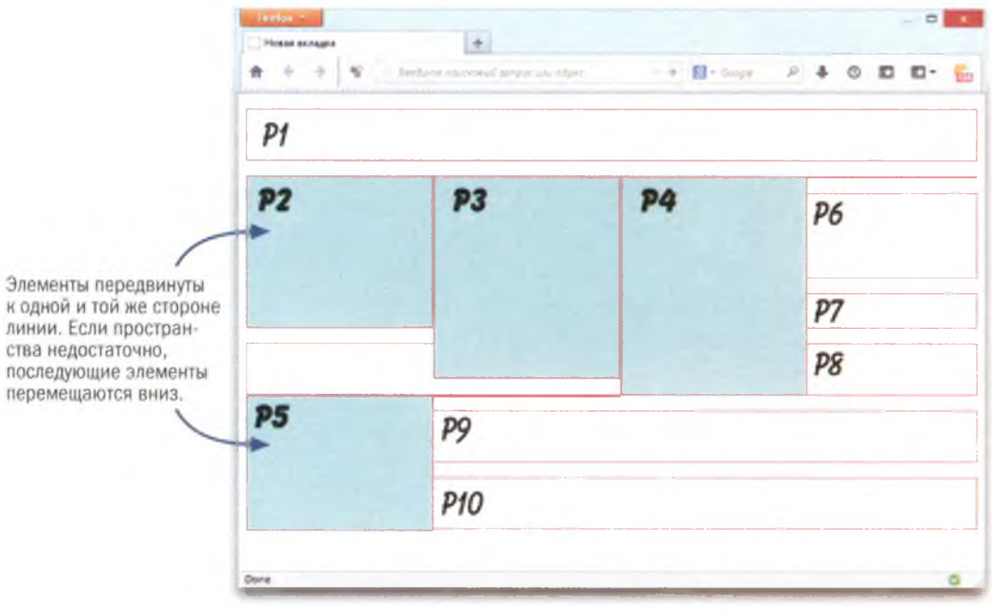


Рис. 15.8. *Несколько обтекаемых элементов выстраиваются в линию и не перекрывают друг друга*

Разметка

```
<p>P1</p>
<p class="float">P2</p>
<p class="float">P3</p>
<p class="float">P4</p>
<p class="float">P5</p>
<p>P6</p>
<p>P7</p>
<p>P8</p>
```

```
<p>P9</p>
<p>P10</p>
```

Таблица стилей

```
p#float {
float: left;
width: 200px;
margin: 0px;
background: #CCC;
}
p {border: 1px solid red;
}
```

Это основное поведение, но давайте применим его к чему-нибудь более практическому, например к меню навигации. Имеет смысл семантически размечать навигацию в виде неупорядоченного списка, как показано здесь. Я опустила URL-адреса в элементах для упрощения примера разметки.

```
<ul>
<li><a href="#">Serif</a></li>
<li><a href="#">Sans-serif</a></li>
<li><a href="#">Script</a></li>
<li><a href="#">Display</a></li>
<li><a href="#">Dingbats</a></li>
</ul>
```

Существуют различные подходы к преобразованию его в горизонтальную панель (см. примечание), но основные шаги в нашем примере с обтеканием следующие.

1. Отключите маркеры списка и установите значения отступов и полей равные нулю.

```
ul {
list-style-type: none;
margin: 0;
padding: 0;
}
```

2. Установите обтекание каждого элемента списка слева так, чтобы они выстроились в линию, благодаря описанному выше поведению при обтекании нескольких элементов.

```
ul li {
float: left;
}
```

3. Отобразите якорные элементы пунктов списка (`a`) как блочные, чтобы можно было задать размеры, отступы, поля и другие визуальные стили. Также вы можете задать стили для других состояний ссылки (таких, как `a:hover`), но мой пример будет коротким.

```
ul li a {
display: block;
/* еще стили */
}
```

4. Запретите обтекание элемента, расположенного в документе после меню, чтобы он размещался ниже меню.

По крайней мере, вам нужно будет придать небольшой отступ и/или поля якорным элементам, чтобы предоставить ссылкам чуть больше места, но вы можете добавить любой из уже рассмотренных нами стилей — цвета, границы, скругленные углы, фоновые изображения, чтобы меню навигации оказалось желаемого вида. Следующие стили превращают приведенный выше пример списка в показанное на рис. 15.9 меню с элементами, похожими на вкладки.

Рис. 15.9. Неупорядоченный список превращается в меню с элементами, похожими на вкладки с помощью CSS и без использования изображений

Заключение плавающих элементов в контейнер

Пока мы говорим об обтекании нескольких элементов, самое время рассмотреть еще одну хитрость обтекания —заключение плавающих элементов в контейнер. По умолчанию обтекаемые элементы спроектированы так, что они выходят за пределы своих элементов-контейнеров. Это просто прекрасно для того, чтобы позволить тексту обтекать изображение, но иногда такое поведение может привести к нежелательному результату.

Возьмем пример, показанный на рис. 15.10. Очевидно, было бы лучше, если бы граница растягивалась так, чтобы вместить весь контент, но плавающее изображение выступает за нижний край.

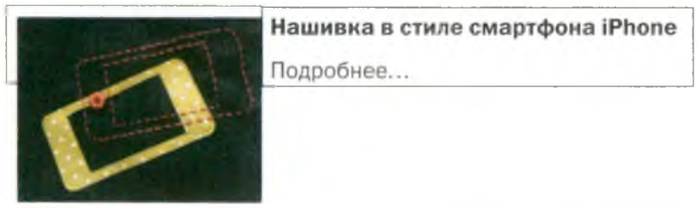


Рис. 15.10. Элемент-контейнер не растягивается до размеров целого изображения

И если вы задаете обтекание для всех элементов в контейнере, как можно было бы сделать для создания многоколоночного макета, в потоке не останется элементов, чтобы удерживать контейнер открытым. Это явление проиллюстрировано на рис. 15.11. Элемент `#container div`

ПРИМЕЧАНИЕ

Другой способ выровнять элементы списка в строку — это отобразить их не как блочные, а как встроенные (`li{display: inline;}`). Отсюда вы сможете задать отображение якорных элементов в виде блоков и применить стили. Этот метод, однако, затруднит точное управление расстоянием между элементами навигации, так как браузер задает размер пробелов между элементами списка в исходном коде, согласно свойству `font-size` контейнера.

включает в себя два абзаца. Вид нормального потока (слева) показывает, что у элемента **#container** есть фоновый цвет и граница, которые являются оболочкой для контента. Однако когда оба абзаца задаются плавающими, блок элемента **#container** закрывается, его высота становится равной нулю, а плавающие элементы выступают за его пределы снизу (вверху все еще видна пустая граница). Ясно, что это не тот эффект, который нам нужен.

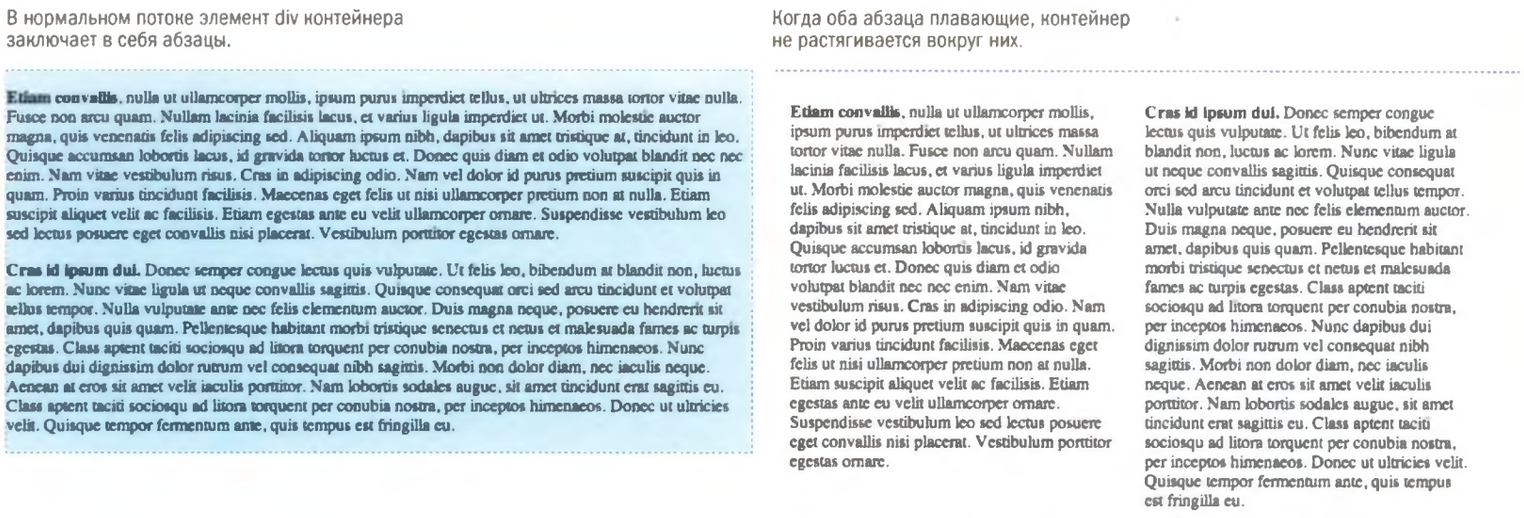


Рис. 15.11. Блок контейнера не исчезает полностью, когда все его содержимое становится плавающим

К счастью, есть несколько способов исправить эту проблему, и они довольно просты. Одним из вариантов является сделать плавающим сам контейнер и задать ему ширину 100%.

```
#container {
float: left;
width: 100%;
background-color: #GGG;
padding: 1em;
}
```

Другое распространенное решение — воспользоваться поведением свойства **overflow**. Если задать для элемента-контейнера значение свойства **auto** или **hidden**, это также заставит его растянуться, чтобы заключить в себя плавающие элементы. Кроме того, я добавила точное значение ширины, чтобы избежать ошибок в старых версиях браузера Internet Explorer, но имейте в виду, что если у вашего элемента контейнера есть граница, ширина 100% заставит границу выйти за пределы окна браузера.

```
#container {
overflow: auto;
width: 100%;
background-color: #GGG;
padding: 1em;
}
```

На рис. 15.12 показан результат применения к предыдущим примерам техники заключения плавающих элементов в контейнер. Один из способов позволит добиться результата.

Теперь самое время украсить раздел навигации на странице интернет-магазина «Малышок» в упражнении 15.2.

Нашивка в стиле смартфона iPhone

Подробнее...

Etiam convallis, nulla ut ullamcorper mollis, ipsum purus imperdiet tellus, ut ultrices massa tortor vitae nulla. Fusce non arcu quam. Nullam lacinia facilisis lacus, et varius ligula imperdiet ut. Morbi molestie auctor magna, quis venenatis felis adipiscing sed. Aliquam ipsum nibh, dapibus sit amet tristique at, tincidunt in leo. Quisque accumsan lobortis lacus, id gravida tortor luctus et. Donec quis diam et odio volutpat blandit nec nec enim. Nam vitae vestibulum risus. Cras in adipiscing odio. Nam vel dolor id purus pretium suscipit quis in quam. Proin varius tincidunt facilisis. Maecenas eget felis ut nisi ullamcorper pretium non at nulla. Etiam suscipit aliquet velit ac facilisis. Etiam egestas ante eu velit ullamcorper ornare. Suspendisse vestibulum leo sed lectus posuere eget convallis nisi placerat. Vestibulum porttitor egestas ornare.

Cras id ipsum dui. Donec semper congue lectus quis vulputate. Ut felis leo, bibendum at blandit non, luctus ac lorem. Nunc vitae ligula ut neque convallis sagittis. Quisque consequat orci sed arcu tincidunt et volutpat tellus tempor. Nulla vulputate ante nec felis elementum auctor. Duis magna neque, posuere eu hendrerit sit amet, dapibus quis quam. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc dapibus dui dignissim dolor rutrum vel consequat nibh sagittis. Morbi non dolor diam, nec iaculis neque. Aenean at eros sit amet velit iaculis porttitor. Nam lobortis sodales augue, sit amet tincidunt erat sagittis eu. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Donec ut ultricies velit. Quisque tempor fermentum ante, quis tempus est fringilla eu.

Рис. 15.12. *Наши свисавшие плавающие элементы теперь заключены в контейнер*

УПРАЖНЕНИЕ 15.2 | СОЗДАНИЕ ПАНЕЛИ НАВИГАЦИИ

Откройте копию файла *malishok.html* (или *malishok_ch15.html*), если еще не сделали этого.

1. Сначала сделайте элементы `ul` как можно более нейтральными. Маркеры списка уже отключены, но нам необходимо убрать отступы и поля, которые могут там оказаться.

```
#nav ul {
list-style-type: none;
padding: 0;
margin: 0;
}
```

2. Далее установите обтекание элементов списка слева и запретите следующему далее элементу **div** продуктов появляться рядом с плавающими элементами.

```
#nav ul li {
…
float: left;
}
#products {
…
clear: both;
}
```

Сохраните документ и просмотрите его в браузере. Вы должны увидеть, что ссылки теперь размещены в строке довольно плотно, а фиолетовая панель навигации сократилась почти полностью — заключить обтекаемые элементы в контейнер не получилось! Давайте исправим это, применив метод со свойством **overflow**. И пока мы этим занимаемся, сделаем то же самое для элемента **#products div**, чтобы он обязательно вмещал в себя плавающие изображения.

```
#nav {
…
overflow: hidden;
width: 100%;
}
#products {
…
overflow: hidden;
}
```

3. Теперь можно поработать над внешним видом ссылок. Сначала заставьте все элементы **a** отображаться как блочные, а не встроенные. Вместо того, чтобы устанавливать конкретные размеры для каждой ссылки, мы применим отступ (.5em), чтобы предоставить им немного пространства внутри границы, и поля (.25em) для добавления пространства между ссылками. Я сделала границу лавандового цвета установленной по умолчанию, но осветлила ее до белого в состояниях **:focus** и **:hover**.

```
#nav ul li a {
display: block;
padding: .5em;
border: 1px solid #ba89a8;
border-radius: .5em;
margin: .25em;
}
#nav ul a:focus {
color:#fc6
border-color: #fff;
}
#nav ul a:hover {
color: #fc6;
border-color: #fff;
}
```

4. Наконец, отцентрируем список по ширине раздела **nav**. Это можно сделать, применив свойство **width** к элементу **ul** и задав значение боковых полей **auto**. Признаюсь, мне пришлось повозиться с несколькими величинами ширины, чтобы подобрать одну, которая подходит ко всему меню именно так, как нужно (31em). Если последнее слишком широкое, оно не будет правильно выровнено по центру.

```
#nav ul {
list-style: none;
padding: 0;
margin: 0 auto;
width: 31em;
}
```

На рис. 15.13 показано, как должна выглядеть ваша панель навигации при просмотре в браузере.

Рис. 15.13. Список ссылок теперь стилизован под горизонтальное меню

Использование плавающих элементов для создания колонок

Пока мы добавляли обтекание небольшим элементам на странице, но, как упоминалось ранее, можно сделать плавающими целые разделы страницы, создав колонки. На самом деле профессионалы так и поступают! Есть несколько решений, и их выбор, в основном, дело вкуса.

Для создания двух плавающих колонок можно выполнить следующее:

- Придать обтекание одному элементу **div** и добавить большое поле со стороны текстового элемента, обтекающего его.
- Добавить обоим элементам **div** обтекание слева или справа.
- Задать обтекание одного элемента **div** слева, а второго — справа (или наоборот).

Три плавающие колонки создаются примерно так же, только придется произвести больше расчетов.

Независимо от того, какой метод лучше всего подходит для вашего контента или вашей фантазии, нужно учитывать несколько вещей. В первую очередь, каждый плавающий элемент должен иметь определенную ширину. Далее, нужно быть точно уверенным, что вы верно рассчитали ширину каждой колонки, расставив отступы, границы и поля. Если суммарная ширина всех колонок превышает доступную ширину браузера или иного элемента-контейнера, у вас получится так называемое падение плавающих элементов. То есть, последняя плавающая колонка выйдет за отведенное пространство и будет сдвинута вниз под соседнюю с ней колонку.

Ограничение на использование плавающих элементов для создания колонок таково, что все зависит от порядка элементов в исходном коде. Плавающий элемент должен появиться *перед* обтекающим его контентом, и ваш исходный код не всегда может содержать элементы в удобном порядке. Теперь попробуйте создать двухколоночный макет с плавающими элементами в упражнении 15.3, применив технику «один плавающий элемент плюс поле», описанную выше.

ПРИМЕЧАНИЕ

Существуют способы освободиться от порядка элементов в исходном коде, используя отрицательные значения полей, как вы узнаете в главе 16.

УПРАЖНЕНИЕ 15.3 | СОЗДАНИЕ КОЛОНОК С ПЛАВАЮЩИМИ ЭЛЕМЕНТАМИ

Макет, который мы использовали для сайта «Малышок» может быть хорош для устройства с небольшим экраном, но он становится неуклюжим в больших окнах браузеров. В этом упражнении мы напишем стили, чтобы придать странице «жидкий» двухколоночный макет с помощью плавающих элементов. Я рекомендую скопировать текущую версию файла страницы магазина «Малышок» и переименовать его в *malishok-float.html*. Так вы сохраните копию без изменений для следующего упражнения, и вам не нужно будет отменять действия, которые вы совершите сейчас.

Мы собираемся задать ширину элементу **#products div**, сместить его влево и позволить блоку отзывов обтекать его справа, создавая вторую колонку. Я хочу, чтобы ширина этого макета менялась пропорционально ширине экрана и всегда заполняла его полностью, поэтому я буду использовать процентные значения (а значит, внесу несколько изменений в имеющийся код).

1. Начните с задания ширины элемента **#products div** равной 55% и смещения его влево.

Сейчас величина отступов и полей по всем сторонам установлена 1em, но для «жидкого» макета измените ширину левого и правого отступов и полей на 2%. Это означает, что блок товаров теперь занимает примерно 63% ширины экрана (2% + 2% + 55% + 2% + 2%), плюс еще несколько пикселов границы. На рис. 15.14 показаны результаты этих изменений.

Кроме того, задайте величину верхнего поля элемента **#products** равную нулю.

```
#products {
background-color: #FFF;
line-height: 1.5em;
padding: 1em 2%;
border: double #FFBC53;
margin: 0 2% 1em;
clear: both;
float: left;
width: 55%;
}
```

Здесь наблюдается несколько интересных моделей поведения. Текст отзывов переместился вправо от блока товаров, что и ожидалось, но блок отзывов (вместе с изображением восклицательного знака) скрыт за блоком товаров. Обтекает только контент, блок элемента перемещается вверх и не меняет своего размера.

2. Пора привести блок отзывов в форму. Нам нужно настроить поля особым образом, чтобы левое поле блока отзывов стало достаточно широким и открыло блок товаров. Блок товаров занимает чуть больше 63% от ширины страницы, так что давайте зададим значение левого поля блока отзывов 64%, чтобы приспособить его и добавить немного пространства между блоками. Я также установила узкое правое поле равное 2% (помните, что порядок указания значений в определении — верх, право, низ, лево). Обновите страницу, и блок отзывов должен оказаться по центру правой колонки.

```
#testimonials {
...
margin: 1em 10%;/* удалить */
margin: 1em 2% 1em 64%;
}
```

3. Осталось еще несколько настроек. Укажите положение сведений об авторских правах так, чтобы они появились в нижней части страницы. Наконец, я думаю, элемент **h2** «Ассортимент» будет выглядеть лучше, выровненным в макете по левому краю так, что давайте и это подправим.

```
p#copyright {
...
clear: left;
}
```

```
#products h2 {
...
text-align: center left;
}
```

Результаты представлены на рис. 15.15. Эй, полюбуйтесь! Ваш первый двухколоночный макет, созданный с использованием плавающего элемента и широкого поля. Это основная концепция многих шаблонов макетов на основе CSS, как вы увидите в главе 16.

Рис. 15.14. Результат обтекания элемента **div** *раздела товаров*

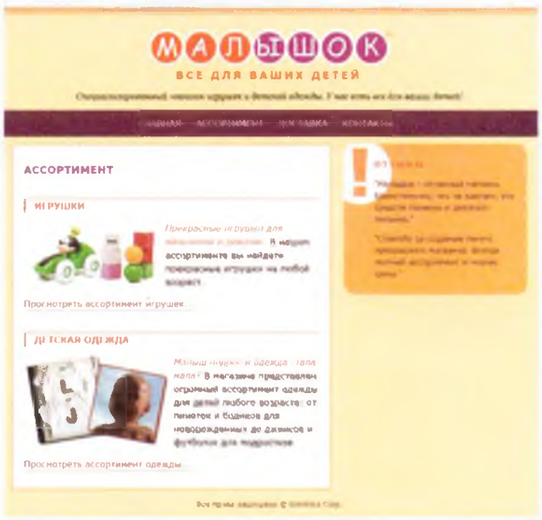


Рис. 15.15. Новый двухколоночный макет главной страницы сайта «Малышок», созданный с помощью плавающего элемента и широкого поля для последующего контента. Этот макет будет хорошо работать на планшетных устройствах или в окнах браузеров настольных компьютеров

Смешение процентов и единиц измерения em

В упражнении 15.2 мы указали поля как сочетание процентных значений и единиц измерения em. Фактически это распространено в современной веб-разработке, в частности, для создания «жидких» макетов, которые реагируют на размер области просмотра. Некоторые разработчики используют процентные значения для всех размеров по горизонтали, чтобы они менялись относительно размера окна просмотра, но указывают единицы em для всех измерений по вертикали потому, что это сохраняет масштаб и размер строк текста. Такой метод — дело вкуса, а не обязательное требование, но его стоит иметь в виду.

ПРЕДУПРЕЖДЕНИЕ

С осторожностью сочетайте «жидкие» макеты и границы. Как правило, лучше, если сумма ваших процентных значений меньше 100%, чтобы уместить ширину границ (если они используется) и чтобы приспособиться к ошибкам округления, иногда допускаемым браузерами. Если ширина колонок слишком часто округляется в бóльшую сторону, колонки могут быть рассчитаны браузером как слишком широкие, и вам грозит падение плавающего элемента.

На этом изучение основ обтекания завершено. Теперь мы перейдем к следующему способу перемещения элементов на странице — позиционированию.

Основы позиционирования

Каскадные таблицы стилей предоставляют несколько методов расположения элементов на странице. Они могут быть позиционированы относительно того, где они должны были появиться в потоке, или всецело удалены из него и помещены в конкретную позицию страницы. Вы можете также позиционировать элементы относительно окна браузера (формально известного как *окно просмотра* в рекомендациях CSS), и они останутся на экране, пока остальная часть страницы будет прокручиваться.

Типы позиционирования

`position`

Принимаемые значения: **`static`** | **`relative`** | **`absolute`** | **`fixed`** | **`inherit`**

Значение по умолчанию: **`static`**

Применение: ко всем элементам

Наследование: нет

Свойство **`position`** указывает, что элемент должен быть позиционирован и задает для этого метод, который следует использовать. Ниже я кратко представлю каждое зарезервированное значение, а затем мы более детально рассмотрим каждый метод.

`static`

Это обычная схема позиционирования, в которой элементы размещаются так, как они встречаются в нормальном потоке документа.

`relative`

Относительное позиционирование сдвигает блок относительно его начального положения в потоке. Характерным поведением относительного позиционирования является то, что пространство, которое элемент занимал бы в нормальном потоке, сохраняется.

absolute

Абсолютно позиционированные элементы полностью удаляются из потока документа и позиционируются относительно окна браузера или содержащего элемента (позже мы поговорим об этом подробнее). В отличие от относительно позиционированных элементов, пространство, которое они занимали бы, смыкается. На самом деле абсолютно позиционированные элементы никак не влияют на расположение окружающих.

fixed

Отличительной чертой *фиксированного позиционирования* является то, что элемент остается в одном положении в окне, даже если документ прокручивается. Фиксированные элементы удаляются из потока документа и позиционируются относительно окна браузера (или другого окна просмотра), а не другого элемента в документе. На данный момент оно вызывает некоторые сбои на мобильных устройствах, о чем будет говориться далее в этой главе.

У каждого метода позиционирования есть своя цель, но абсолютное позиционирование наиболее универсально. С абсолютным позиционированием вы можете разместить объект в любой позиции в окне просмотра или внутри другого элемента. Абсолютное позиционирование может даже применяться для создания макетов с несколькими колонками, но чаще используется для решения мелких задач, таких как позиционирование поля поиска в верхнем углу заголовка. Кроме того, его можно использовать, чтобы извлечь изображение или фрагмент из его контейнера, создавая эффект висячих отступов или наложения. Это удобный инструмент, если использовать его осторожно и в меру.

Задавание положения

Как только вы установили метод позиционирования, действительное положение задается с помощью четырех свойств *смещения*.

top, **right**, **bottom**, **left**

Принимаемые значения: *значение длины* | *проценты* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: к позиционированным элементам (где значением положения является **relative**, **absolute** или **fixed**)

Наследование: нет

Значения, заданные для каждого свойства смещения, определяют расстояние, на которое элемент должен быть сдвинут *от* соответствующего края. Например, значение **top** определяет расстояние, на которое

верхний внешний край позиционированного элемента должен быть смещен от верхнего края браузера или другого содержащего элемента. Положительное значение **top** сместит блок элемента *вниз* на эту величину. Аналогично, положительное значение **left** переместит позиционированный элемент вправо (к центру содержащего элемента) на эту величину.

Дальнейшие объяснения и примеры свойств смещения будут предоставляться при рассмотрении каждого метода позиционирования. Мы начнем с довольно простого метода **relative**.

ПРИМЕЧАНИЕ

Допустимы отрицательные значения, которые перемещают элемент в противоположном направлении. Например отрицательное значение **top** в результате переместит элемент вверх.

Относительное позиционирование

Как ранее упоминалось, относительное позиционирование перемещает элемент относительно его начальной позиции в потоке. Пространство, которое он занимал бы, сохраняется и продолжает влиять на расположение окружающего контента. Это легче понять на простом примере.

Я позиционировала встроенный элемент **em** (цвет фона делает его края видимыми). Для начала я использовала свойство **position**, чтобы установить метод в **relative**, затем — свойство смещения **top** для перемещения элемента на 30 пикселов вниз от его начального положения, и свойство **left** для перемещения его на 60 пикселов вправо. Как вы должны знать, значения свойства смещения отодвигают элемент от конкретного края, так что, если вы хотите переместить его вправо, как в этом примере, используйте свойство смещения **left**. Результаты показаны на рис. 15.16.

```
em {
position: relative;
top: 30px;
left: 60px;
background-color: fuchsia;
}
```

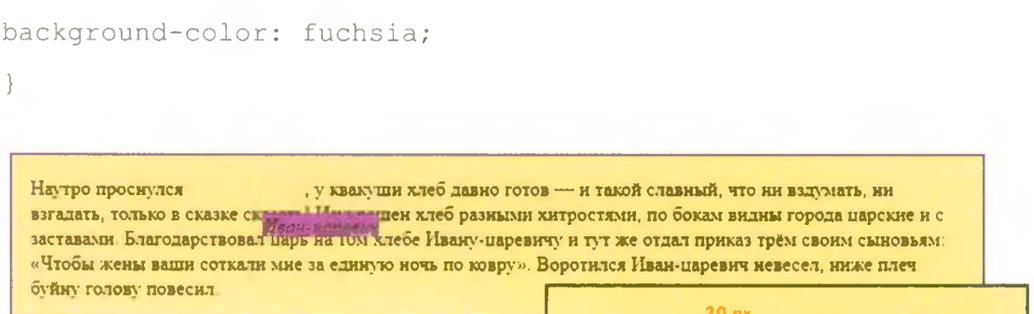


Рис. 15.16. Когда элемент позиционирован с помощью относительного метода, пространство, которое он занимал бы, сохраняется

Я хочу обратить внимание на некоторые моменты.

Исходное пространство в потоке документа сохраняется.

Как вы видите, на рисунке есть пустое пространство, где находился бы выделенный текст, если бы элемент не был позиционирован. Окружающий контент расположен так, как будто элемент все еще находится там, поэтому мы говорим, что элемент по-прежнему «влияет» на окружающий контент.

Происходит перекрытие.

Из-за того, что это позиционированный элемент, он потенциально может перекрыть другие, как показано на рис. 15.16.

Пустое пространство, оставленное относительно позиционированными объектами, может выглядеть некрасиво, поэтому данный метод применяется не так часто, как обтекание и абсолютное позиционирование. Однако относительное позиционирование широко используется для создания позиционирующего контекста для абсолютно позиционированных элементов, как я объясню в следующем разделе.

Абсолютное позиционирование

Абсолютное позиционирование выполняется немного по-другому и является действительно более гибким методом для совершенствования макетов страниц, чем относительное позиционирование. Теперь, когда вы увидели, как работает последнее, давайте возьмем тот же пример, какой показан на рис. 15.16, но на этот раз мы изменим значение свойства **position** на **absolute**.

```
em {
position: absolute;
top: 30px;
left: 60px;
background-color: fuchsia;
}
```

Как вы можете видеть на рис. 15.17, пространство, когда-то занимаемое элементом **em**, теперь сомкнуто, как это бывает для всех абсолютно позиционированных элементов. В своем новом положении блок элемента перекрывает окружающий контент. В конечном счете абсолютно позиционированные элементы совсем не влияют на расположение окружающих элементов.

Наиболее существенное отличие — это размещение позиционированного элемента. На этот раз значения смещения помещают элемент **em** на 30 пикселов вниз и 60 пикселов вправо от верхнего левого угла окна браузера.

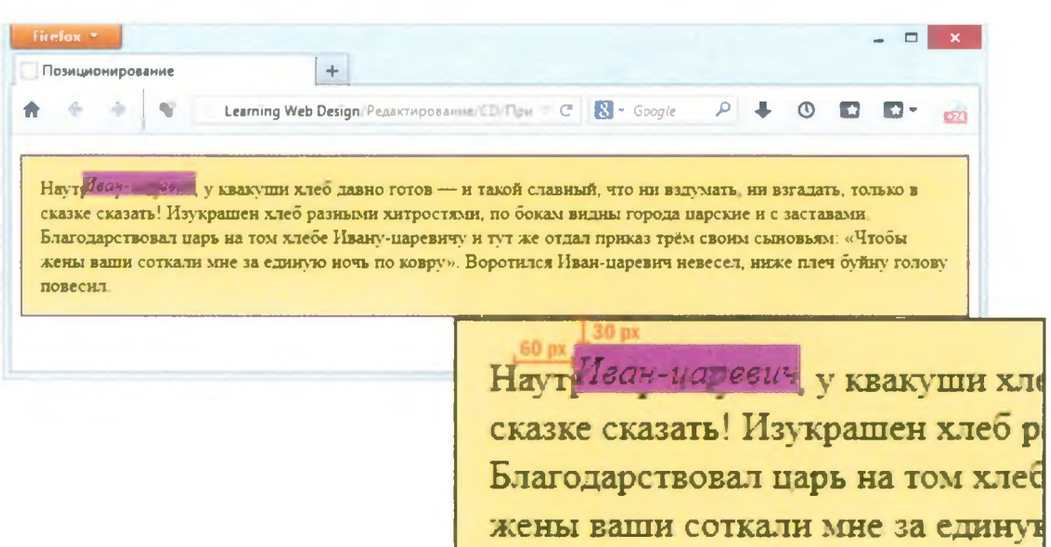


Рис. 15.17. Когда элемент абсолютно позиционирован, он удаляется из потока и пространство смыкается

Но подождите. Прежде чем подумать, что абсолютно позиционированные элементы всегда помещаются относительно окна браузера, стоит учесть еще кое-что.

При абсолютном позиционировании элемент помещается относительно его ближайшего *содержащего блока*. Так случилось, что ближайшим содержащим блоком на рис. 15.17 является корневой (`html`) элемент (также известный как *начальный содержащий блок*, поэтому значения смещения размещали элемент `em` относительно всей области окна браузера.

Понимание принципа содержащего блока является первым шагом в понимании абсолютного позиционирования.

ПРИМЕЧАНИЕ

Некоторые браузеры основывают начальный содержащий блок на элементе `body`. Конечный результат такой же, потому он что заполняет окно браузера.

Содержащие блоки

Рекомендация CSS2.1 утверждает: «Положение и размер блока(ов) элемента иногда вычисляются относительно определенного прямоугольника, называемого *содержащим блоком* элемента». Важно понимать принцип содержащего блока элемента, который вы хотите разместить. Иногда его называют *контекстом позиционирования*.

Рекомендации содержат ряд запутанных правил для определения содержащего блока элемента, но в основном это сводится к следующему:

- Если позиционированный элемент *не* содержится внутри другого позиционированного элемента, тогда он будет размещен относительно начального содержащего блока (созданного элементом `html`).
- Но если элемент имеет «предка» (то есть содержится внутри другого элемента), который имеет положение, установленное в `relative`, `absolute` или `fixed`, он вместо этого будет позиционирован относительно краев *того* элемента.

Или разместить его другим способом...

Содержащий блок для абсолютно позиционированного элемента является ближайшим *позиционированным* предшествующим элементом (то есть любой элемент со значением для свойства **position** отличным от **static**).

Если содержащего блока нет (другими словами, если один позиционированный элемент *не* содержится внутри другого), тогда будет использоваться начальный содержащий блок (созданный элементом **html**).

Рис. 15.17 приводит пример первого случая: элемент **p**, который содержит абсолютно позиционированный элемент **em**, *не* позиционирован сам, и нет других позиционированных элементов выше по иерархии, поэтому элемент **em** позиционирован относительно начального содержащего блока, который эквивалентен области окна браузера.

Давайте преднамеренно превратим элемент **p** в содержащий блок и посмотрим, что происходит. Все, что мы должны сделать, это применить к нему свойство **position**; мы не должны реально перемещать его. Наиболее распространенный способ превратить элемент в содержащий блок — это присвоить свойству **position** значение **relative**. Однако не следует перемещать его при помощи смещающих значений. (Кстати, это то, о чем я говорила ранее, когда сказала, что относительное позиционирование чаще всего используется для создания контекста для абсолютно позиционированного элемента).

Мы сохраним правило стилей для элемента **em** прежним, но добавим свойство **position** к элементу **p**, делая из него таким образом содержащий блок для позиционированного элемента **em**. На рис. 15.18 показан результат.

```
p {
position: relative;
padding: 15px;
background-color: #DBFDBA;
border: 2px solid #6C4788;
}
```

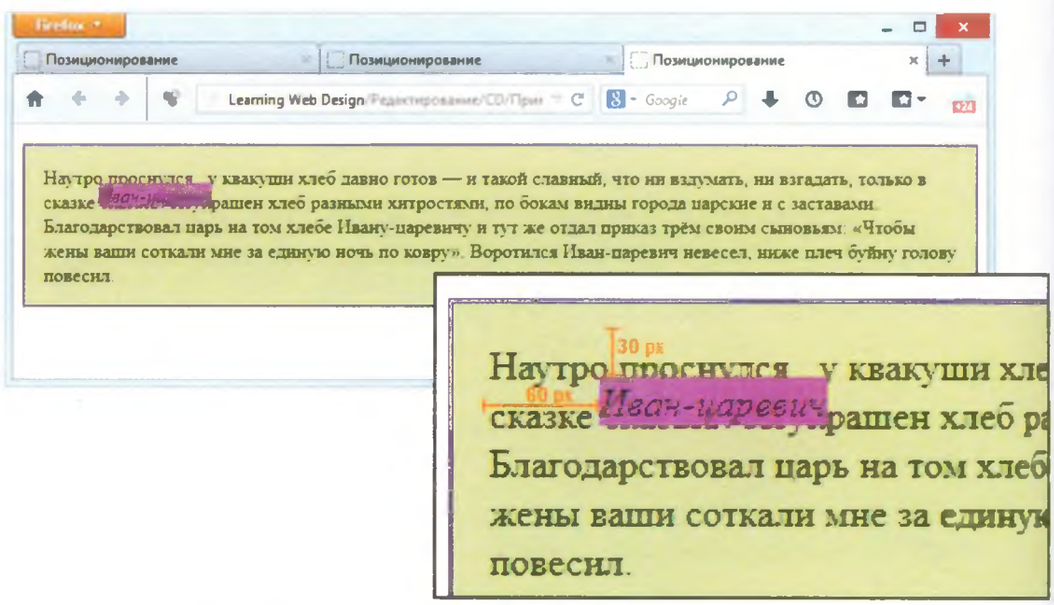


Рис. 15.18. *Относительно позиционированный элемент* **p** *действует как содержащий блок для элемента* **em**

Как вы видите, элемент **em** теперь расположен на 30 пикселов вниз и 60 пикселов от верхнего левого угла блока абзаца, не окна браузера. Заметьте также, что он позиционирован относительно *края отступа*

абзаца (точно внутри границы), но не края области контента. Это обычное поведение, когда блочные элементы используются как содержащие блоки (см. примечание).

Я собираюсь обнаружить дополнительные стороны абсолютно позиционированных объектов. На этот раз, я добавила свойства **width** и **margin** к позиционированному элементу **em** (рис. 15.19).

```
em {
width: 200px;
margin: 25px;
position: absolute;
top: 30px;
left: 60px;
background-color: fuchsia;
}
```

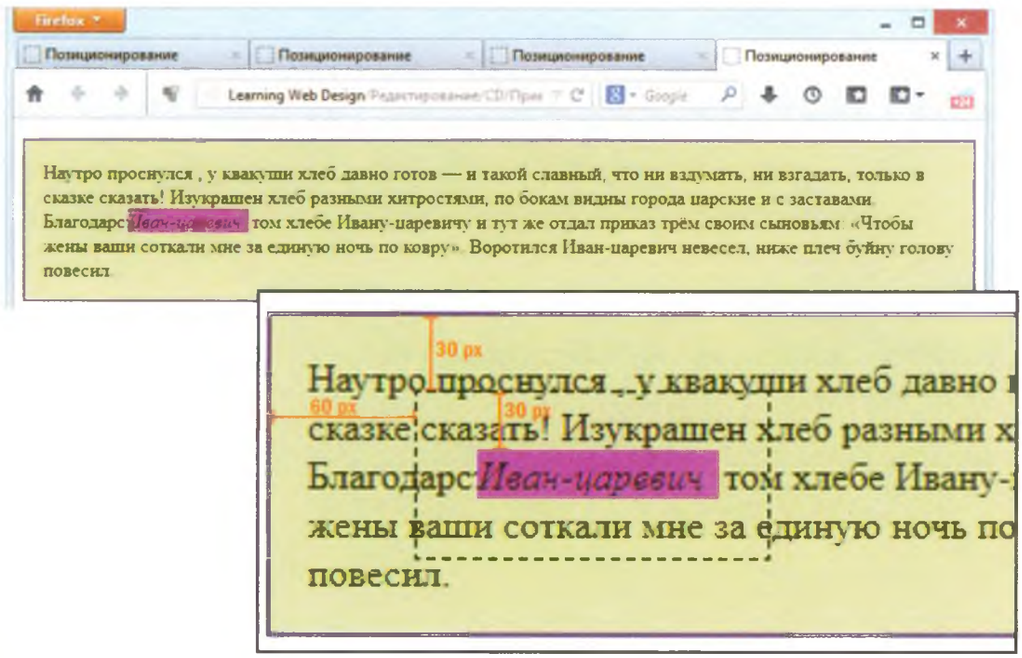


Рис. 15.19. Добавление ширины и полей к позиционированному элементу

Здесь мы можем видеть, что:

- Смещающие значения применяются к внешним краям блока элемента (между краями поля).
- Абсолютно позиционированные элементы всегда ведут себя как блочные. Например, поля со всех сторон сохраняются, даже если это встроенный элемент. Также разрешается устанавливать ширину для элемента.

Важно иметь в виду, что, как только вы позиционируете элемент, он становится новым содержащим блоком для всех элементов, которые содержит. Рассмотрите этот пример, в котором элемент **div** с именем «content» позиционирован в верхнем левом углу страницы. Когда позиционированному элементу списка внутри этого элемента **div** зада-

ПРИМЕЧАНИЕ

Когда встроенные элементы используются как содержащие блоки (а они могут), позиционированный элемент размещается относительно края области контента, но не края отступа.

ются смещающие значения, которые помещают его в правый верхний угол, он появляется в правом верхнем углу содержания элемента **div**, не всей страницы (рис. 15.20). Так происходит потому, что, позиционировав элемент **div**, он действует как содержащий блок для элемента **li**.

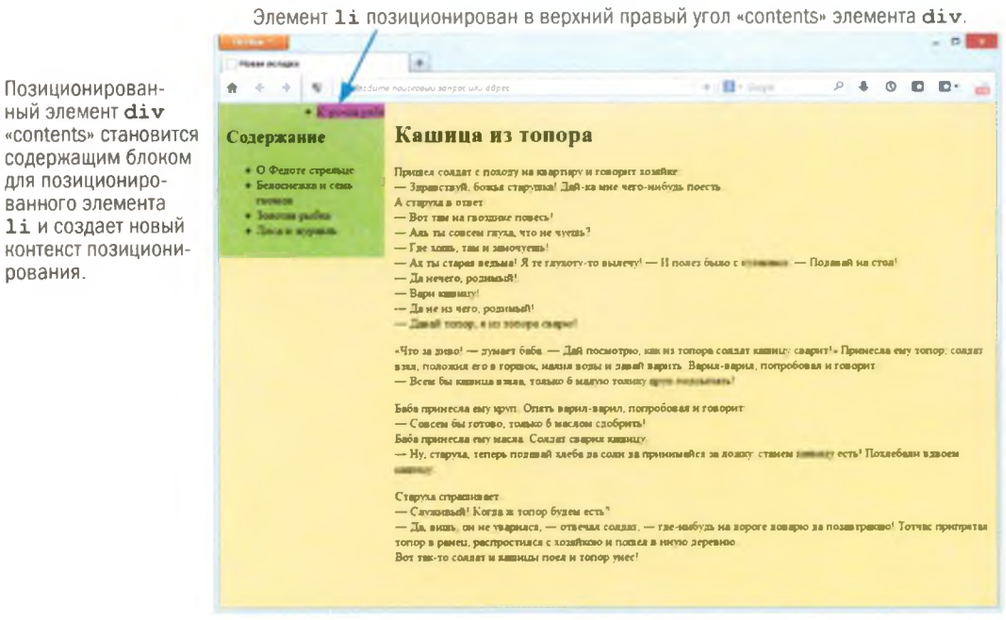


Рис. 15.20. *Позиционированные элементы становятся содержащим блоком для элементов, которые они содержат. В этом примере элемент списка позиционирован относительно содержащего элемента* **div**, *а не всей страницы*

Разметка

```
<div id="preface">
...
</div>

<div id="content">
<h2>Содержание</h2>
<ul>
<li>О Федоте стрельце</li>
<li id="special">Курочка ряба</li>
<li>Белоснежка и семь гномов</li>
<li>Золотая рыбка</li>
<li>Лиса и журавль</li>
</ul>
</div>
```

Таблица стилей

```
div#content {
width: 200px;
position: absolute;
top: 0; /* позиционирован в верхний левый угол */
left: 0;
background-color: #AFD479;
padding: 10px;
}
li#special {
position: absolute;
top: 0; /* позиционирован в верхний правый угол */
right: 0;
background-color: fuchsia;
}
```

Задание положения

Теперь, когда вы лучше понимаете принцип содержащего блока, давайте уделим немного времени тому, чтобы лучше познакомиться со свойствами смещения. До сих пор мы видели только перемещение элемента на несколько пикселов вниз и вправо, но это, конечно же, не все, что вы можете делать.

Измерения в пикселах

Как ранее упоминалось, положительные значения смещения *отодвигают* позиционированный блок элемента *от* заданного края по направлению к центру содержащего блока. Если для края не задано значение, оно устанавливается как **auto**, и браузер добавляет достаточно пространства, чтобы выполнить работу по размещению. В этом примере я использовала значение длины в пикселах для всех четырех свойств смещения, чтобы поместить позиционированный элемент в конкретное место его содержащего элемента (рис. 15.21).

```
div#a {
position: relative; /* создает содержащий блок */
height: 120px;
width: 300px;
border: 1px solid;
background-color: #CCC;
}
```

```
div#b {
position: absolute;
top: 20px;
right: 30px;
bottom: 40px;
left: 50px;
border: 1px solid;
background-color: teal;
}
```

Рис. 15.21. *Установка значений смещения для всех четырех сторон позиционированного элемента*

Обратите внимание, что, установив смещение на всех четырех сторонах, я косвенно установила размеры позиционированного элемента **div#b** (он занимает пространство в 60×220 пикселов, которое оставлено внутри содержащего блока после применения значений смещения). Если бы я также задала ширину и другие свойства блока для элемента **div#b**, возникла бы вероятность конфликтов, если итоговая сумма значений для позиционированного блока и его смещений не соответствовала доступному пространству внутри содержащего блока.

Спецификация CSS предоставляет устрашающий набор правил для регулирования конфликтов, но вывод — вы просто должны быть осторожными и не определять чересчур много свойств блока и смещений. В общем, ширина (плюс необязательные отступ, граница и поле) и одно или два свойства смещения — все, что необходимо для получения нужного вам макета. Позвольте браузеру позаботиться об оставшихся вычислениях.

Процентные значения

Вы можете также задать положения при помощи процентных значений. В первом примере на рис. 15.22 изображение сдвинуто вниз на половину левого края содержащего блока. Во втором примере справа элемент

img позиционирован таким образом, что всегда появляется в нижнем правом углу содержащего блока.

```
img#A {
position: absolute;
top: 50%;
left: 0%; /* символ % может быть пропущен для значения 0*/
}
img#B {
position: absolute;
bottom: 0%; /* символ % может быть пропущен для значения 0 */
right: 0%; /* символ % может быть пропущен для значения 0 */
}
```

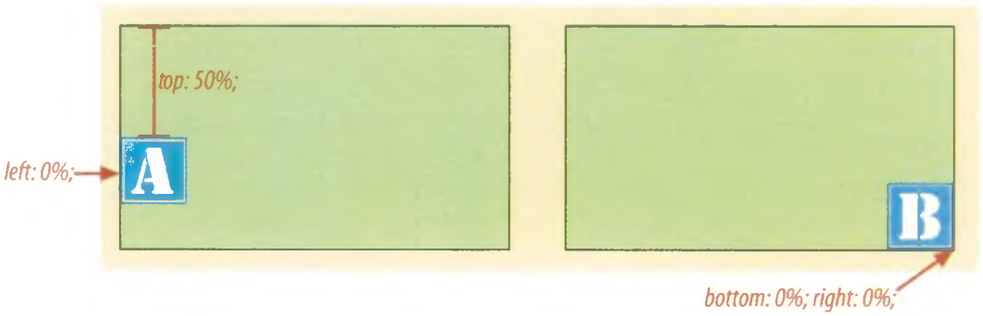


Рис. 15.22. Использование процентных значений для размещения элемента в нижнем углу содержащего блока

Несмотря на то что в примере указаны смещения и по горизонтали, и по вертикали, для позиционированного элемента принято задавать только одно смещение, например вправо или влево на поле с помощью свойства **left** или **right**.

В упражнении 15.4 мы продолжим работу с главной страницей интернет-магазина «Малышок», применив на этот раз абсолютное позиционирование.

ПРЕДУПРЕЖДЕНИЕ

Будьте осторожны при позиционировании элементов внизу начального содержащего блока (элемент **html**). Хотя вы можете ожидать, что он окажется внизу всей страницы, браузеры на самом деле помещают элемент в нижний угол своего окна. Результаты могут быть непредсказуемы. Если вы хотите разместить контент в нижнем углу вашей страницы, поместите его в содержащий блочный элемент в конце исходного документа.

УПРАЖНЕНИЕ 15.4. АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ

В этом упражнении мы будем использовать абсолютное позиционирование для добавления изображения награды сайту и создания макета с двумя колонками. Откройте версию файла *malishok.html*, которую сохранили перед выполнением упражнения 15.3 (или *malishok_ch15.html*), в текстовом редакторе. Вы должны начать с одноколоночного макета с плавающими изображениями и горизонтальным меню.

1. Допустим, что сайт «Малышок» выиграл награду «Самый-самый сайт недели», и теперь у нас есть шанс отобразить небольшой баннер награды на домашней странице. Из-за того, что это новый контент, нам понадобится добавить его в разметку. Из-за того, что это не необходимая информация, мы добавим изображение в новый элемент **div** в самый конец документа, после абзаца об охране авторского права.

```
<div id="award">
<img src="images/awesomesite.gif"
alt="Самый-самый сайт недели">
</div>
```

То, что контент находится в конце исходного документа, не означает, что он должен отображаться в нижней части страницы. Мы можем использовать абсолютное позиционирование, чтобы разместить элемент **div**, как показано ниже:

```
#award {
position: absolute;
top: 35px;
left: 25px;
}
```

Сохраните документ и взгляните на него (рис. 15.23). Сделайте размер окна браузера очень узким, и вы увидите, что позиционированное изображение награды перекрывает заголовок. Также обратите внимание, что когда вы прокручиваете документ, изображение прокручивается вместе с остальной страницей. Попробуйте поэкспериментировать с другими свойствами и значениями смещения, чтобы изменить позиционирование в окне браузера (или, если использовать правильный термин, «начальном содержащем блоке»).

Рис. 15.23. Абсолютно позиционированное изображение награды

2. В упражнении 15.3 мы создали двухколоночный макет, применив плавающий элемент. Посмотрим, можем ли мы выполнить то же самое при помощи абсолютного позиционирования. В этот раз сделаем блок отзывов фиксированной ширины и позволим блоку товаров растянуться и занять оставшееся пространство. Это еще один распространенный подход, который следует научиться применять.

В том виде, как документ выглядит сейчас, если мы позиционируем элемент **div** раздела «testimonials», он будет располагаться относительно окна браузера, а нам это не нужно. Мы хотим, чтобы он всегда появлялся под элементом **#nav div**, поэтому начнем с создания нового содержащего блока после элемента **#nav**, который будет содержать элементы **div** разделов «products» и «testimonials» и служить в качестве нового позиционирующего контекста.

Это потребует некоторых изменений в разметке. Вставьте разделы **#products** и **#testimonials** в новый элемент **div** с идентификатором контента **id**. Структура документа должна выглядеть так:

```
<div id="content">
<div id="products">... </div>
<div id="testimonials"> ... </div>
</div>
<p class="copyright">...</p>
```

3. Теперь мы можем превратить элемент **div** «content» в содержащий блок, позиционировав его приемом «неподвижное относительное положение»

```
#content {
position: relative;
}
```

4. Задав его положение, мы можем позиционировать **#testimonials** в верхнем правом углу элемента **#content div**. Добавьте позиционирование, а также свойства **top** и **right** к правилу **#testimonials**, как показано ниже. Кроме того, задайте ширину контента равной 14em. Установите значение верхнего поля 0 и поменяйте значения правого и левого полей с 10% на 1 em.

```
#testimonials {
...
margin: 1em 10%; 0 1em;
position: absolute;
top: 0;
right: 0;
width: 14em;}
```

5. Если вы сохраните документ и просмотрите его в браузере, то должны увидеть, что блок отзывов находится в правом углу, прямо поверх блока товаров. Следующий шаг — задать правое поле блока товаров, чтобы предоставить пространство отзывам. Но сколько пространства? Давайте посчитаем.

6. Блок отзывов складывается из примерно 3.5em левого отступа (55px), ширины контента равной 14em, 1em правого отступа и 1em правого поля, что в сумме составляет 19.5em. Если мы установим ширину правого поля элемента **#products** равную 20.5em, у нас появится место для блока отзывов, плюс немного пространства между колонок. Мы сделаем это как показано ниже.

```
#products {
...
margin: 1em 20.5em 1em 1em;
...
}
```

Сохраните документ и просмотрите его в браузере (рис. 15.24). Измените размер окна и посмотрите, как ведут себя блоки по сравнению с предыдущим примером с плавающей колонкой.

Проверка в реальных условиях

Перед тем как вы слишком воодушевитесь легким созданием многоколоночных макетов при помощи абсолютного позиционирования, позвольте мне отметить, что это упражнение представляет наиболее благоприятное развитие событий, при котором позиционированная боковая колонка почти гарантированно будет короче основного контента. Также нет значительного нижнего колонтитула, о котором стоит беспокоиться. Если бы врезка стала длиннее с большим количеством отзывов, она бы во всю ширину наложилась на любой колонтитул, который может оказаться на странице, что далеко от совершенства. Считайте это небольшим замечанием, так как к этому можно еще многое добавить, как мы увидим в главе 16.

Рис. 15.24. Макет в две колонки, созданный с помощью приема абсолютного позиционирования врезки с отзывами

Порядок наложения

Перед тем как мы завершим тему абсолютного позиционирования, есть еще один, последний, связанный с ним принцип, который я должна представить. Как мы видели, абсолютно позиционированные элементы перекрывают другие, отсюда следует, что несколько позиционированных элементов имеют вероятность расположиться друг на друге.

По умолчанию элементы располагаются в том порядке, в котором появляются в документе, но вы можете изменить его при помощи свойства **z-index**. Представьте *ось Z* как линию, которая проведена от кончика вашего носа перпендикулярно странице.

z-index

Принимаемые значения: *число* | **auto** | **inherit**

Значение по умолчанию: **auto**

Применение: к позиционированным элементам

Наследование: нет

Значением свойства **z-index** является число (положительное или отрицательное). Чем больше число, тем выше в стеке будет появляться элемент. Давайте посмотрим пример, чтобы стало понятней (рис. 15.19).

В примере есть три элемента абзаца, каждый содержит изображение буквы (A, B и C соответственно), они позиционированы таким образом, что перекрывают друг друга на странице. По умолчанию абзац «C» появился бы сверху, потому что он указан последним в исходном коде. Однако, присвоив более высокие значения **z-index** абзацам «A» и «B», мы можем их наложить в нашем порядке предпочтения.

Обратите внимание, что значения свойства **z-index** не обязаны быть последовательными, и, в частности, они не соотносятся с чем-нибудь определенным. Это значит, что более высокие значения размещают элемент выше в стеке.

Разметка

```
<p id="A"><img src="A.gif" alt="A"></p>
<p id="B"><img src="B.gif" alt="B"></p>
<p id="C"><img src="C.gif" alt="C"></p>
```

Таблица стилей

```
#A {
z-index: 10;
position: absolute;
top: 200px;
left: 200px;
}
#B {
z-index: 5;
position: absolute;
top: 225px;
left: 175px;
}
#C {
z-index: 1;
position: absolute;
top: 250px;
left: 225px;
}
```

Честно говоря, свойство **z-index** не часто требуется для большинства макетов страниц, но вы должны знать, что оно есть, если понадобится. В случае, когда вы хотите определить, чтобы позиционированный

элемент всегда оказывался сверху, присвойте ему очень большое значение **z-index**, например:

```
img#essential {
z-index: 100;
position: absolute;
top: 0px;
left: 0px;
}
```

Рис. 15.25. *Изменения порядка наложения при помощи свойства z-index*

Фиксированное позиционирование

Мы охватили относительное и абсолютное позиционирование, теперь пришло время изучить оставшийся метод: *фиксированное позиционирование*.

В основном фиксированное позиционирование работает точно так же, как абсолютное. Существенное различие в том, что значения смещения для фиксированных элементов *всегда* относительны окну просмотра, а значит, позиционированный элемент остается на месте, даже когда остальная часть страницы прокручивается. Вы можете вспомнить, что когда прокручивали страницу интернет-магазина «Малышок» в упражнении 15.4, изображение награды прокручивалось вместе с документом — хотя он был позиционирован относительно начального содержащего блока (эквивалентного окну браузера). Все иначе в случае фиксированного позиционирования, где положение *фиксировано*.

Фиксированные элементы часто используются для создания меню, которое остается на одном и том же месте в верхней, нижней или боковой части экрана, чтобы быть доступным даже при прокручивании контента. См. врезку «Избегайте использовать свойство position:fixed на мобильных устройствах», содержащую предупреждение о потенциальных проблемах.

Давайте переключим изображение награды на странице сайта «Малышок» в фиксированное позиционирование, чтобы увидеть разницу.

Избегайте использовать свойство position:fixed на мобильных устройствах

На момент написания книги свойство **position: fixed** становилось причиной странностей в поведении многих мобильных браузеров. Некоторые считали его просто статическим, позволяя прокручиваться вместе с остальным контентом. Некоторые «поддерживающие» его браузеры прокручивали фиксированный элемент с экрана, но возвращали его на место, как только прокручивание прекращалось (и при этом как минимум один браузер неправильно рассчитывал его местоположение), доставляя неудобства пользователям. Другие приводили к нестабильной работе всего сайта. Существует несколько способов исправления этих ошибок, но у них есть свои недостатки. Одним из решений является отключение возможности изменения пользователем масштаба страницы, но при этом исчезает полезная для удобства характеристика. Другой вариант заключается в использовании решения JavaScript для создания правильного поведения для фиксированного позиционирования, но это вводит новый уровень сложности и вероятность несовместимой поддержки. Лучшим выходом будет подумать, нужен ли вообще фиксированный элемент для удобства пользования, а затем изучать варианты JavaScript по мере необходимости. Чтобы изучить все возможные проблемы и их решения посредством языка JavaScript, я рекомендую статью Брэда Фроста, опубликованную на сайте **bradfrostweb.com/blog/mobile/fixed-position/**.

УПРАЖНЕНИЕ 15.5. ФИКСИРОВАННОЕ ПОЗИЦИОНИРОВАНИЕ

Это должно быть просто. Откройте страницу сайта «Малышок» и отредактируйте правило стилей для элемента **#award div**, чтобы сделать его положение фиксированным, а не абсолютным.

```
#award {
position: fixed;
top: 35px;
left: 25px;
}
```

Сохраните документ и откройте его в браузере. Однако когда прокрутите страницу, вы увидите, что награда зафиксирована в позиции, где мы ее разместили в окне браузера (рис. 15.26).

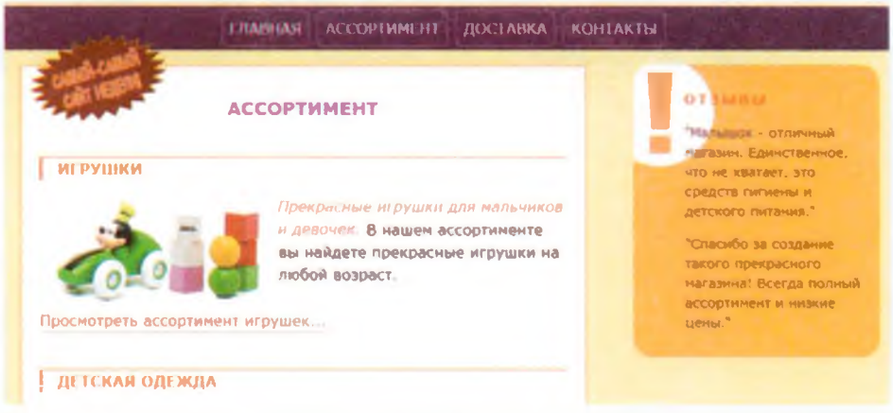


Рис. 15.26. *Изображение награды зафиксировано в верхнем левом углу окна браузера, когда документ прокручивается*

Теперь вам представлены все инструменты верстки макета на основе CSS: обтекание и три типа позиционирования (относительное, абсолютное и фиксированное). Вы должны хорошо представлять, как они работают, прежде чем применять их в различных подходах к дизайну и шаблонам в главе 16.

Резюме

Ниже приведена сводка свойств, охваченных в этой главе, в алфавитном порядке.

Свойство	Описание
`float`	Передвигает элемент вправо или влево и позволяет расположенному ниже тексту обтекать его.
`clear`	Предотвращает элемент от размещения рядом с плавающим элементом
`position`	Задает метод позиционирования, который должен быть применен к элементу
`top`, `bottom`, `right`, `left`	Задает величину смещения от каждого соответствующего края.
`z-index`	Задает порядок появления позиционированных элементов внутри стека наложения.

ГЛАВА 16

МАКЕТЫ СТРАНИЦ СРЕДСТВАМИ CSS

Теперь, когда вы понимаете принципы размещения элементов на странице с помощью позиционирования и плавающих элементов, мы можем использовать эти инструменты в некоторых стандартных макетах страниц. Данная глава рассматривает различные подходы к веб-дизайну средствами CSS и предоставляет простые шаблоны, которые откроют вам путь к созданию основных веб-страниц с двумя и тремя колонками.

Прежде чем мы начнем, необходимо сказать, что существует бесчисленное множество вариантов создания многоколоночных макетов средствами CSS. Эта глава предназначена быть «стартовым набором». Шаблоны, представленные здесь, упрощены и могут работать не в каждой ситуации, но я постаралась указать на существенные недостатки каждого.

В этой главе

- Фиксированные, жидкие и эластичные макеты страниц
- Шаблоны макетов с двумя и тремя колонками с использованием плавающих элементов
- Независимый от исходного кода макет с использованием плавающих элементов
- Шаблоны макетов с тремя колонками с использованием абсолютного позиционирования
- Псевдоколонки

Стратегии верстки страниц

Перед тем как начать разбирать на части CSS-макеты, давайте поговорим о различных вариантах структурирования веб-страниц. Как уже неоднократно было сказано, пользователи просматривают их в браузерах любого размера — от крошечных дисплеев смартфонов до огромных экранов ЖК-телевизоров, а иногда могут менять размер шрифта отображаемого текста. Все это неизбежно влияет на макет страницы. Со временем появились три основных подхода к верстке страниц, учитывающих эти факты:

- *Фиксированные макеты* размещают контент в области страницы, ширина которой задана в пикселах и остается неизменной, независимо от величины окна браузера или размера шрифта текста.
- *Жидкие (или резиновые) макеты* меняют свой размер в соответствии с окном браузера.
- *Эластичные макеты* пропорционально меняют свой размер в зависимости от размера шрифта отображаемого текста.
- *Гибридные макеты* совмещают фиксированные и масштабируемые области.

Давайте рассмотрим работу этих стратегий, а также доводы «за» и «против» использования каждой из них.

Оптимальная длина строки

Длина строки — это количество слов или символов в строке текста. Опытным путем установлено, что оптимальная длина строки — от 10 до 12 слов, или между 60 и 75 символами.

Когда длины строк становятся слишком большими, текст сложнее читать. Приходится долго концентрировать внимание, добираясь до конца длинной строки, и чтобы снова найти начало следующей, тоже требуется дополнительное усилие.

Длина строки — основная причина споров о том, какой прием верстки лучше. В жидких макетах строки могут становиться слишком длинными, когда окно браузера устанавливается очень широким. В дизайнах с фиксированной шириной — нескладно короткими, если размер шрифта крупный, а текст размещен в узких колонках. Однако эластичный макет, рассматриваемый позже в этой главе, предлагает предсказуемые длины строк, даже когда шрифт текста становится крупнее. Этот вариант наиболее распространен, поскольку дает оптимальный баланс дизайна и удобства просмотра.

Фиксированные макеты

Макеты с фиксированной шириной, как и предполагает название, сохраняют конкретную ширину, указанную дизайнером в пикселах. Они напоминают печатные макеты, поскольку позволяют разработчику управлять взаимосвязью элементов страницы, выравниванием и длиной строк (см. врезку «Оптимальная длина строки»). Этот подход к созданию макетов стал популярным благодаря тому, что люди чаще просматривали веб-страницы на довольно крупном экране монитора настольного компьютера, и веб-дизайнерам несложно было представить, как будет выглядеть результат их труда на большинстве экранов. Но времена изменились, и мы больше не делаем таких предположений и не стремимся к пиксельному совершенству.

Когда вы задаете для страницы конкретную ширину, не следует забывать о паре моментов. Для начала вам нужно подобрать ширину, обычно основанную на распространенном разрешении мониторов. На момент написания книги большинство веб-страниц проектировались так, чтобы их ширина была 960 пикселов или около того — для качественного отображения при наиболее распространенном разрешении монитора 1024×768 пикселов. Некоторые дизайнеры делают страницы более узкими; другие предпринимают попытки создавать их все более широкими по мере того, как увеличивается разрешение мониторов. В любом случае нужно принять решение связанное с выбором дизайна.

Вам также следует решить, где именно в окне браузера должен быть позиционирован макет с фиксированной шириной. По умолчанию он расположен с левого края, а справа остается дополнительное пространство. Вы также можете центрировать страницу, распределяя дополнительное пространство между левым и правым полями, что позволяет странице лучше заполнять окно браузера. На рис. 16.1 показаны два таких макета, позиционированных по-разному.

Одна из главных характерных черт макетов фиксированной ширины — если пользовательское окно браузера не такое широкое, как страница, контент в ее правой части не будет виден. Можно использовать горизонтальную прокрутку, но не всегда понятно, что часть контента скрыта. Кроме того, хотя структура страницы не меняется, если пользователь задаст очень крупный размер шрифта, чтобы облегчить прочтение текста, в строке может оказаться всего несколько слов, и макет будет казаться неуклюжим или вовсе развалится.

Рассмотрим плюсы и минусы использования макетов с фиксированной шириной.

Положительные стороны	Отрицательные стороны
Макет предсказуем и предлагает лучший контроль над длиной строки.	Контент в правой части будет скрыт, если ширина окна браузера меньше, чем страницы.
Его проще проектировать и разрабатывать.	На экранах с большой диагональю может оставаться определенное количество лишнего пространства.
Он ведет себя так, как большинство веб-страниц (на момент написания данной книги), но это может измениться, поскольку люди все чаще посещают веб-страницы не с настольного компьютера, а с помощью иных устройств.	Длины строк могут стать излишне короткими при очень больших размерах текста.
	Лишает пользователя контроля над шириной области отображения контента.

```
#wrapper {width: 750px;
  position: absolute;
  margin-left: auto;
  margin-right: auto;
  border: 1px solid black;
  padding: 0px;}

#extras {position: absolute;
  top: 0px;
  left: 0px;
  width: 200px;
  background: orange; }

#main {margin-left: 225px;
  background-color: yellow;}
```

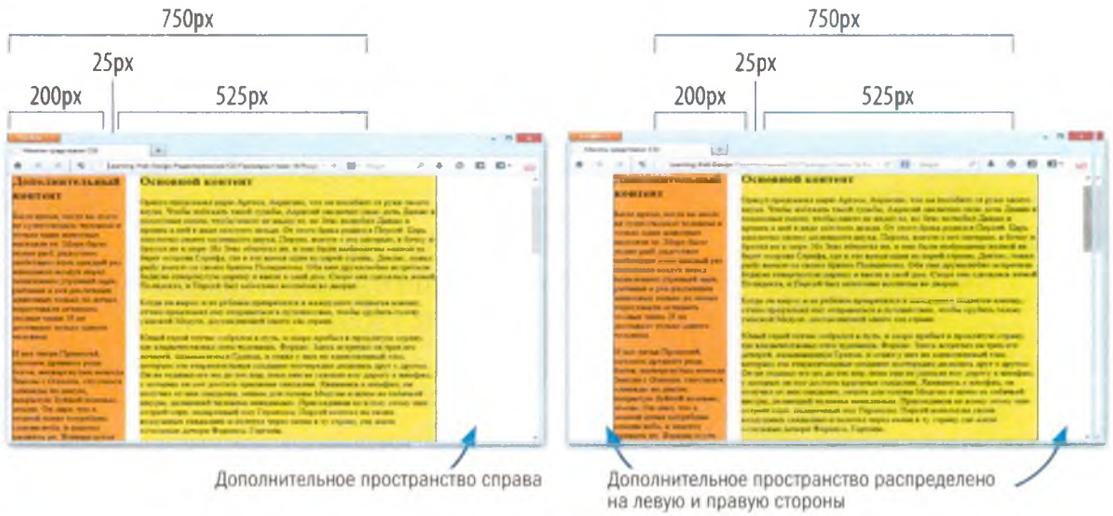


Рис. 16.1. Примеры фиксированных макетов (выровненных по левому краю и центру)

Как создавать макеты с фиксированной шириной

Макеты с фиксированной шириной можно создать, задав значение ширины в пикселах. Обычно содержимое всей страницы вставляется в элемент **div** (часто называемый «контент», «контейнер», «оболочка» или «страница»), ширина которого потом может быть установлена в конкретное пиксельное значение. Этот элемент **div** также может быть центрирован в окне браузера. Ширина колонок и даже полей и отступов тоже задается в пикселах. Как это сделать, мы рассмотрим чуть позже.

Системы сеток CSS

Специалисты использовали сетки для выравнивания и организации контента с первых дней графического дизайна — системы сеток стали для них полезным инструментом. Сетка — невидимая основа, делящая страницу на равные части, которые могут быть использованы для позиционирования колонок, заголовков, изображений и так далее (рис. 16.2). Придерживаясь элементов сетки, вы не только гарантируете, что ваш контент будет пропорциональным, но и получаете возможность ускорить проектные решения. Многие сетчатые макеты CSS появились, чтобы упростить процесс дизайна и разработки. Пожалуй, самым известным является система сеток 960 (**960.gs**), которая делит страницу шириной 960-пикселов на 12 или 16 колонок. Blueprint (**www.blueprintcss.org**) и BlueTrip (**bluetrip.org**) основаны на подобных сетках фиксированной ширины. Для жидкой сетки с двумя и тремя колонками существует система YUI2 компании Yahoo! (**developer.yahoo.com/yui/grids/**).

С появлением мобильных устройств набирают популярность адаптивные системы сеток, в том числе 1140 CSS Grid (**cssgrid.net**), Skeleton (**getskeleton.com**) и Bootstrap компании Twitter (**twitter.github.com/bootstrap**).

Конечно, это всего лишь краткий экскурс в область систем сеток CSS и в списке указаны лишь некоторые из многих. В любом случае, поищите во Всемирной паутине последние и самые лучшие. Для применения системы потребуются большие фрагменты кода HTML и CSS, но они могут сэкономить ваше время. Недостатком является то, что этот код, как правило, более раздутый, чем если бы он был написан вручную, и, возможно, вы будете вынуждены загружать ненужные данные. По этой причине некоторые дизайнеры используют системы сеток для ускорения процесса дизайна, но создают пользовательский код на финальной стадии разработки сайта.

Если вы хотите узнать больше о системах сеток и их преимуществах, я рекомендую книгу Чои Вина «Как спроектировать современный сайт» (Питер, 2011).

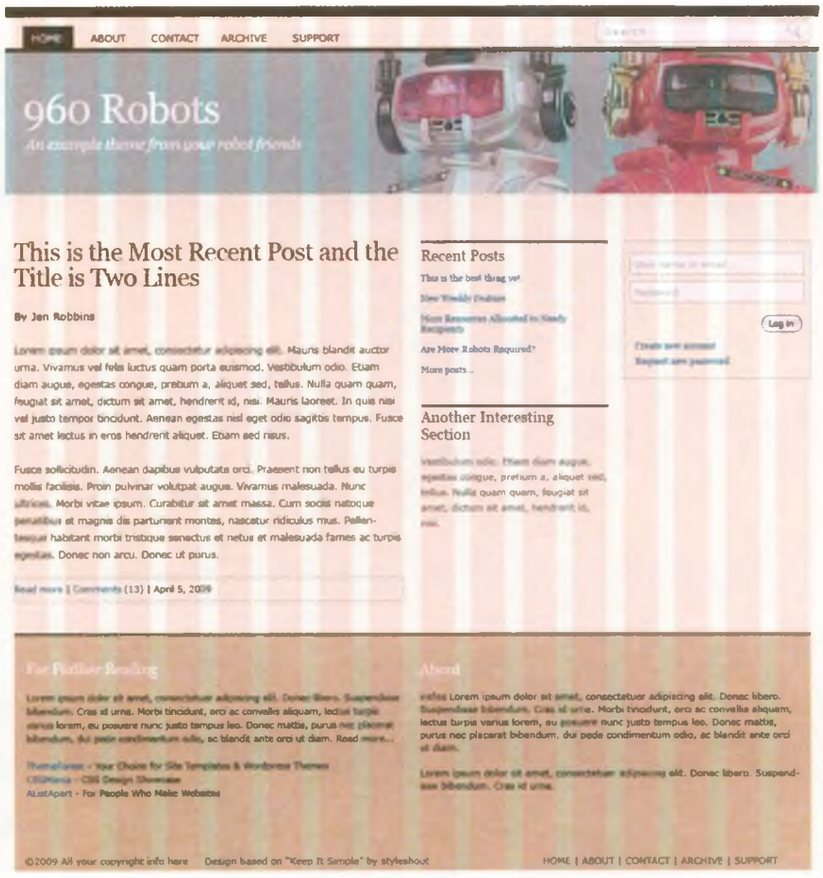


Рис. 16.2. Пример дизайна веб-страницы с использованием 16-колоночной системы сеток

Жидкие макеты

В *жидких макетах* страниц (также называемых *текучими* макетами) область страницы и колонки внутри нее становятся шире или уже, чтобы заполнить пространство, доступное в окне браузера. Другими словами, они следуют заданному по умолчанию поведению нормального потока. Ширина контента или разрывы (переносы) строк не контроли-

руются — тексту разрешено повторно заливаться при изменении ширины окна. На рис. 16.3 показан сайт **W3.org**, который является хорошим примером жидкого макета.

Жидкие макеты заполняют окно браузера. Контент переза-ливается, когда окно браузера и колонки меняют размер.

www.w3.org

Рис. 16.3. Пример жидкого макета

Жидкие макеты являются краеугольным камнем метода адаптивного веб-дизайна. Сейчас, когда веб-дизайнеры примиряются с огромным разнообразием размеров окна браузера и экранов, особенно тех, которые существенно меньше традиционных мониторов настольных компьютеров и ноутбуков, многие переходят на дизайны, которые меняются, чтобы заполнить ширину окна браузера, какой бы она ни была. Поскольку бесполезно пытаться создать дизайн фиксированной ширины для каждого размера экрана, я думаю, что жидкие макеты ждет возрождение. Разумеется, этот метод также имеет как плюсы, так и минусы.

Положительные стороны	Отрицательные стороны
Жидкие макеты соответствуют характеристикам среды просмотра.	На больших мониторах длины строк могут стать очень большими и неудобными для чтения.
Они убирают потенциально пустое пространство, так как текст заполняет окно.	Поведение макетов менее предсказуемо. Элементы могут быть расположены слишком рассеяно или, наоборот, сжато при крайне больших (или малых) размерах окна браузера.
На мониторах (экранах) компьютеров пользователи могут управлять шириной окна и контента.	Сложнее добиться появления «воздуха» (пустого пространства).
Нет горизонтальных полос прокрутки.	Для вычисления размеров требуется больше математических расчетов.

ПРИМЕЧАНИЕ

Итан Маркотт (автор термина «адаптивный веб-дизайн») говорит о создании сайта по стандартам консорциума Всемирной паутины с помощью жидкой сетки в своей книге «Отзывчивый веб-дизайн» (Манн, Иванов и Фербер, 2012). Это доказательство того, что использование жидких макетов не ведет к потере контроля над дизайном.

Как создавать жидкие макеты

Создавайте жидкий макет, задавая ширину в процентных значениях. Вы можете также не задавать ее вообще, в этом случае ширина будет установлена в значение по умолчанию `auto`, а элемент заполнит доступную ширину окна или другого содержащего элемента.

Создавайте жидкий макет, задавая ширину в процентных значениях или не задавайте значений вовсе.

В этом макете с двумя колонками (рис. 16.4) ширина каждого элемента `div` была задана в процентах от доступной ширины страницы. Основная колонка будет всегда 70% от ширины окна, а правая заполнит 25% (оставшиеся 5% используются для полей между ними), независимо от размера окна. Вы уже опробовали этот подход, когда создавали колонку с плавающим элементом в упражнении 15.3. Один из потенциальных недостатков жидких макетов — чересчур длинные строки, но вы можете предотвратить разрастание макета до смешной ширины, указав максимальную ширину страницы (см. врезку «Использование свойства `max-width`» далее в этой главе) Вы также можете использовать свойство `min-width`, чтобы не позволить странице стать излишне узкой. Это предоставляет некоторые преимущества перед фиксированным макетом, в то же время обеспечивая гибкость промежуточных размеров.

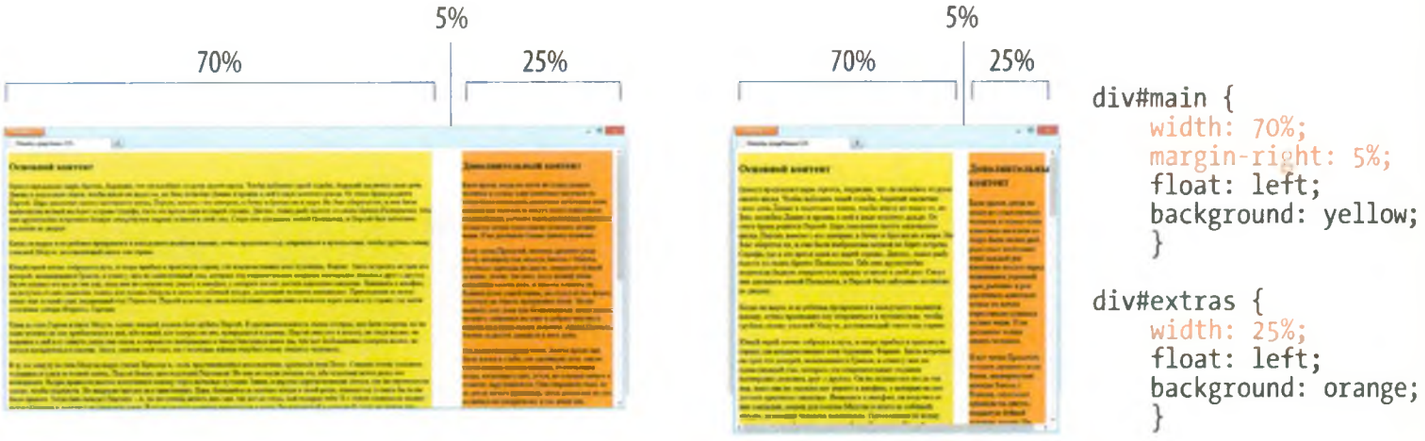


Рис. 16.4. Жидкий макет, спроектированный с помощью процентных значений

Эластичные макеты

ПРИМЕЧАНИЕ

На сайте **habrahabr.ru/post/21209/** опубликована статья об эластичных макетах, которая, несмотря на то, что написана довольно давно, содержит полезные подробности этого метода.

Третий подход объединяет возможность переопределять размер шрифта с предсказуемыми длинами строк. *Эластичные макеты* увеличиваются или уменьшаются вместе с размером текста. Если пользователь увеличивает размер шрифта, блок, который его содержит, увеличивается пропорционально. Аналогично, если пользователю нравится очень мелкий размер шрифта, блок уменьшится, чтобы подходить по размеру. Результат — длины строк (количество слов или символов в каждой) остаются неизменными независимо от размера шрифта текста. Это является преимуществом по сравнению с жидкими макетами, где длины строк могут становиться слишком большими, и фиксированными макетами, где очень большой шрифт в результате вызывает нескладные несколько символов на строку.

На рис. 16.5 показан дизайн «Elastic Lawn» Патрика Гриффитса на сайте CSS Zen Garden (**www.csszengarden.com/?cssfile=/063/063.css**) — старый, но подходящий для демонстрации пример эластичного макета в действии. Обратите внимание, что когда размер шрифта становится

больше, область контента страницы также увеличивается. Однако вместо удлинения строк их переносы сохраняются.

Рис. 16.5. Дизайн «Elastic Lawn» Патрика Гриффитса на сайте CSS Zen Garden является классическим примером эластичного макета страницы

Функция *масштабирования всей страницы*, предлагаемая большинством современных браузеров, украла часть славы эластичного дизайна. Теперь все веб-страницы масштабируются пропорционально, но эластичные макеты по-прежнему могут решать проблемы, связанные с изменением пользовательских настроек размера шрифта, заданных по умолчанию в браузере.

Сторонникам эластичных дизайнов нравится, что пропорции страницы привязаны к печатному контенту. Сегодня, когда размеры экрана неизвестны, имеет смысл создавать дизайны, беря за основу элементы контента. Однако у эластичных макетов на экранах с большой диагональю возникают те же проблемы, что и у фиксированных (которые все же можно устранить с помощью свойства **`max-width`**) и, как правило, они менее полезны в мобильных устройствах, чем жидкие макеты. Еще одним недостатком является то, что, хотя масштаб сетки страницы меняется вместе с текстом, встроенные мультимедийные элементы, такие как изображения и видеоролики, остаются неизменными (существуют решения и этой проблемы, но они выходят за рамки данной главы).

Пришло время рассмотреть плюсы и минусы эластичных макетов:

Положительные стороны	Отрицательные стороны
Обеспечивает согласованность фиксированного макета, допуская гибкость при изменении размера шрифта. Контроль над длинами строк жестче, чем в жидком и фиксированном макетах.	Изображения и видеоролики не масштабируются автоматически вместе с текстом и остальной частью макета (но есть способы это изменить). Ширина макета может превысить ширину окна браузера при самых крупных размерах шрифта. Не так полезен при использовании на различных устройствах и при различных размерах окна браузера. Его сложнее создать, чем макет с фиксированной шириной.

Как создавать эластичные макеты

Эластичные макеты создаются, если задать ширину в единицах измерения em.

Ключом к эластичным макетам является em — единица измерения, которая основана на размере шрифта. Например для элемента с размером шрифта в 16 пикселов единица em равна 16 пикселам. Принято задавать размер шрифта в единицах измерения em. В эластичных макетах размеры содержащих элементов также задаются в единицах измерения em. Так ширина может соответствовать размеру шрифта текста. Например, если размер шрифта установлен 16 пикселов (заданный по умолчанию в большинстве браузеров), а ширина страницы установлена в 40 em, то ширина страницы будет 640 пикселов (40em×16px/em). Если пользователь увеличит размер шрифта текста до 20 пикселов, ширина страницы станет равной 800 пикселам.

Гибридные макеты

Макеты, которые используют сочетание значений в пикселах, процентах и единицах измерения em, иногда называют *гибридными*. Во многих случаях имеет смысл смешивать фиксированные и масштабируемые области контента. Например, на сайте может использоваться боковая панель, содержащая несколько рекламных баннеров, которые должны оставаться определенного размера. Вы можете задать ей определенную ширину в пикселах и позволить расположенной рядом колонке менять размер, чтобы заполнить оставшееся пространство. Вы, наверное, помните, что мы создавали подобную страницу в упражнении 15.4.

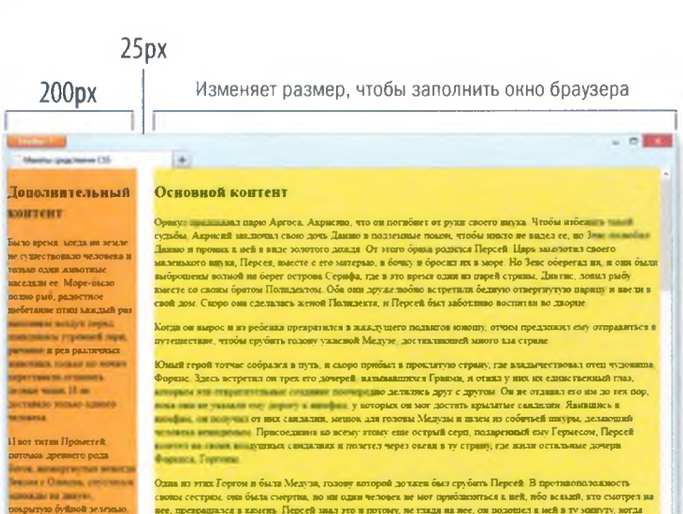
Рис. 16.6 иллюстрирует гибридный макет. Второстепенной колонке слева задана определенная ширина в пикселах, а для области основного контента установлено значение `auto`, и она заполняет оставшееся пространство в окне. Небольшое предупреждение: когда вы сочетаете единицы измерения (px, % и em), становится гораздо труднее рассчитать ширину страницы и элемента. Но при необходимости это вполне возможно.

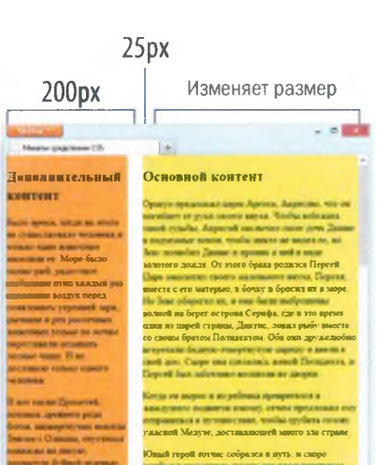
Какой макет следует использовать?

Как вы можете видеть, каждый подход к верстке имеет свои положительные и отрицательные стороны. И так как тенденции в использовании макетов появляются и исчезают, мы наблюдаем переход от фиксированных, подходящих для настольного компьютера, к жидким дизайнам, которые лучше приспособлены для работы на всех устройствах. Вы можете обнаружить, что жидкий макет лучше всего работает, если просматривать адаптивный сайт на экране с небольшой диагональю, а фиксированный макет предоставляет необходимый контроль, когда страница просматривается на мониторе с очень большой диагональю.

Возможно, ваш сайт — это сложное приложение для интранета, которое требует фиксированного дизайна и, скорее всего, будет использоваться только в браузерах настольных компьютеров и ноутбуков. Таким образом, нет «правильного» способа верстки веб-страниц, и важно рассмо-

треть все варианты, чтобы принять оптимальное решение при создании сайта, учитывая его контент, назначение и основное использование.






```
div#main {
    width: auto;
    position: absolute;
    top: 0;
    left: 225px;
    background: yellow; }

div#extras {
    width: 200px;
    position: absolute;
    top: 0;
    left: 0;
    background: orange; }
```

Рис. 16.6. Гибридный макет, сочетающий колонки с фиксированной шириной и автоматическим изменением размера

Шаблоны макетов страниц

Теперь мы разберемся, как создавать макеты с двумя и тремя колонками с помощью CSS. Примеры кода в этом разделе должны сильно облегчить вам задачу на начальном этапе, но они не являются универсальными ответами. Ваш контент может требовать поиска более замысловатых решений.

Этот раздел предоставляет шаблоны и приемы для разработки:

- Макетов с двумя и тремя колонками при помощи плавающих элементов
- Независимых от исходного кода макетов при помощи плавающих элементов и отрицательных значений полей
- Многоколоночных макетов посредством позиционирования

Использование шаблонов

Примеры страниц в этом разделе непривлекательны. Я убрала из них все лишнее, чтобы сделать структуру и стратегию как можно более понятной. Ниже приведены несколько замечаний относительно шаблонов и их использования.

Упрощенная разметка

Я включила в примеры только самую минимальную разметку и стили — достаточно, чтобы понять, как создается каждый макет. Все пра-

ПРИМЕЧАНИЕ

Файлы с кодом HTML и CSS для всех шаблонов этого раздела доступны в папке материалов для данной главы на диске, прилагающемся к книге.

вила стилей, не имеющие отношения к макету, были опущены ради экономии пространства и с целью сосредоточиться на том, что необходимо для перемещения элементов.

Оформление цветом

Я добавила контуры (см. врезку «Контуры средствами CSS») вокруг каждой колонки, чтобы вам были видны края плавающих или позиционированных элементов макета. Чтобы сделать связи более ясными, контурам добавлен цвет с помощью разметки и стилей.

Заголовки и нижние колонтитулы

Я включила заголовок и элемент нижнего колонтитула во многие из этих примеров, но любой из них или оба могут быть пропущены для минимального макета с двумя или тремя колонками.

Сделайте на свой вкус

Очевидно, что с текстом, фоном, полями, отступами и границами можно сделать гораздо больше, чтобы повысить привлекательность страниц. Как только наложите эти шаблоны на структуру, вы должны почувствовать себя свободно: изменяйте размеры и добавляйте ваши собственные стили.

Контуры средствами CSS

В примерах данного раздела я воспользовалась свойством **outline**, чтобы показать края плавающих и позиционированных элементов. Контуры выглядят так же, как границы, и используют тот же синтаксис, но, в отличие от границ, не учитываются при расчете ширины блока элемента. Они просто накладываются поверх, ни на что не влияя. Это делает контуры замечательным инструментом для проверки работы макета, потому что их можно включить и отключить, не изменяя все остальное.

Сокращенное свойство **outline**, как и граница, сочетает значения свойств ширины (**outline-width**), стиля (**outline-style**) и цвета (**outline-color**).

```
div#links { outline: 2px
dashed red; }
```

Многоколоночные макеты при помощи плавающих элементов

Основной инструмент для создания колонок на веб-страницах — плавающие элементы. Они не идеальны, но это лучшее, что у нас есть на момент написания книги. Более совершенные решения, появляющиеся на горизонте, описаны во врезке «Будущее CSS-макетов».

Преимущества плавающих элементов над абсолютным позиционированием в макетах заключаются в том, что они не позволяют элементам контента пересекаться, и с их помощью проще расположить колонтитул в нижней части страницы. Недостатком является их зависимость от порядка, в котором элементы указаны в исходном коде, хотя здесь существует обходной путь, задействующий отрицательные значения полей, как мы увидим далее в этой главе.

Следующие шаблоны демонстрируют общую стратегию подхода к макетам с двумя и тремя колонками при помощи плавающих элементов и должны сильно упростить вам задачу на начальном этапе создания ваших собственных макетов.

Жидкий макет с двумя колонками

В упражнении 15.3 предыдущей главы мы создали макет с двумя колонками, сделав один элемент плавающим и применив поля ко второму, чтобы освободить место для первого элемента. Вы можете сместить плаваю-

Будущее CSS-макетов

Каскадные таблицы стилей постоянно развиваются, чтобы удовлетворять потребности дизайнеров и разработчиков. Кроме того, сейчас создаются новые технологии создания макетов, которые могут освободить нас от создания и выравнивания колонок с помощью плавающих и позиционированных элементов.

Колонки

Наиболее простая схема улучшения макета, которая уже была реализована в браузерах — это поделить элемент на колонки. Можно указать количество колонок (**`column-count`**) или определенную их ширину (**`column-width`**), тогда колонки заданной ширины будут повторяться, пока не закончится пространство. Конечно, это далеко не все, и подробнее ознакомиться с вопросом вы можете на сайте консорциума Всемирной паутины по адресу **www.w3.org/TR/css3-multicol**. Колонки CSS в настоящее время поддерживаются браузерами Safari и Chrome с префиксом **`-webkit-`**, браузером Firefox с префиксом **`-moz-`**, а также Opera и Internet Explorer 10 (и выше) без префиксов.

Flexbox

Модель создания макета Flexible Box (известная сокращенно как Flexbox) предоставляет более простой способ организовать блоки элементов по отношению друг к другу. Например, вы можете выровнять дочерние элементы, созданные внутри родительского, выбрать, где появится дополнительное пространство, отцентрировать элементы по горизонтали или по вертикали, даже изменить порядок появления — и все это без плавающих элементов, смещения полей и сопутствующих им сложных расчетах. С Flexbox, в основном, можно просто сказать: «Сделайте этот элемент блоком и выровняйте его дочерний элемент по горизонтали и вертикали внутри него». В спецификации слишком много информации, чтобы даже касаться ее здесь, но я рекомендую прочитать вступительную статью Ричарда Шепарда по адресу **www.getincss.ru/2011/12/14/vvedenie-v-verstku-na-baze-css3-flexible-box-chast-1/**. Flexbox в настоящее время поддерживается актуальными версиями браузеров с использованием префиксов.

Системы сеток

Корпорация Microsoft приступила к применению коллекции свойств CSS, которые позволяют создавать для элемента сетку из строк и столбцов, а затем позиционировать другие элементы на этой сетке. На момент написания книги работа находилась на ранних стадиях, но за ней стоит следить. Подробнее об этом читайте на сайтах консорциума Всемирной паутины (**dev.w3.org/csswg/css3-grid-layout**) и Microsoft Developer Network (**msdn.microsoft.com/library/ie/hh673536.aspx#_CSSGrid**).

Регионы и исключения

Корпорация Adobe вносит свой вклад в канон CSS в виде модулей, которые дублируют некоторые функции ее продуктов для создания макетов страниц. Модуль CSS Regions позволяет контенту перетекать из одного элемента в другой, подобно тому, как текст перетекает из одного текстового блока в другой в программе InDesign. Модуль CSS Exclusions предоставляет способ заставить текст обтекать фигуры неправильной формы, какие можно увидеть в журнальной верстке. Более подробную информацию о развитии этих передовых функций можно найти на сайте консорциума Всемирной паутины (**dev.w3.org/csswg/css3-regions и dev.w3.org/csswg/css3-exclusions**), а также на сайте корпорации Adobe (**html.adobe.com**).

ПРИМЕЧАНИЕ

Во всех шаблонах макетов в этом разделе используются поля для поддержания расстояния между колонками. Если вы хотите добавить отступы и границы для плавающих элементов, не забудьте настроить значения ширины, чтобы их уместить. Кроме того, вы можете присвоить значение **box-model** свойству **box-sizing** (помните о вендорных префиксах), и вам не нужно будет производить расчеты для отступов и границ.

щие колонки влево или вправо в зависимости от порядка их следования в исходном коде и от того, в каком месте страницы вы хотите видеть каждую колонку. Начнем с очень простого макета с двумя колонками.

Стратегия

Задайте ширину обоих элементов колонок и переместите их влево. Примените свойство **clear** к нижнему колонтитулу, чтобы он остался в нижней части страницы. Основная структура и получившийся макет показаны на рис. 16.7.

Разметка

```
<div id="header">Титульные данные и заголовок</div>
<div id="main">Основная статья</div>
<div id="extras">Список ссылок и новостей</div>
<div id="footer">Информация о защите авторского права</div>
```

Стили

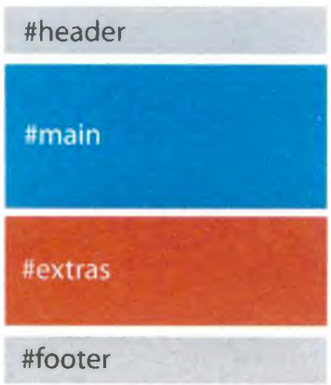
Помните о правиле для указания порядка значений полей: верхее, правое, нижнее, левое.

```
#main {
float: left;
width: 60%;
margin: 0 5%;
}
#extras {
float: left;
width: 25%;
margin: 0 5% 0 0;
}
#footer {
clear: left;
}
```

Замечания

Этот макет довольно прост, но поскольку он у нас первый, я укажу несколько моментов:

- Помните, чтобы примеры были как можно проще, я опустила стили заголовка, нижнего колонтитула и текста. Имейте в виду, что код файла из данного примера несколько сложнее и развернутее, чем показано в книге (однако ничего такого, до чего вы не смогли бы додуматься самостоятельно: цвет фона, отступы и тому подобное).
- Исходный документ был разделен на четыре элемента **div**, по одному для заголовка, контента, дополнительной информации и нижнего колонтитула. В разметке показан порядок, в котором они появляются в исходном коде.



Порядок элементов в исходном коде

Макет

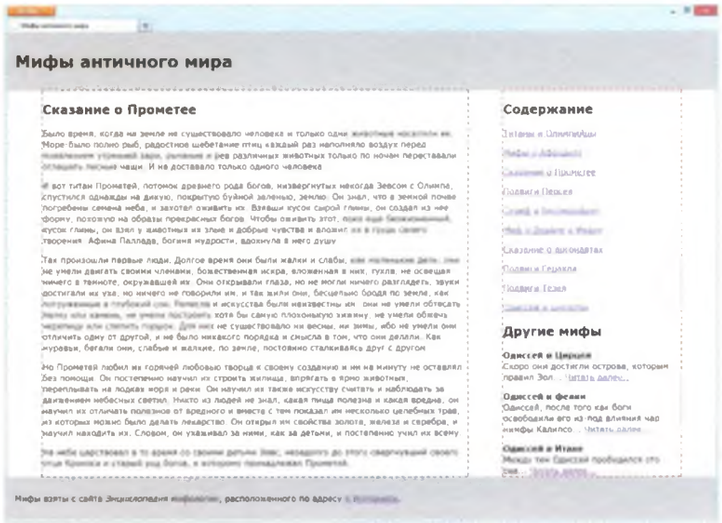


Рис. 16.7. Жидкий макет с двумя колонками

- Элементы **#main** и **#extras** были смещены влево. Поскольку это плавающие элементы, для каждого была указана ширина. Вы можете задать своим колонкам любую ширину.
- К элементу **#main** добавлены поля справа и слева шириной 5%.
- Элементу **#extras** необходимо поле только с правой стороны. Верхние поля заданы равными нулю, чтобы выровнять элементы по вертикали.
- Для элемента **#footer** установлен запрет обтекания (при помощи свойства **clear**), так что он располагается ниже колонки обтекаемого основного контента.

ПРИМЕЧАНИЕ

Для достижения того же самого эффекта можно переместить одну колонку влево, а другую — вправо.

Использование свойства max-width

Жидкие макеты замечательны тем, что они могут приспособиться к размеру экрана или окна браузера, в котором отображаются. Мы тратим много времени, раздумывая, как наши страницы выглядят на небольших экранах, но не стоит забывать, что существуют и мониторы с высоким разрешением, ширина которых равна или превышает 2000 пикселов. Пользователи могут не разворачивать окна браузера на весь экран, чтобы заполнить его целиком, но есть вероятность, что окно браузера будет настолько большим, что текст в жидких колонках станет трудно читать.

Вы можете положить конец этой проблеме с помощью свойства **max-width.** Примените его к элементу колонки, за который вы больше всего волнуетесь (например, к колонке **#main** в примере «жидкий макет с двумя колонками») или поместите всю страницу в элемент-контейнер и задайте ограничения ширины страницы целиком.

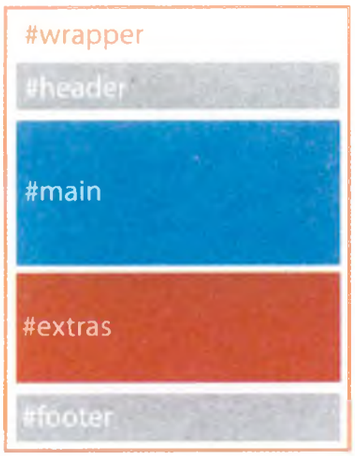
Аналогично доступно свойство **min-width**, если необходимо, чтобы ваша страница не выглядела слишком сжатой.

Фиксированный макет с двумя колонками

На этот раз создадим не жидкий макет, а фиксированной ширины.

Стратегия

Заключите контент в элемент `div` и установите для него определенную ширину в пикселах. Укажите значения в пикселах также и для плавающих элементов, но обтекание и метод запрета обтекания остаются те же. Получившийся макет показан на рис. 16.8.



Порядок элементов в исходном коде

Макет

Рис. 16.8. Фиксированный макет с двумя колонками, в котором используются плавающие элементы

Стили

```
#wrapper {
width: 960px;
}
#main {
float: left;
width: 650px;
margin: 0 20px;
}
#extras {
float: left;
width: 250px;
margin: 0 20px 0 0;
}
#footer {
clear: left;
}
```

Разметка

```
<div id="wrapper">
<div id="header">Титульные данные и заголовок</div>
<div id="main">Основная статья</div>
<div id="extras">Список ссылок и новостей</div>
<div id="footer">Информация о защите авторского права</div>
</div>
```

Замечания

- Весь контент заключен в элемент `#wrapper div`, которому задано распространенное значение ширины, равное 960 пикселам.
- Значения ширины и полей также были изменены на пиксельные, чтобы избежать превышения общей суммы в 960 пикселов. Если

такое случится, произойдет падение плавающего элемента. Имейте в виду, что, если вы добавляете отступы или границы, их общую ширину необходимо вычитать из значений ширины элементов, чтобы ширина всей страницы оставалась неизменной.

Фиксированный макет с двумя колонками, центрированный

На этом этапе очень легко центрировать макет фиксированной ширины.

Стратегия

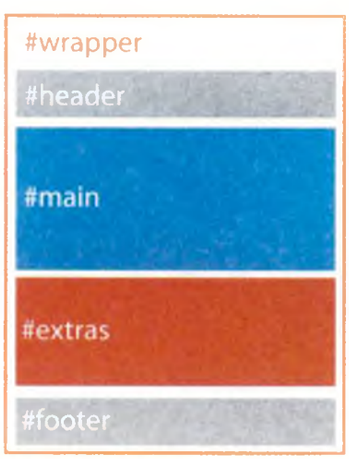
Задайте для правого и левого полей контейнера **#wrapper** значения **auto**, которые выровняют всю страницу по центру. Разметка точно такая же, как и в предыдущем примере. Нам нужно лишь добавить в стили определение поля. Готовый макет показан на рис. 16.9.

Стили

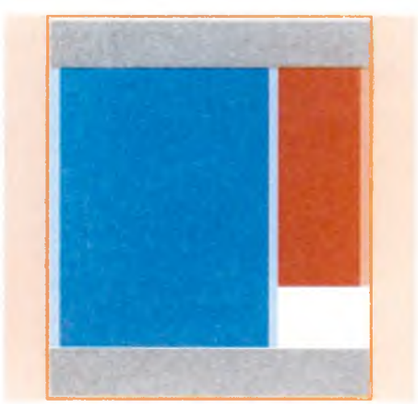
```
#wrapper {
width: 960px;
margin: 0 auto;
}
```

Замечания

- Настройка **auto** для левого и правого полей выравнивает элемент **#wrapper** по центру окна браузера.



Порядок элементов в исходном коде



Макет

Рис. 16.9. Фиксированный макет теперь центрирован в окне браузера

Заголовки и нижние колонтитулы во всю ширину страницы

Если вы хотите, чтобы элементы **#header** и **#footer** были растянуты во всю ширину окна браузера, но при этом вам нужно, чтобы контент между ними оставался фиксированной ширины и отцентрированным (рис. 16.10), измените разметку так, чтобы внутри элемента **#wrapper** находились только элементы **#main** и **#extras**. Все остальное остается так же, как в примере центрированного макета фиксированной ширины с двумя колонками.

```
<div id="header">Титульные данные и заголовок</div>
<div id="wrapper">
<div id="main">Основная статья</div>
<div id="extras">Список ссылок и новостей</div>
</div>
<div id="footer">Информация о защите авторского права</div>
```

Рис. 16.10. Заголовок и нижний колонтитул заполняют всю ширину окна браузера, но контент между ними остается фиксированной ширины

Жидкий макет с тремя колонками

Теперь мы возьмемся за макеты с тремя колонками, которые используют те же принципы, но требуют чуть больше хитрости. В этом примере мы переместим все плавающие элементы влево. Используя простые плавающие элементы, мы очень зависим от порядка, в котором они указаны в исходном коде.

Стратегия

Задайте ширину всех трех элементов колонок и переместите их влево. Запретите обтекание нижнего колонтитула с помощью свойства **clear**, чтобы он оставался в нижней части страницы. Основная структура и получившийся макет показаны на рис. 16.11.

Разметка

```
<div id="header">Титульные данные и заголовок</div>
<div id="links">Список ссылок и новостей</div>
<div id="main">Основная статья</div>
<div id="news">Новости</div>
<div id="footer">Информация о защите авторского права</div>
```

Стили

```
#links {
float: left;
width: 22.5%;
margin: 0 0 0 2.5%;
}
#main {
float: left;
width: 45%;
margin: 0 2.5%;
}
#news {
float: left;
width: 22.5%;
margin: 0 2.5% 0 0;
}
#footer {
clear: left;
}
```

Рис. 16.11. Макет с тремя колонками с помощью трех плавающих элементов

Замечания

- Разметка демонстрирует, что теперь у нас в документе присутствуют все пять элементов `div`: `#header`, `#links`, `#main`, `#news` и `#footer`.
- Если мы используем только простые плавающие элементы, и нам требуется, чтобы колонка основного контента отображалась в середине между колонками ссылок и новостей, тогда в исходном коде элемент `#main div` необходимо поместить между элементами `#links div` и `#news div`. (Мы освободимся от порядка исходного кода в примере «Колонки в любом порядке с помощью отрицательных значений полей», который следует далее.)
- Задайте трем колонкам фиксированную ширину и переместите их влево. Следует удостовериться, что сумма значений свойств `width` и `margin` не превышает 100%.

Колонки в любом порядке с помощью отрицательных значений полей

Когда макеты на основе плавающих элементов начали приобретать все большую популярность, многие дизайнеры подумали: «Есть ли способ создать макет с тремя плавающими колонками, независимый от порядка элементов в исходном коде?» Оказывается, что ответ: «Да!» Хитрость заключается в использовании математики и отрицательных значений полей. Этот метод описан в статье Алекса Робинсона, доступной по адресу positioniseverything.net/articles/onetruelayout/.

УПРАЖНЕНИЕ 16.1. ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Мы уже видели много примеров макетов с двумя и тремя колонками, использующих плавающие элементы, как для жидких макетов, так и для макетов с фиксированной шириной. Я думаю, что самое время вам применить некоторые из этих методов, используя в качестве отправной точки макет с тремя колонками, который мы только что рассмотрели. Файл *mountolympus-ex1.html* для этого упражнения находится на диске, прилагаемом к книге. Получившиеся в результате стили перечислены в приложении А. Стили контура добавлены, но вы можете скрыть их, пометив как комментарии (заключите их в символы `/*` и `*/`, чтобы скрыть), если хотите просмотреть макет без них.

Прежде всего, переместите боковые колонки так, чтобы ссылки оказались справа, а новости слева. Вам не нужно менять разметку всего в нескольких значений стилей. (Подсказка: направление обтекания.) Убедитесь, что вы скорректировали правое и левое поля боковых колонок и запретили обтекание элемента `#footer`.

Далее преобразуйте этот жидкий макет в центрированный дизайн с фиксированной шириной. На этот раз вам нужно будет добавить разметку (если потребуется помощь, обратитесь к макету фиксированной ширины с двумя колонками). Получившаяся страница показана на рис. 16.12.

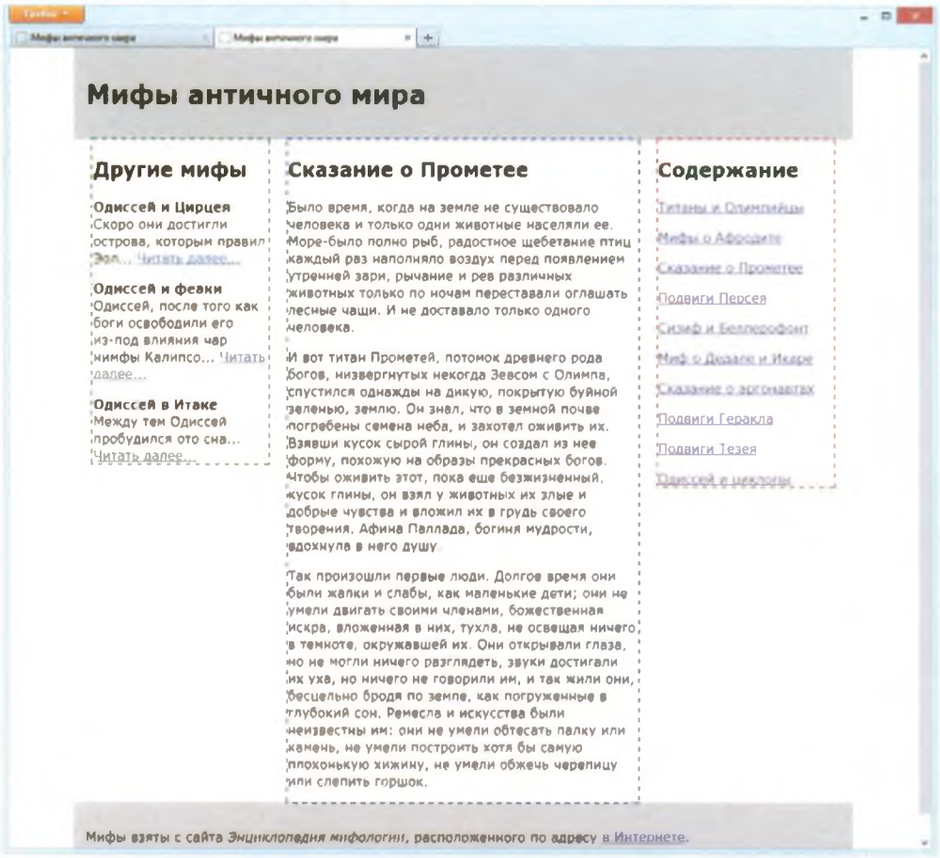


Рис. 16.12. *Готовый макет фиксированной ширины, в котором колонки поменялись местами*

Стратегия

Примените значения ширины и свойство **float** ко всем трем элементам колонок, а также используйте отрицательные поля, чтобы «перетащить» правую колонку через страницу к левому краю. Основная структура и получившийся макет показаны на рис. 16.13. Обратите внимание, что, хотя элемент **#main** в исходном коде указан первым, эта колонка вторая по счету. Кроме того, колонка элемента **#links div** (последняя в исходном коде) занимает первую позицию слева. Это пример фиксированного макета, но вы можете выполнить то же самое с жидким макетом, используя процентные значения.

Разметка

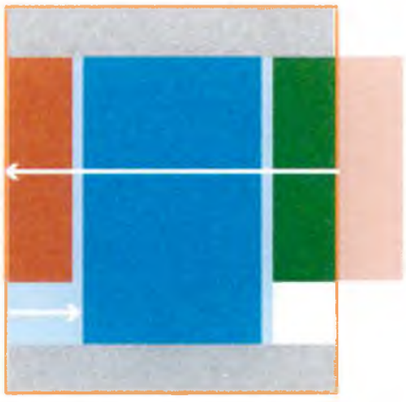
```
<div id="wrapper">
<div id="header">Титульные данные и заголовок</div>
<div id="main">Основная статья</div>
<div id="news">Новости</div>
<div id="links">Список ссылок и новостей</div>
<div id="footer">Информация о защите авторского права</div>
</div>
```

Стили

```
#wrapper {
width: 960px;
margin: 0 auto;
}
#main {
float: left;
width: 520px;
margin-top: 0;
margin-left: 220px;
margin-right: 20px;
}
#news {
float: left;
width: 200px;
margin: 0;
}
#links {
```

```
float: left;
width: 200px;
margin-top: 0;
margin-left: -960px;
}
#footer {
clear: left;
}
```

Порядок элементов в исходном коде



Макет

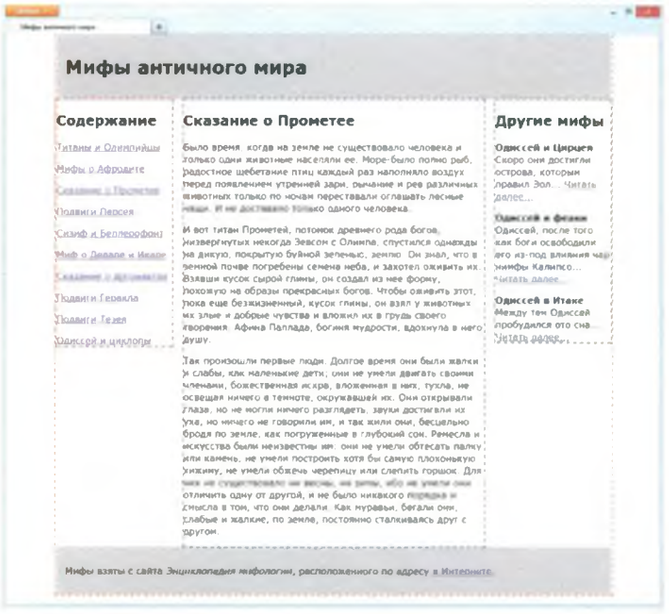


Рис. 16.13. Фиксированный макет с тремя колонками с помощью трех плавающих элементов. Он похож на предыдущий пример, но отличается тем, что порядок колонок не такой, как в исходном коде

Замечания

Пример требует более подробных объяснений, поэтому мы рассмотрим его создание пошагово. В разметке элемент **#main** указан первым, видимо, потому, что это наиболее важный контент, а элемент **#links** идет последним. Вся страница «обернута» в элемент **#wrapper**, чтобы можно было задать ей определенную ширину (960px). В макете, однако, порядок столбцов слева направо такой: **#links** (200px в ширину), **#main** (520px в ширину), затем **#news** (200px в ширину). Расстояние между колонками в этом макете составляет 20 пикселов.

Первым шагом будет перемещение контента элемента **#main** в середину путем добавления левого поля, которое сдвигает его достаточно далеко, чтобы предоставить место для левой колонки (200px) и пространства между ними (20px). Таким образом, **margin-left: 220px**. Попутно также добавим к элементу **#main** 20px правого поля, чтобы освободить

место справа от него. На рис. 16.14 показано, как выглядит макет после применения стилей к элементу **#main**.

Теперь перетащите влево контент, который вы хотите переместить в левую колонку (**#links** в данном случае) с помощью *отрицательного* значения поля. Фокус в том, чтобы выяснить, как далеко влево его необходимо сместить. Если взглянуть на рис. 16.14, можно увидеть призрачную копию элемента **#links**, показывающую, где он мог бы находиться, если бы элемент **#wrapper** был достаточно широк. Я считаю, полезно посмотреть на макет с такой стороны, потому что это объясняет, что нам необходимо перетащить элемент **#links** влево на расстояние ширины всех блоков элемента, стоящих перед ним в исходном коде.

*Рис. 16.14. Макет после применения полей к среднему элементу колонки (**#main**). Затененный блок справа показывает, где мог бы находиться элемент **#links**, если бы его не пришлось переместить под элемент **#news***

В этом примере ширина блока элемента **#main** составляет 520px + 220px левого поля и 20px правого поля, что в сумме дает 760 пикселов. Общая ширина элемента **#news** составляет 200px (поля не применяются). Это означает, что элемент **#links div** необходимо сместить влево в общей сложности на 960 пикселов, чтобы он оказался помещенным в левую колонку (**margin-left: -960px;**). При применении отрицательных значений полей, элемент **#links** встает на место, и мы имеем окончательный макет, показанный на рис. 16.13. Метод отрицательных полей

ПРЕДУПРЕЖДЕНИЕ

При самостоятельной работе не забудьте включить также отступы и границы в расчет общей ширины окна элемента, если вы не применяете модель определения размера блока **border-box**.

УПРАЖНЕНИЕ 16.2. ИСПОЛЬЗОВАНИЕ ОТРИЦАТЕЛЬНЫХ ЗНАЧЕНИЙ ПОЛЕЙ

Теперь, когда вы знаете стратегию, потренируйтесь в написании стилей, перемещающих контент элемента **#news** в левую колонку, а элемента **#links** — в правую. В основе этого упражнения лежит тот же порядок элементов в исходном коде, что и в предыдущем примере. Отметим, однако, что значения ширины колонок изменились. Как и прежде, добавьте расстояние между колонками, равное 20 пикселам.

Если вы хотите поэкспериментировать в текстовом редакторе с настоящими файлами, возьмите файл *mountolympusex2.html*, который находится на диске, прилагающемся к книге, в папке материалов для этой главы. Финальный код стилей приведен в приложении А. Запомните, ключевой момент — перемещение элемента **#news** влево, используя отрицательное поле, на суммарную ширину элементов, предшествующих ему в исходном коде.

```
#main {
float: left;
width: 400px;
/* напишите определения поля ниже */

}
#news {
float: left;
width: 300px;
/* напишите определения поля ниже */

}
#links {
float: left;
width: 220px;
/* напишите определения поля ниже */

}
```

Получившийся макет должен выглядеть так, как показано на рис. 16.15.

Рис. 16.15. Финальный вариант макета с тремя колонками из упражнения 16.2

можно использовать для размещения любого количества колонок в любом порядке. В упражнении 16.2 вы получите возможность изменить порядок колонок, чтобы элемент **#news** оказался слева.

Позиционированные макеты

Плавающие элементы мы рассмотрели. Другой способ создания колонок в макете — с помощью абсолютного позиционирования. Ранее, в упражнении 15.4, мы создали гибридный макет с двумя колонками, имеющий позиционированную колонку фиксированной ширины.

В этом разделе мы применим позиционирование, чтобы организовать три колонки на жидких страницах и страницах с фиксированной шириной.

Обратите внимание, что в обоих примерах я опустила элемент `#footer`. Я поступила так по нескольким причинам. Во-первых, когда вы позиционируете все элементы макета (что мы и сделаем в этих примерах), они больше не участвуют в создании макета, а значит, ничто не удерживает нижний колонтитул в нижней части страницы. Он поднимется на самый верх. Существуют решения этой проблемы с помощью JavaScript, но они выходят за рамки данной главы.

Однако мы позиционируем только две боковые колонки и позволим средней колонке основного контента остаться в потоке, чтобы удержать нижний колонтитул в нижней части страницы. Это, конечно, возможно, но если одна из боковых колонок станет длиннее средней, они будут перекрывать контент нижнего колонтитула. Прыгающие нижние колонтитулы и потенциальные наложения — это неаккуратно, вот почему я решила опустить нижний колонтитул в этом случае (по этой же причине плавающие элементы — более популярная техника создания макетов).

Позиционированный жидкий макет с тремя колонками

В этом макете применяются процентные значения для создания трех гибких колонок. Получившийся в результате макет показан на рис. 16.16.

Порядок элементов в исходном коде Макет

Рис. 16.16. Три позиционированных жидких колонки

Стили

```
#content {
position: relative;
margin: 0;
}
#main {
width: 50%;
position: absolute;
top: 0;
left: 25%;
margin: 0;
}
#news {
width: 20%;
position: absolute;
top: 0;
left: 2.5%;
margin: 0;
}
#links {
width: 20%;
position: absolute;
top: 0;
right: 2.5%;
margin: 0;
}
```

Стратегия

Заключите три элемента **div** контента (**#main**, **#news**, **#links**) в **div** (**#content**), который послужит контейнером для трех позиционированных колонок. Затем задайте элементам колонок ширину и позиционируйте их в контейнере **#content**.

Разметка

```
<div id="header">Титульные данные и заголовок</div>
<div id="content">
<div id="main">Основная статья</div>
<div id="news">Новости</div>
<div id="links">Список ссылок и новостей</div>
</div>
```

Замечания

Я думаю, что стили для этого макета покажутся вам довольно простыми.

- Я создала содержащий блок **#content** для размещения колонок, так как нам нужно, чтобы они всегда начинались под элементом **#header**. Если бы мы позиционировали их относительно окна браузера (начальный содержащий блок), они могли оказаться в неправильной позиции, когда высота заголовка изменится, например в результате изменения размера шрифта элемента **h1**. Сделайте элемент **#content div** содержащим блоком, применив определение **position: relative**.
- Элементу **#main div** задана ширина 50%, а абсолютное позиционирование используется, чтобы поместить его в верхней части элемента **#content div** и на 25% от левого края. Так, ширина левой колонки будет 20%, плюс поля 2,5% слева и справа от нее.
- Элемент **#news div** позиционируется в верхней части элемента **#content div** и на 2,5% от левого края (**top: 0; left: 2.5%;**).
- Элемент **#links div** позиционируется в верхней части элемента **#content div** (**top: 0; right: 2.5%;**) и на 2,5% от правого края. Не нужно вычислять позицию по левому краю... просто поместите **#links div** справа! Обратите внимание, что мы могли бы позиционировать колонки **#news** и **#links** каждую вплотную к своему краю и затем применить отступы, чтобы добавить немного пространства. Обычно задачи макета можно решить несколькими способами.
- Чтобы все сделать правильно, необходимо убедиться, что размеры свойств **width** и **margin** в сумме не превышают 100%. Не забудьте учесть в расчетах также отступы и границы.

Позиционированный фиксированный макет с тремя колонками

Если вы предпочитаете контролировать позиционированный макет до последнего пиксела, это очень легко сделать, как в данном примере (рис. 16.17). Он отличается от предыдущего тем, что вся страница заключена в элемент **#wrapper**, который можно фиксировать, центрировать и применить для измерений пиксельные значения. Для экономии места я покажу вам применяемые стили. Стратегия позиционирования та же самая.

Рис. 16.17. Три позиционированных колонки центрированной страницы с фиксированной шириной

Стили

```
#wrapper {
width: 960px;
margin: 0 auto;
}
#content {
margin: 0;
position: relative;
}
#main {
width: 520px;
```

```
position: absolute;
top: 0;
left: 220px;
margin: 0;
}
#news {
width: 200px;
position: absolute;
top: 0;
left: 0;
margin: 0;
}
#links {
width: 200px;
position: absolute;
top: 0;
right: 0;
margin: 0;
}
```

Фоновый цвет колонок сверху донизу

Добавление цвета к колонкам — эффективный способ выделить впоследствии разделы информации и привнести цвет на страницу. Но если вы взглянете на пунктирные границы всех примеров, рассмотренных выше, то увидите, что элемент колонки часто заканчивается задолго до конца страницы. А значит, придется исхитриться, если мы хотим добавить фоновый цвет сверху донизу.

К сожалению, нет предусмотренного способа задать высоту элемента в 100% от высоты страницы, и хотя существуют обходные пути посредством JavaScript и зарождающейся спецификации Flexbox, которые создают элементы колонок во всю высоту, но эти способы выходят за рамки данной главы.

Однако не волнуйтесь. Существует надежное решение, известное как *псевдоколонки*, которое будет работать с любым шаблоном фиксированной ширины, описанным здесь. Прием построен на том, что вы создаете рисунок, в котором цвета колонок расположены на отведенных им местах, и применяете его при помощи размещающегося мозаикой изображения на фоне страницы или содержащего элемента (такого как **#wrapper** в примере). Метод псевдоколонок был впервые представлен Дэном Седерхольмом в 2004 году в его статье для сайта A List Apart и книге «Web Standards Solutions».

Рассмотрим, как это работает. Серая колонка на рис. 16.18 является результатом горизонтального изображения с полосками цвета, которые соответствуют ширине колонок. Когда для изображения задано вертикальное мозаичное размещение на заднем плане, результатом являются вертикальные полосы, поверх которых может быть расположен многоколоночный макет. Этот метод работает, только когда ширина колонки или страницы задана в конкретном пиксельном измерении. Чуть ниже мы поговорим о фонах жидких колонок.

На рис. 16.18 представлен фиксированный центрированный макет с двумя колонками, который мы создали ранее (см. рис. 16.9). В этот раз к нему применяется изображение two_column.png, с вертикальным мозаичным размещением в элементе **#wrapper**.

```
#wrapper {
width: 960px;
margin: 0 auto;
background-image: url(two_column.gif);
background-repeat: repeat-y;
}
```

two_columns.png

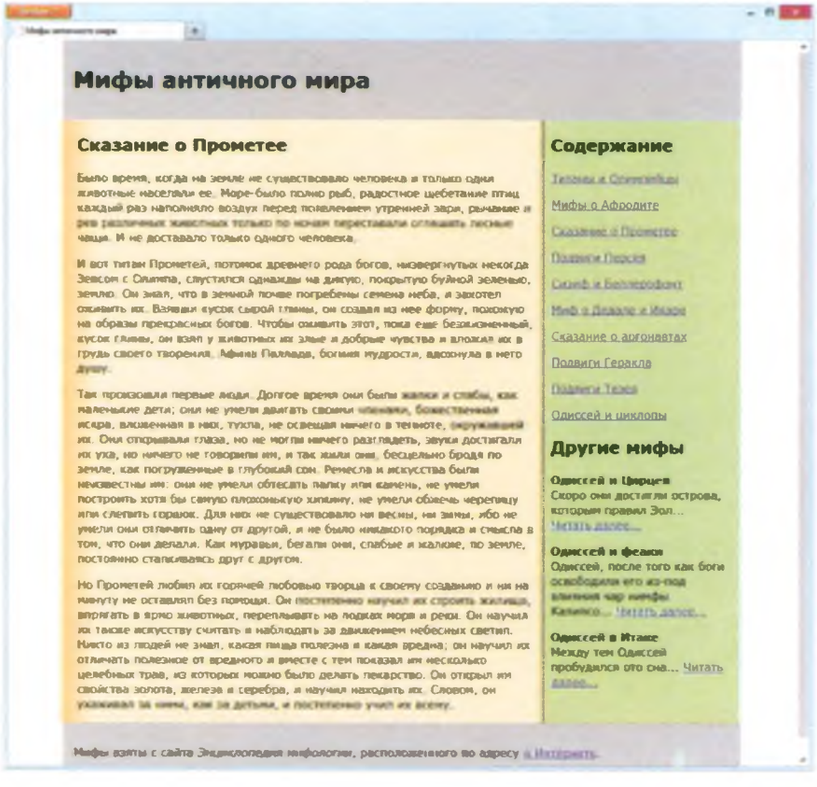


Рис. 16.18. Размещающееся мозаикой изображение используется для создания цветных колонок

ПРИМЕЧАНИЕ

Если у вашего макета нет нижнего колонтитула, то чтобы удержать содержащий элемент открытым после того, как колонкам задано обтекание, примените свойство **overflow: hidden;** к элементу **#wrapper**, чтобы растянуть его вокруг плавающих элементов.

Псевдоколонки для жидких макетов

Теперь, когда вы понимаете основной метод, возможно, вы задумываетесь, как заставить его работать для колонок с меняющейся шириной. Секрет — *очень, очень* широкий фоновый рисунок и свойство **background-position**. Нам может быть неизвестна точная ширина колонок жидкого макета, но мы *знаем* позицию, где они разделяются. В качестве примера давайте используем жидкий шаблон с двумя колонками из рис. 16.7. Разделение на колонки происходит на 67,5% от левого края (5% левое поле + 60% ширина колонки элемента **#main** + 2,5%, что составляет половину от 5% пространства между полями). В программе Photoshop (или другом графическом редакторе на ваш выбор) создайте горизонтальное изображение, которое будет шире экрана любого монитора — 3000 пикселов должно хватить. Поскольку изображение должно быть всего несколько пикселов в высоту и, скорее всего, будет содержать малое количество цветов, размер файла должен оказаться небольшим. Создавая цвета, убедитесь, что они соответствуют пропорциям ваших колонок. В нашем примере фон левой колонки должен заполнять 67,5% ширины рисунка (67,5% × 3,000 = 2025 пикселов).

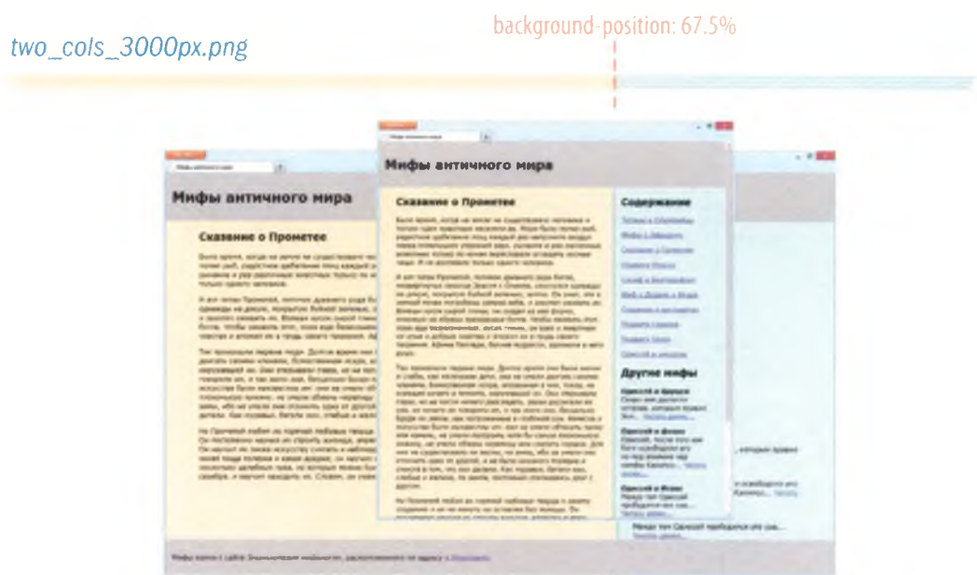


Рис. 16.19. *Фоновое изображение привязано к позиции между двумя колонками, поэтому, когда окно браузера увеличивается или уменьшается, оно всегда находится в нужном месте. Графический файл достаточно широк, чтобы изображения хватало на обе колонки, даже в самых широких окнах браузеров*

СОВЕТ

Существует изящный трюк, если вам нужна всего лишь тонкая линия между колонками без заливки фона. Задайте ближним границам двух соседних колонок одинаковую ширину, а затем используйте отрицательные поля, равные ширине границы, чтобы заставить границы наложиться друг на друга. Таким образом, какая бы из колонок ни оказалась длиннее, выглядит это так, будто на странице продолжается одна и та же линия.

Примените широкое изображение в качестве фонового узора к элементу **body** и с помощью свойства **background-position** совместите позицию, где изменяется цвет рисунка (67,5%) с позицией, где колонки разделяются на странице (также 67,5%). Таким образом, разрыв колонок в изображении всегда будет в центре пространства между ними. И у вас получились псевдоколонки, которые растягиваются и сжимаются вместе с шириной колонок.

```
body {
background-image: url(two_cols_3000px.png);
background-repeat: repeat-y;
background-position: 67.5%;
}
```

Три псевдоколонки

Так, сработало на двух колонках, но что насчет трех? Это возможно благодаря методу, введенному Дагом Боуманом. По сути, процесс такой же, как только что наблюдаемый нами: пропорционально позиционировать очень широкое фоновое изображение в контейнере **div**. Но для трех колонок вы позиционируете два фоновых изображения. Одно обеспечивает полосу цвета для левой колонки, а оставшаяся правая часть является прозрачной. Второе изображение обеспечивает цвет для правой колонки, а его левая часть остается прозрачной (рис. 16.20). Цвет фона страницы проявляется сквозь прозрачные области и обеспечивает цвет для средней колонки.

В разметке необходимы два контейнера. Я назвала их в этом примере **#wrapper** и **#inner**:

```
<div id="wrapper">
<div id="inner">
<div id="main"></div>
<div id="news"></div>
<div id="links"></div>
</div> <!-- end inner div -->
</div> <!-- end wrapper div -->
```

Изображение для левой колонки помещается в элемент **#wrapper**, его позиция — в точке между левой и средней колонками (26,25% для примера на рис. 16.20). Изображение для правой колонки заключается в элемент **#inner**, расположенный между средней и правой колонками

ПРИМЕЧАНИЕ

Изображения с прозрачными областями в форматах GIF и PNG обсуждаются в главе 19.

ПРИМЕЧАНИЕ

Такой же эффект может быть достигнут при помещении нескольких фоновых изображений в элемент **#wrapper**, что избавляет от необходимости дополнительной разметки. Задайте одному изображению мозаичное отображение по вертикали по левой стороне, а другому — такое же по правой. Изображения должны быть достаточно широкими, чтобы простираться от точки деления на колонки далеко за пределы края окна браузера. Недостаток этого метода в том, что он не будет работать в браузере Internet Explorer версии 8 и ниже.

(73,75%). Когда размер окна браузера изменяется, фоновые изображения остаются на месте в соответствующих позициях между колонками, а цвет фона заполняет пространство между ними.

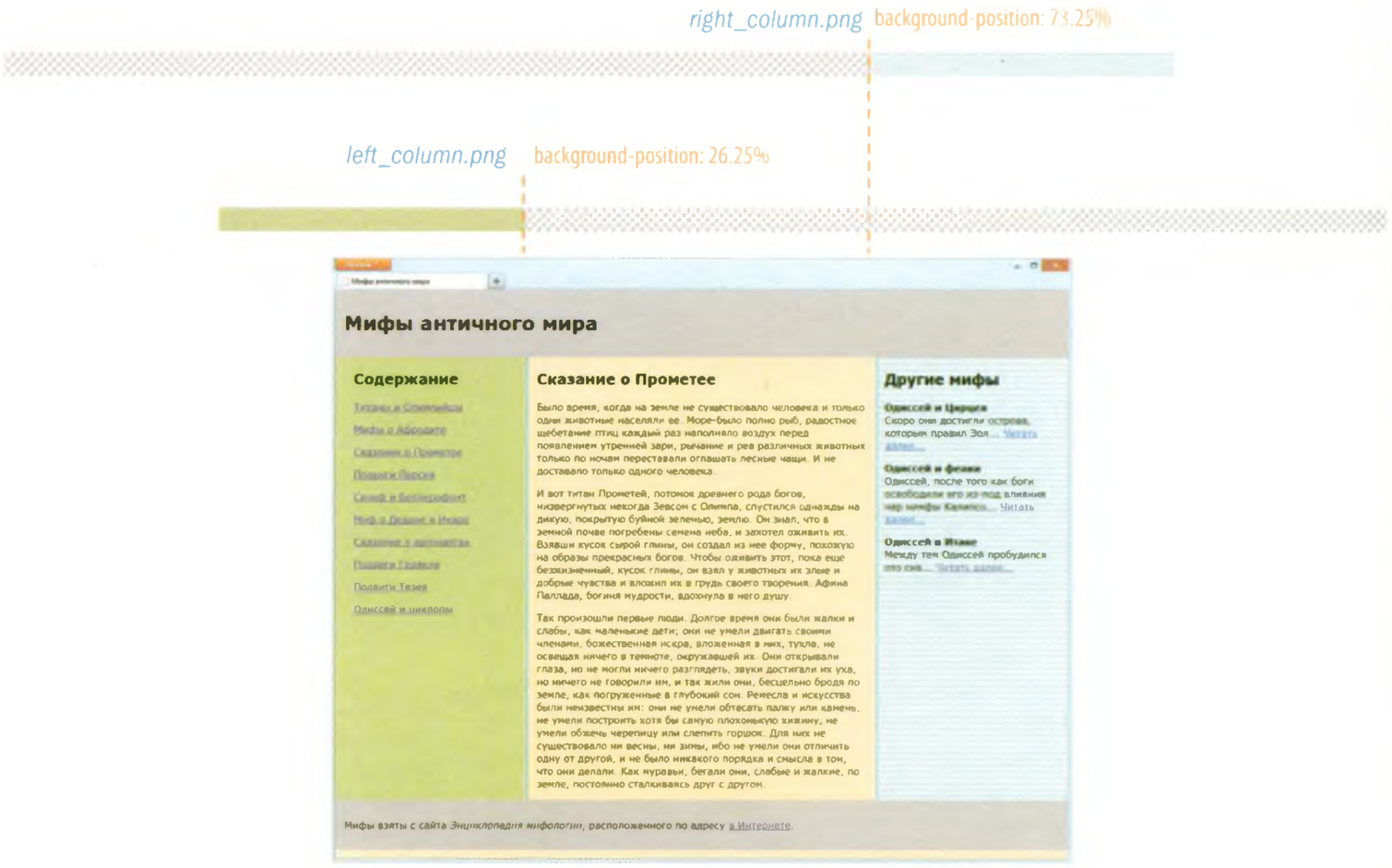


Рис. 16.20. Псевдоколонки для жидкого макета с тремя колонками

ГЛАВА 17

ПЕРЕХОДЫ, ПРЕОБРАЗОВАНИЯ И АНИМАЦИЯ

Мы видели, как CSS используются для создания визуальных эффектов, таких как скругленные углы, градиенты и тени, которые раньше приходилось делать с помощью изображений. В этой главе рассмотрим некоторые свойства CSS3 для создания анимированных интерактивных эффектов, которые раньше были возможны только с применением технологий Flash или JavaScript. Мы начнем с CSS-переходов — отличного способа плавно модифицировать стили. Затем обсудим CSS-преобразования, используемые для перемещения, масштабирования, вращения и наклона элементов, и посмотрим, как можно анимировать их с помощью переходов. Я завершу главу кратким введением в трехмерные преобразования и CSS-анимации, о которых важно знать. Однако стоит учесть, что эта тема слишком велика, чтобы основательно рассмотреть ее в рамкой данной книги.

Проблема в том, что анимация и временные эффекты не работают на бумаге, поэтому я не могу показать их прямо здесь. Однако нашлось другое хорошее решение: вы сможете взаимодействовать с большинством интерактивных примеров из этой главы, открыв в браузере файл *Примеры/глава-17/Примеры/Примеры.htm* с диска, прилагающегося к книге.

В этой главе

- Создание плавных переходов
- Перемещение, вращение и масштабирование элементов
- Комбинирование переходов и преобразований
- Несколько слов о трехмерных преобразованиях
- Несколько слов об анимации по ключевым кадрам

CSS-переходы

Представьте себе ссылку в меню навигации, цвет фона которой при наведении на нее указателя мыши меняется с синего на красный. Фон синий... указатель мыши перемещается на нее... Красный! Она переходит из одного состояния в другое мгновенно, без промежуточных вариантов. Теперь представьте себе, что вы навели указатель мыши, и фон постепенно меняется с синего на красный, проходя по пути через несколько оттенков фиолетового. Это плавно. И когда вы уберете указатель мыши, фон постепенно вернется обратно к синему.

Вот что делают переходы CSS. Веб-дизайнеры называют подобные эффекты *переходами*. Они сглаживают во времени изменения (которые

ПРИМЕЧАНИЕ

Вы можете ознакомиться с модулем переходов CSS по адресу **www.w3.org/TR/css3-transitions/**.

в противном случае были бы резкими) от одного состояния до другого, заполняя кадры между ними. Если применять переходы осторожно и умеренно, можно добавить интерфейсу утонченности и лоска и сделать его использование более приятным. CSS-переходы изначально были разработаны командой Webkit для браузера Safari, но сейчас они включены в рабочий проект консорциума Всемирной паутины. На момент написания книги набор свойств переходов был еще очень неполным, и спецификация, скорее всего, будет изменена. Поэтому для всех браузеров, поддерживающих их, необходимы соответствующие префиксы.

Переходы хорошо поддерживаются (с префиксами) в операционных системах iOS, Android, а также мобильных браузерах Opera.

Старые версии браузеров не поддерживают переходы — Internet Explorer версии 9 и ниже, Firefox 3.6 и ниже, и Opera 10.1 и ниже. Но если вы используете переходы для прогрессивного улучшения, вас не должно чересчур сильно беспокоить то, что пользователи этих браузеров не увидят эффекты. Для них будет происходить просто мгновенный переход от синего к красному.

Основные сведения о переходах

При создании переходов необходимо принять несколько решений, каждое из которых связано со свойством CSS:

- Какое свойство CSS изменять (**`transition-property`**)
- Сколько времени переход должен продолжаться (**`transition-duration`**)
- Каким образом переход ускоряется (**`transition-timing-function`**)
- Должна ли быть пауза перед его началом (**`transition-delay`**).

Кроме того, инициировать переход должно некое событие. Изменение одного из состояний, такого как: **`:hover`**, **`:focus`** или **`:active`**, станет хорошим инициирующим событием, и именно их мы будем использовать для примеров в этой главе. Вы можете применить сценарии JavaScript для изменения элемента (например, для добавления атрибута **`class`**) и так же использовать его в качестве события, инициирующего переход.

ПРЕДУПРЕЖДЕНИЕ

Модуль переходов CSS сам находится в «переходном» состоянии. Таким он был на момент написания книги, но вам обязательно следует проверить последние разработки на сайте консорциума Всемирной паутины.

Объединим все это в простой пример. Ниже показана кнопка-ссылка, превращающаяся из синей в красную (рис. 17.1). В разметке нет ничего особенного. Я добавила имя класса, чтобы можно было уточнить, к каким ссылкам применяются переходы.

Свойства перехода применяются к элементу **`a`** в обычном состоянии. Вы увидите их в наборе других определений для **`a.smooth`** таких, как **`padding`** и **`background-color`**. Это позволит повторно применить их для изменения других состояний в документе. Я изменила цвет фона

ссылки на красный, определив свойство **background-color** для состояния **:hover** (и для состояния **:focus**, на случай, если пользователь перемещается по ссылкам с помощью клавиатуры).

Рис. 17.1. Цвет фона данной ссылки постепенно меняется с синего на красный, когда применяется переход

Разметка

```
<a href="" class="smooth">Вкусный соус</a>
```

Стили

```
a.smooth {
display: block;
text-decoration:none;
text-align: center;
padding: 1em 2em;
width: 10em;
border-radius: 1.5em;
color: #fff;
background-color: mediumblue;
transition-property: background-color;
transition-duration: 0.3s;
}
a.smooth:hover, a.smooth:focus {
background-color: red;
}
```

Табл. 17.1. Свойства CSS, к которым можно добавить анимацию

Фоны
background-color
background-position
Границы и контуры
border-bottom-color
border-bottom-width
border-left-color
border-left-width
border-right-color
border-right-width
border-top-color
border-top-width
border-spacing
outline-color
outline-offset
outline-width
Цвет и непрозрачность
color
opacity
visibility
Шрифт и текст
font-size
font-weight
letter-spacing
line-height
text-indent
text-shadow
word-spacing
vertical-align
Размеры блока элемента
height
width
max-height
max-width
min-height
min-width
margin-bottom
margin-left
margin-right
margin-top
padding-bottom
padding-left
padding-right
padding-top
crop
Положение
top
right
bottom
left
z-index
clip
Преобразования
(не указаны в спецификации на момент написания книги, но поддерживаются)
transform
transform-origin

Определение свойства

transition-property

Новое в CSS3

Принимаемые значения: *имя свойства* | **all** | **none**

Значение по умолчанию: **all**

Применение: ко всем элементам, к псевдоэлементам **:before** и **:after**

Наследование: нет

Свойство **transition-property** определяет свойство CSS, к которому надлежит применить плавный переход. В нашем примере это цвет фона. Вы также можете изменить цвет переднего плана, границы, размеры, атрибуты, относящиеся к шрифту и тексту, и многое другое. Полный перечень (на момент написания книги) приведен в табл. 17.1. Скорее всего, в этот список будут добавляться еще свойства по мере того, как браузеры будут их применять, поэтому проверьте обновления спецификации.

Продолжительность перехода

transition-duration

Новое в CSS3

Принимаемые значения: *значение времени*

Значение по умолчанию: **0s**

Применение: ко всем элементам, к псевдоэлементам **:before** и **:after**

Наследование: нет

Свойство **transition-duration** задает количество времени, которое потребуется для воспроизведения анимации в секундах (**s**) или миллисекундах (**ms**). Я выбрала значение 0,3 секунды, которого как раз достаточно. Конечно, не существует правила длительности, но в своих поисках я выяснила, что распространенное время переходов для элементов пользовательского интерфейса составляет около 0,2 секунды. Поэкспериментируйте, чтобы найти продолжительность, подходящую именно для вашего веб-проекта.

Функции тайминга

transition-timing-function

Новое в CSS3

Принимаемые значения: **ease** | **linear** | **ease-in** | **ease-out** | **ease-in-out** | **step-start** | **step-end** | **steps** | **cubic-bezier(#,#,#,#)**

Значение по умолчанию: **ease**

Применение: ко всем элементам, к псевдоэлементам **:before** и **:after**

Наследование: нет

Свойство и продолжительность составляют основу перехода, но вы можете добавить и дополнительные настройки. Существует несколько вариантов поведения перехода во времени. Например, он может начинаться быстро, а затем замедляться, начинаться медленно и ускоряться или не менять скорость на протяжении всего действия, и это только некоторые варианты. Они — своего рода «стиль» перехода, но в спецификации используется термин «рассчет по времени» или *тайминг*. Я могу присвоить свойству **transition-timing-function** значение **ease-in-out**, чтобы сделать изменение цвета ссылки от синего к красному более плавным. Честно говоря, при очень небольшой длительности перехода, различия едва заметны.

```
a.smooth {
...
transition-property: background-color;
transition-duration: 0.3s;
transition-timing-function: ease-in-out;
}
```

Свойство **transition-timing-function** принимает одно из следующих значений, обозначаемых зарезервированными словами.

ease

Начинается медленно, заметно ускоряется, затем замедляется в конце. Это значение задано по умолчанию и превосходно подходит для большинства коротких переходов.

linear

Скорость остается неизменной от начала и до конца перехода.

ease-in

Начинается медленно, затем ускоряется.

ease-out

Начинается быстро, затем замедляется.

ease-in-out

Начинается медленно, ускоряется, затем замедляется снова в самом конце. Похоже на значение **ease**, но с менее выраженным ускорением в середине.

cubic-bezier(#,#,#,#)

Функция для определения кривой Безье, которая описывает ускорение перехода. Вы можете прочитать, как ее применять, в спецификации (**www.w3.org/TR/css3-transitions/#transitiontiming-function-property**).

steps(#, start|end)

Делит переходы на несколько шагов, как определено ступенчатой функцией. Первое значение — количество шагов, а зарезервированные слова **start** и **end** определяют, происходит ли изменение состояния в начале (**start**) или в конце каждого шага. Подробнее читайте в спецификации.

step-start

Производит смену состояния за один шаг, в начале периода (так же, как **steps(1,start)**). Результат — резкое изменение состояния, такое же, как без применения перехода.

step-end

Производит смену состояния за один шаг, в конце периода (так же, как **steps(1,end)**).

Я не могу продемонстрировать различные настройки переходов на странице, но привела несколько примеров (рис. 17.2). Ширина каждого помеченного элемента (белый с оранжевой границей) меняется в течение четырех секунд при наведении указателя мыши на синий блок. Они все обретают максимальную ширину одновременно, но разными способами.

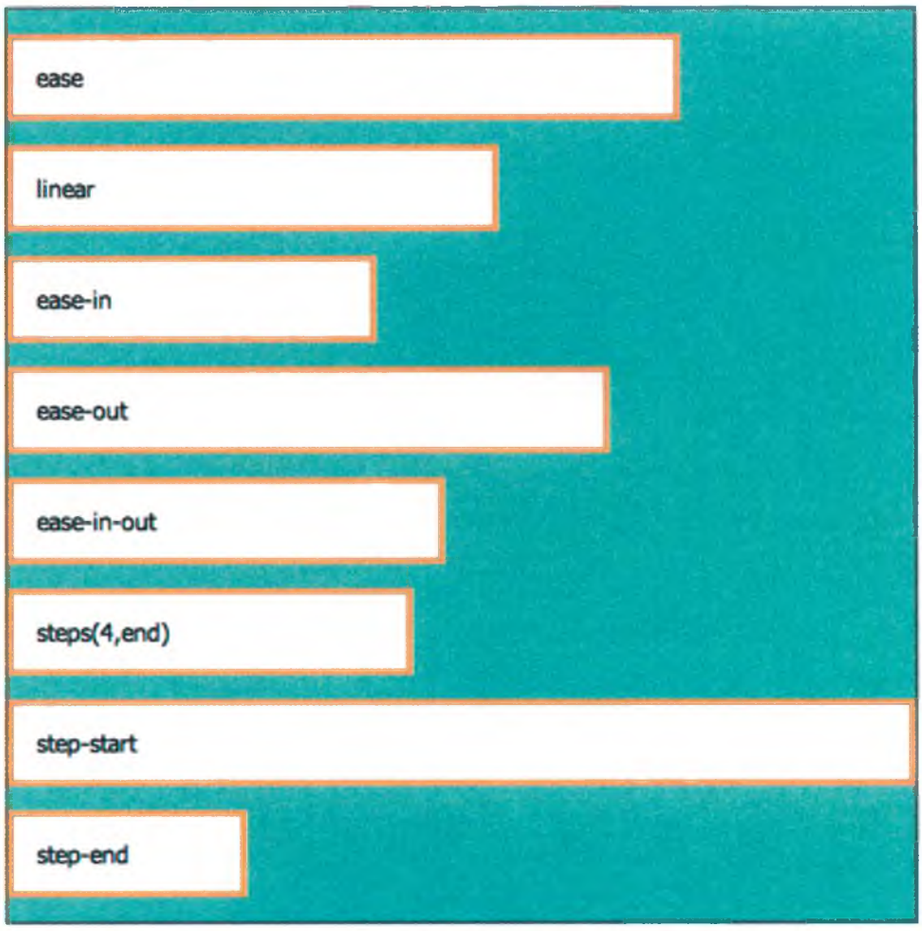


Рис. 17.2. *В этом примере, демонстрирующем работу свойства* ***transition-timing-function****, элементы достигают максимальной ширины одновременно, но способы выполнения различны*

Настройка задержки

transition-delay

Новое в CSS3

Принимаемые значения: *значение времени*

Значение по умолчанию: 0s

Применение: ко всем элементам, к псевдоэлементам **:before** и **:after**

Наследование: нет

Свойство **transition-delay** задерживает начало анимации на указанное время. В следующем примере переход цвета фона начнется через 0,2 секунды после наведения указателя на ссылку.

```
a.smooth {
…
transition-property: background-color;
transition-duration: 0.3s;
transition-timing-function: ease-in-out;
transition-delay: 0.2s;
}
```

По желанию вы можете заставить кнопку исчезнуть (**opacity: 0;**) на две секунды (**transition-delay: 2s;**), после того, как пользователь коснется ее или наведет курсор (**:active**), как показано в следующем примере и на рис. 17.3.

```
a.smooth {
…
transition-property: opacity; -
transition-duration: .05s;
transition-timing-function: ease-out;
transition-delay: 2s;
}
a.smooth:hover, a.smooth:focus {
background-color: red;
}
a.smooth:active {
opacity: 0;
}
```

Хочу заметить, что в примерах я использовала свойства без префиксов, чтобы вам было проще следить за кодом, но помните, что необходимо добавлять вендорные префиксы для всех браузеров, если вы применяете переходы на своих страницах.

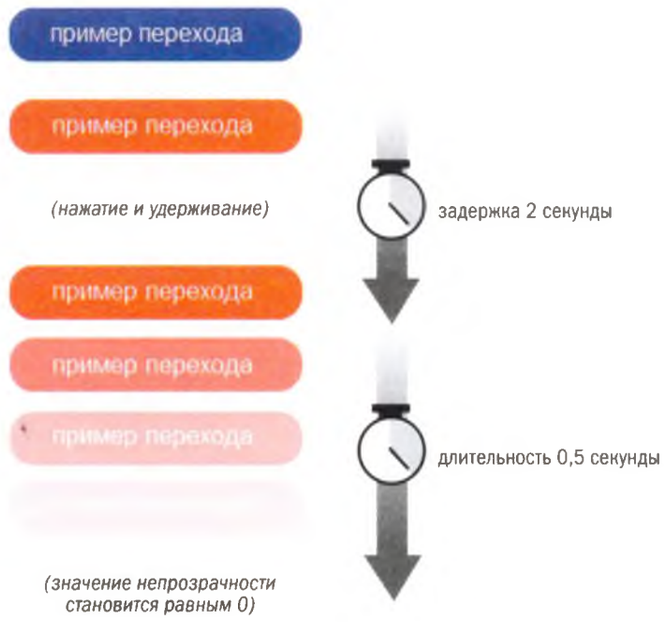


Рис. 17.3. Свойство **`transition-delay`** *запускает эффект анимации (в данном случае — исчезновение кнопки при помощи настройки непрозрачности) через две секунды*

Версию кода без префиксов всегда указывайте последней. Ниже показано, как переход цвета ссылки от синего к красному, над которым мы работали, выглядел бы в реальности:

```
a.smooth {
...
-webkit-transition-property: background-color;
-webkit-transition-duration: 0.3s;
-webkit-transition-timing-function: ease-in-out;
-webkit-transition-delay: 0.2s;
-moz-transition-property: background-color;
-moz-transition-duration: 0.3s;
-moz-transition-timing-function: ease-in-out;
-moz-transition-delay: 0.2s;
-o-transition-property: background-color;
-o-transition-duration: 0.3s;
-o-transition-timing-function: ease-in-out;
-o-transition-delay: 0.2s;
-ms-transition-property: background-color;
-ms-transition-duration: 0.3s;
-ms-transition-timing-function: ease-in-out;
-ms-transition-delay: 0.2s;
```

```
transition-property: background-color;
transition-duration: 0.3s;
transition-timing-function: ease-in-out;
transition-delay: 0.2s;
}
```

Это дополнительная работа, но так придется делать только в обозримом будущем, пока все старые браузеры не выйдут из употребления, а спецификация не станет стабильной и применяемой везде одинаково. К счастью, существует сокращенное свойство, позволяющее уменьшить объем кода.

Сокращенное свойство перехода

Благодаря здравомыслию разработчиков спецификации CSS3 мы получили сокращенное свойство **transition** для объединения всех изученных свойств в одном определении. Вы видели подобное с сокращенным свойством **border**. Его синтаксис следующий:

```
transition: свойство продолжительность тайминг задержка;
```

Значения для каждого из свойств **transition-*** перечисляются с разделением пробелами. Если вы предоставите только одно значение времени, будет считаться, что это продолжительность. Если вы предоставляете два значения времени, убедитесь, что продолжительность указана первой.

В примере со ссылкой, меняющей цвет с синего на красный, примененные выше четыре свойства перехода могут быть объединены в одну строку:

```
a.smooth {
…
transition: background-color 0.3s ease-in-out 0.2s;
}
```

А полная версия с префиксами сократится с 20 строк до 5.

```
a.smooth {
…
-webkit-transition: background-color 0.3s ease-in-out 0.2s;
-moz-transition: background-color 0.3s ease-in-out 0.2s;
-o-transition: background-color 0.3s ease-in-out 0.2s;
-ms-transition: background-color 0.3s ease-in-out 0.2s;
transition: background-color 0.3s ease-in-out 0.2s;
}
```

Определенно, так код выглядит лучше.

Применение множественных переходов

До сих пор мы меняли только одно свойство за раз, но можно задать переход для нескольких свойств одновременно. Давайте вернемся к примеру ссылки. В этот раз, помимо изменения синего цвета на красный, я хочу немного увеличить интервалы между буквами. Также я хочу, чтобы цвет текста поменялся на черный, но более медленно, чем другие эффекты анимации. На рис. 17.4 предпринята попытка показать эти переходы на печатной странице.

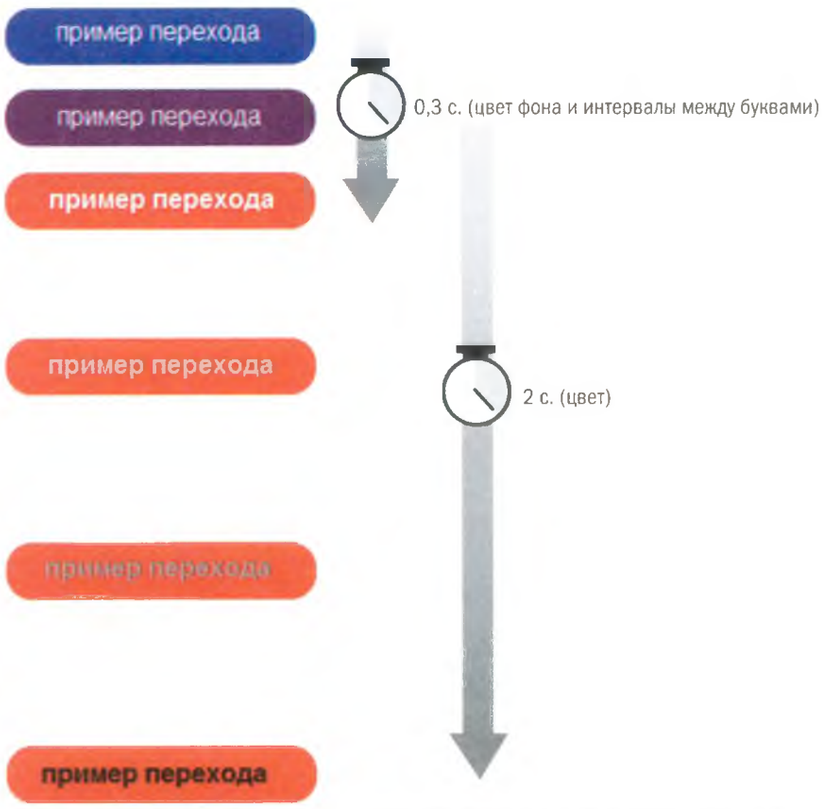


Рис. 17.4. Свойства `color`, `background-color` *и* `letter-spacing` *меняются с разной скоростью*

Один из способов достичь желаемого эффекта — перечислить все значения для каждого свойства через запятую, как показано в примере.

```
a.smooth {
...
transition-property: background-color, color, letter-spacing;
transition-duration: 0.3s, 2s, 0.3s;
transition-timing-function: ease-out, ease-in, ease-out;
}
a:hover, a:focus {
background-color: red;
letter-spacing: 3px;
color: black;
}
```

Значения указаны в зависимости от их позиций в списке. Например переход свойства **color** (второй в списке) длится две секунды (**2s**) и использует функцию тайминга **ease-in**. Если в одном списке значений меньше, чем в остальных, браузер повторяет их, начиная самого первого. В предыдущем примере, если бы я опустила третье значение (**0.3s**) свойства **transition-duration**, браузер вернулся бы к началу списка и использовал первое значение (**0.3s**) для свойства **letter-spacing**. В этом случае эффект был бы тот же.

Подобным образом вы можете выстроить значения сокращенного свойства **transition**. Тот же самый набор стилей, который мы сейчас рассматривали, можно записать и так:

```
a.smooth {
…
transition: background-color 0.3s ease-out,
color 2s ease-in,
letter-spacing 0.3s ease-out;
}
```

Похоже, это хороший способ, особенно если учесть, что к каждому определению перехода нужно добавить четыре варианта префиксов для браузеров.

Переход на все случаи жизни

Но что если вам просто нужно сделать все изменения состояний чуть более плавными, независимо от того, какое свойство может измениться? В тех случаях, когда вы хотите применить ко всем переходам, которые могут встретиться в элементе, одинаковую продолжительность, функцию тайминга и задержку, используйте значение **all** свойства **transition-property**. В следующем примере я указала, что любое свойство элемента **a.smooth**, которое может меняться, должно иметь продолжительность 0,2 секунды и анимироваться с помощью функции **ease-in-out**.

```
a.smooth {
…
-webkit-transition: all 0.2s ease-in-out;
-moz-transition: all 0.2s ease-in-out;
-o-transition: all 0.2s ease-in-out;
-ms-transition: all 0.2s ease-in-out;
transition: all 0.2s ease-in-out;
}
```

Часто для изменения пользовательского интерфейса достаточно короткого, ненавязчивого перехода во всех случаях, поэтому значение **all** вам пригодится. На этом урок, посвященный CSS-переходам, завершается. Теперь попытайтесь реализовать полученные знания на практике — в упражнении 17.1.

УПРАЖНЕНИЕ 17.1. ИСПЫТАНИЕ ПЕРЕХОДОВ

В этом упражнении мы настроим вид ссылки меню при наведении и нажатом состояниях (рис. 17.5) с помощью анимированных переходов. Я сверстала для вас начальный документ *exercise1.html* в папке материалов для этой главы. Готовый код показан в приложении А.

1. Сначала взглянем на стили, которые уже применяются. Список был преобразован в горизонтальное меню с помощью плавающих элементов. Элементу **a** задано отображение в качестве блочного, подчеркивание отключено, применены размеры и отступы, а также установлены цвет, фоновый цвет и граница. Я использовала свойство **box-shadow**, чтобы придать ссылкам такой вид, будто они выступают из страницы.
2. Теперь определим стили для состояний при наведении и в фокусе. Сделайте так, чтобы цвет фона менялся на золотистый (**#fdca00**), а цвет границы — на оранжевый (**#fda700**), когда пользователь наводит указатель мыши на ссылку или выбирает ее с помощью клавиатуры.

```
a:hover, a:focus {
background-color: #fdca00;
border-color: #fda700;
}
```

3. Когда пользователь щелкает по ссылке или нажимает клавишу **Enter** (**:active**), сместите изображение кнопки вниз на три пиксела, как будто бы она нажата. Для этого задайте относительное позиционирование элемента **a**, а затем измените значение свойства **top** активного состояния. Ссылка сместится на три пиксела от верхнего края (другими словами, вниз).

```
a {
…
position: relative;
}
a:active {
top: 3px;
}
```

4. По логике, если кнопка нажата, тень станет меньше, поэтому мы также сократим расстояние свойства **box-shadow**.

```
a:active {
top: 3px;
box-shadow: 0 1px 2px rgba(0,0,0,.5);
}
```

5. Сохраните файл и протестируйте его в браузере. Ссылки должны становиться желтыми и перемещаться вниз при щелчке по ним мышью или выборе посредством клавиатуры. Теперь мы можем улучшить эффект, добавив несколько плавных переходов.

6. Задайте переходам цвета фона и границы значение тайминга **ease** с продолжительностью 0,2 секунды и посмотрите, как изменится впечатление от использования меню. Я употребила сокращенное свойство **transition**, чтобы код оставался простым. Также я сначала применила заданное по умолчанию значение **ease** функции тайминга, поэтому мы можем исключить его из таблицы стилей.

В этом примере я укажу все префиксы браузеров, но если вы используете Chrome или Safari, можете просто добавить значение **-webkit-**, чтобы сэкономить время на верстке кода. В следующих примерах я буду указывать только стандартные свойства без префиксов для экономии места (но версии с префиксами будут подразумеваться).

```
a {
-webkit-transition: background-color 0.2s, border-color
0.2s;
-moz-transition: background-color 0.2s, border-color
0.2s;
-o-transition: background-color 0.2s, border-color
0.2s;
-ms-transition: background-color 0.2s, border-color
0.2s;
transition: background-color 0.2s, border-color 0.2s;
}
```

7. Сохраните документ, откройте его в браузере и попробуйте навести указатель мыши на ссылки (см. примечание). Смотрится лучше? Теперь опробуем несколько других значений продолжительности. Посмотрим, можно ли заметить разницу, если продолжительность составляет 0,1s. Теперь попробуйте целую секунду (**1s**). Я думаю, вы увидите, что одна секунда — это на удивление долго. Попробуйте задать несколько секунд и применить различные значения свойства **timing-function** (добавьте их после значений продолжительности). Заметно, в чем разница? У вас есть предпочтения? Когда закончите экспериментировать, верните значение продолжительности **0.2s**.

ПРИМЕЧАНИЕ

Если для этого упражнения вы пользуетесь устройством с сенсорным экраном, то не заметите данного эффекта, потому что на сенсорных экранах отсутствует состояние «при наведении».

8. Теперь рассмотрим, что произойдет, если мы добавим переход к нисходящему движению ссылки при нажатии на нее или касании. Переход добавляется к обоим свойствам **top** и **box-shadow**, так как они должны выполняться в тандеме. Давайте начнем с продолжительности **0.2s**, как и в других случаях.

```
a {
transition: background-color 0.2s, border-color 0.2s,
top .2s, box-shadow 0.2s;
}
```

Сохраните файл, откройте его в браузере и попробуйте щелкнуть мышью по ссылке. Этот переход действительно меняет впечатление от использования меню, не так ли? Кажется, что кнопки труднее нажимать. Попробуйте увеличить продолжительность.

Стало еще труднее? Мне интересно наблюдать, какой эффект оказывает продолжительность на впечатление от работы с пользовательским интерфейсом. Важно все понять правильно и не делать так, чтобы интерфейс казался медленным. Я бы сказала, что с очень коротким переходом, например 0,1 секунды (или вообще отсутствием перехода) эти кнопки станут казаться быстро реагирующими и чувствительными.

9. Если вы пришли к выводу, что при увеличении продолжительности перехода меню становится неудобно использовать, попробуйте добавить короткую полусекундную задержку к свойствам **top** и **box-shadow**.

```
a {
transition: background-color 0.2s, border-color 0.2s,
top 0.2s 0.5s, box-shadow 0.2s 0.5s;
}
```

Верно подобранное время решает все!

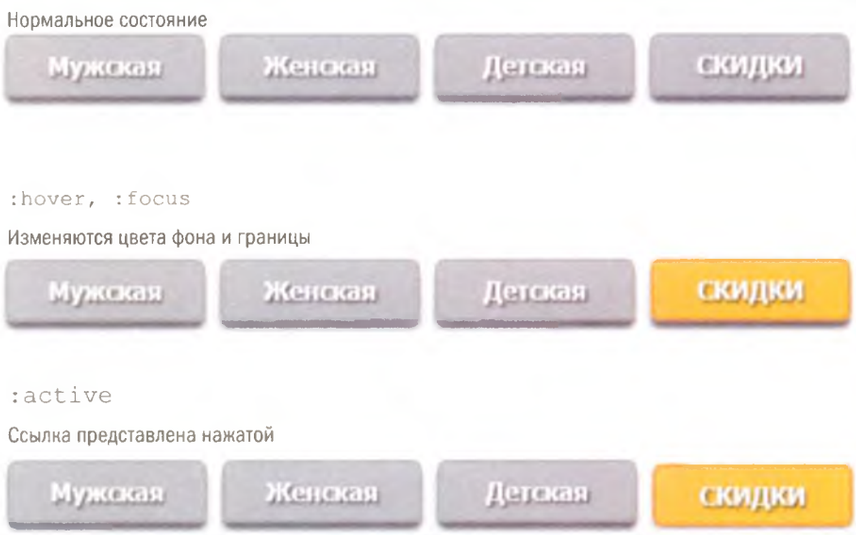


Рис. 17.5. В данном упражнении мы создадим переходы между состояниями ссылки

CSS-преобразования

transform

Новое в CSS3

Принимаемые значения: *функция(и) преобразования* | **none**

Значение по умолчанию: **none**

Применение: к преобразуемым элементам (см. врезку)

Наследование: нет

Модуль преобразований CSS3 предоставляет верстальщикам веб-страниц способ вращать, перемещать, изменять размер и наклонять HTM-элементы в двух- и трехмерном пространствах. Эта глава, однако, уделяет вни-

мание более простым двухмерным переменным, потому что они имеют большее практическое применение. Преобразования поддерживаются всеми основными версиями браузеров с добавлением префиксов.

Вы можете применить преобразование к нормальному состоянию элемента, и при загрузке страницы он появится в обновленном виде. Однако убедитесь, что страницу по-прежнему можно использовать в браузерах, которые не поддерживают подобные действия. Часто преобразования добавляют только тогда, когда пользователи взаимодействуют с элементом через ролловер-эффект или событие JavaScript (на «уровне опыта», как говорит специалист по CSS Дэн Седерхольм в своей книге «CSS3 для веб-дизайнеров» (Манн, Иванов и Фербер, 2013)). В любом случае, преобразование является хорошим вариантом прогрессивного улучшения — если пользователь браузера Internet Explorer 8 увидит элемент расположенным прямо, а не под забавным углом, ничего страшного не произойдет.

На рис. 17.6 продемонстрировано представление четырех типов двухмерных преобразований: поворота, перемещения, масштабирования и наклона. Пунктирной линией показано исходное положение элемента.

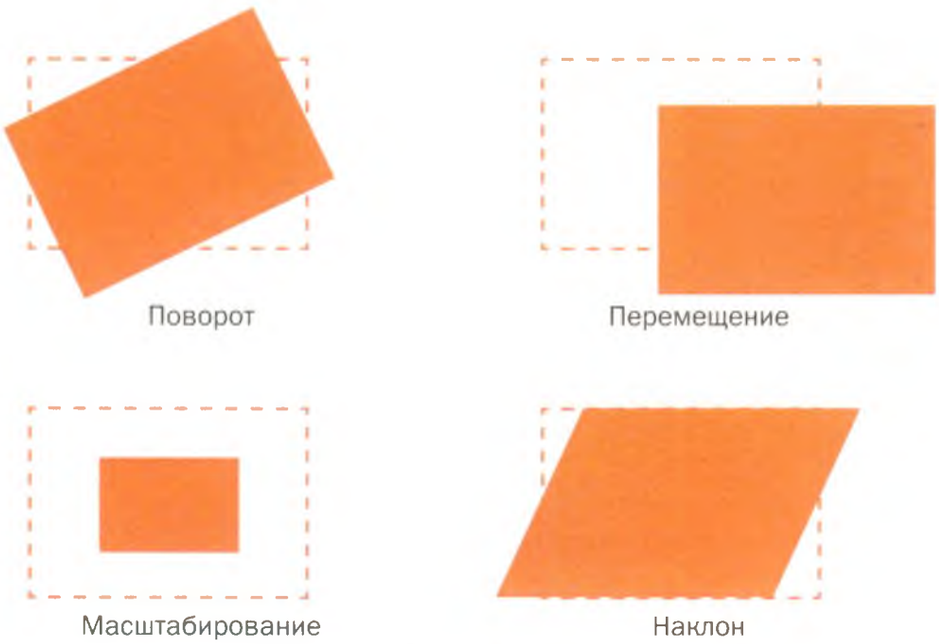


Рис. 17.6. *Четыре типа преобразований: поворот, перемещение, масштабирование и наклон*

Когда элемент преобразуется, его блок сохраняет свое исходное положение и влияет на макет вокруг него таким же образом, как и на пространство, оставленное после относительно позиционированного элемента. Как будто преобразование волшебным образом выбирает пикселы отображенного элемента, обрабатывает их, а потом отображает поверх страницы. Так что если вы переместите элемент с помощью преобразования, вы передвинете только его изображение. Оно не влияет на окружающие элементы макета. Давайте рассмотрим все функции преобразования по очереди, начиная с поворота.

ПРИМЕЧАНИЕ

Модули двухмерных, трехмерных и SVG-преобразований были в объединены в один черновик «CSS Transforms» в 2012 году. Спецификация доступна по адресу: **www.w3.org/TR/css3-transforms/**.

ПРИМЕЧАНИЕ

На самом деле в спецификации CSS есть пять функций двухмерного преобразования. Пятая — матрица — позволяет создавать собственные преобразования, используя шесть значений и серьезные знания тригонометрии. Если вам интересно, определение матрицы преобразования опубликовано по адресу: **www.w3.org/TR/SVG/coords.html#InterfaceSVGMatrix.**

Преобразование угла (поворот)

Если вы хотите отобразить элемент под некоторым углом, примените функцию преобразования **rotate**. Ее значение представляет собой угол, указанный в положительных или отрицательных градусах. Изображение на рис. 17.7 было повернуто на –10 градусов (350 градусов) с помощью правила стилей, приведенного ниже. Затененное изображение показывает исходное положение элемента.

```
img {
width: 300px;
height: 400px;
transform: rotate(-10deg);
}
```

Рис. 17.7. Поворот элемента ***img*** *с помощью свойства* **transform: rotate()**

Обратите внимание, что изображение вращается вокруг своей центральной точки. По умолчанию все преобразования происходят вокруг нее. Но вы можете легко это изменить, добавив свойство **transform-origin**.

transform-origin

Новое в CSS3

Принимаемые значения: *проценты* | *значение длины* | **left** | **center** | **right** | **top** | **bottom**

Значение по умолчанию: 50% 50%

Применение: к преобразуемым элементам

Наследование: нет

Значением свойства **transform-origin** являются два зарезервированных слова, два значения длины или два процентных значения. Первое указывает смещение по горизонтали, а второе — по вертикали. Когда

Преобразуемые элементы

Вы можете применить свойство **transform** к следующим типам элементов:

- HTML-элементы с заменяемым контентом, такие как **img**, **canvas**, элементы формы и встроенные мультимедийные элементы.
- Элементы, у которых задано свойство **display: block**.
- Элементы, у которых задано свойство **display: inline-block**.
- Элементы, у которых задано свойство **display: inline-table** (или любые типы элементов табличного отображения).

дано только одно значение, оно будет использовано для обоих измерений. Если бы мы хотели повернуть изображение соснового леса вокруг точки, расположенной по центру его верхнего края, мы могли бы записать это любым из следующих способов:

```
transform-origin: center top;
transform-origin: 50%, 0%;
transform-origin: 150px, 0;
```

Все изображения на рис. 17.8 были повернуты на 25 градусов, но относительно разных исходных точек.

Рис. 17.8. Изменение точки, вокруг которой вращается изображение с помощью свойства **`transform-origin`**

Продемонстрировать исходную точку с помощью функции поворота несложно, но имейте в виду, что вы можете установить ее для любой функции преобразования.

Преобразование позиции (перемещение)

Также с помощью свойства **`transform`** вы можете задать элементу новое положение на странице, применив одну из трех функций **`translate`**, как показано в примерах на рис. 17.9. Функция **`translateX`** можете перемещать элемент по горизонтальной оси; **`translateY`** используется для перемещения по вертикальной оси, а **`translate`** — как сокращение для объединения значений по осям X и Y **(translate(translateX, translateY)**.

```
transform: translateX(50px);
transform: translateY(25px);
transform: translate(50px, 25px);
```

Допустимо предоставить величину смещения в любой из единиц измерения CSS или в процентах. Последние рассчитываются от ширины *ограничительной рамки*, то есть расстояния от края одной границы до

края противоположной (что случайно совпадает со способом расчета процентов в SVG, откуда и были взяты преобразования). Как показано на рис. 17.9, можно задать положительные или отрицательные значения. Если вы предоставляете только одно значение для сокращенной функции `translate`, оно будет воспринято как значение функции `translateX`, а значение `translateY` будет равно нулю. Таким образом, функция `translate(20px)` эквивалентна применению двух функций `translateX(20px)` и `translateY(0)`.

Рис. 17.9. Перемещение элемента с помощью функции `translate`

Преобразование размера (масштабирование)

Изменяйте размер элемента, с помощью одной из трех функций масштабирования: `scaleX` (по горизонтали), `scaleY` (по вертикали), а также сокращенной функции `scale`. Значение — число без указания единицы измерения, которое обозначает соотношение размеров. Этот пример увеличивает размер изображения на 150% от его начальной ширины:

```
a img {
transform: scaleX(1.5);
}
```

В сокращенной функции `scale` перечисляются значения для `scaleX` и `scaleY`. В этом примере ширина элемента увеличивается в два раза, а высота уменьшается наполовину по сравнению с оригиналом.

```
a img {
transform: scale(2, .5);
}
```

Однако в отличие от свойства `translate`, если вы предоставите только одно значение масштаба, оно будет использовано в качестве коэффициента масштабирования для обоих направлений. Так указать `scale(2)` — это то же самое, что применить `scaleX(2)` и `scaleY(2)` (рис. 17.10).

transform: scale(1.25); transform: scale(.75); transform: scale(1.5, .5);

Рис. 17.10. Результаты всех попыток масштабирования

Скашивание (наклон)

Причудливый набор свойств наклона (**skewX**, **skewY** и сокращенное свойство **skew**) изменяет угол горизонтальной или вертикальной (или обеих) осей на заданное количество градусов. Как и при перемещении, если вы укажете только одно значение, оно будет применено к свойству **skewX**, а значение **skewY** будет равно нулю. Лучший способ понять, как работает наклон — взглянуть на несколько примеров (рис. 17.11).

```
a img {
transform: skewX(15deg);
}
a img {
transform: skewY(30deg);
}
a img {
transform: skew(15deg, 30deg);
}
```

transform: skewX(15deg); transform: skewY(30deg); transform: skew(15deg, 30deg);

Рис. 17.11. Наклон элемента с помощью функции **skew**

Применение множественных преобразований

Конечно, можно применить к одному элементу несколько преобразований. Просто перечислите функции и их значения, разделенные пробелами, например:

```
transform: функция(значение) функция(значение);
```

В примере, показанном на рис. 17.12, я увеличила изображение леса, немного повернула его и переместила вниз и вправо — эти действия совершаются, если навести указатель мыши или сфокусироваться на изображении.

```
img:hover, img:focus {
transform: scale(1.5) rotate(-5deg) translate(50px,30px);
}
```

Рис. 17.12. Применение функций **scale**, **rotate** *и* **translate** *к одному элементу*

Важно отметить, что преобразования применяются в том порядке, в котором они перечислены. Например, если вы употребите функцию **translate**, а затем **rotate**, результат получится иным, нежели при использовании поворота, а затем перемещения. Также стоит обратить внимание на то, что, если вам нужно применить дополнительное преобразование к другому состоянию (например, при наведении, в фокусе или активному), необходимо повторить все преобразования, уже примененные к элементу. Например, некий элемент **a** повернут на 45 градусов в нормальном состоянии. Применив преобразование **scale** при наведении, я потеряю поворот, если я не укажу его еще раз.

Вендорные префиксы

Для ясности я представляла примеры свойств **transform**, используя только стандартный синтаксис. В действительности во всех браузерах, поддерживающих свойство **transform**, к нему необходимо добавлять вендорные префиксы. Ниже еще раз приведен пример множественных преобразований, как они должны выглядеть на опубликованном сайте:

```
a:hover img, a:focus img{
-webkit-transform: scale(1.5) rotate(-5deg)
translate(50px,30px);
-moz-transform: scale(1.5) rotate(-5deg)
translate(50px,30px);
-o-transform: scale(1.5) rotate(-5deg)
translate(50px,30px);
-ms-transform: scale(1.5) rotate(-5deg)
translate(50px,30px);
transform: scale(1.5) rotate(-5deg)
translate(50px,30px);
}
```

```
a {
transform: rotate(45deg);
}
a:hover {
transform: scale(1.25); /* поворот элемента будет потерян
*/
}
```

Чтобы добиться и поворота, и изменения масштаба, укажите значения обоих преобразований:

```
a:hover {
transform: rotate(45deg) scale(1.25); /* поворачивает и мас-
штабирует */
}
```

Плавные преобразования

Множественные преобразования, примененные к изображению соснового леса, смотрятся интересно, но было бы лучше, если бы мы использовали плавную анимацию, а не резкую. Теперь, когда вы знакомы с переходами и преобразованиями, давайте объединим их и сотворим волшебство. А под «волшебством», конечно же, я имею в виду основные эффекты анимации между двумя состояниями. Мы сделаем это вместе, шаг за шагом, в упражнении 17.2.

УПРАЖНЕНИЕ 17.2. ПЕРЕХОД ПРЕОБРАЗОВАНИЙ

В этом упражнении мы заставим туристические фотографии в галерее, показанной на рис. 17.13, увеличиваться в размере и поворачиваться на определенный угол при наведении на них указателя мыши — и мы сделаем эти эффекты плавными с помощью перехода. Исходный документ (*aquarium.html*) и все изображения доступны в папке с материалами к этой главе на диске, прилагающемся к книге.

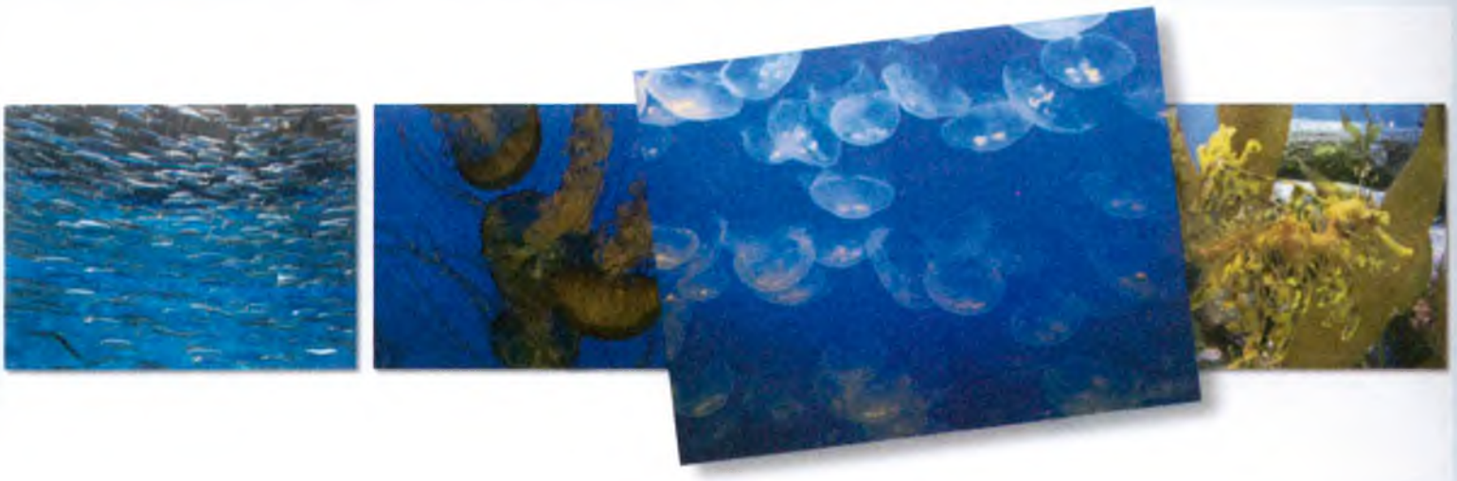


Рис. 17.13. *Фотографии становятся больше и наклоняются в состояниях* **:hover** *и* **:focus**. *Переход используется, чтобы помочь сделать преобразование гладким. Вы можете посмотреть, как он работает, когда закончите это упражнение (или запустить соответствующий файл, записанный на диске, прилагающемся к книге)*

1. Откройте файл *aquarium.html* в текстовом редакторе, и вы увидите, что к нему уже добавлены стили, которые организуют элементы списка по горизонтали и применяют небольшую тень к каждому изображению. (Заметим, что если вы ее не видите, значит, используете устаревшую версию браузера). Первое, что мы собираемся сделать — это добавить свойство **transform** к каждому изображению.
2. В данном случае мы хотим, чтобы преобразования запускались только тогда, когда курсор указывает на изображение, или когда оно сфокусировано, поэтому свойство **transform** должно применяться к состояниям **:hover** и **:focus**. Так как я хочу наклонить все изображения по-разному, следует написать правило для каждого из них, используя уникальный идентификатор изображения в качестве селектора. Когда закончите, сохраните и проверьте свою работу.

```
a:hover #img1, a:focus #img1 {
transform: rotate(-3deg);
}
a:hover #img2, a:focus #img2 {
transform: rotate(5deg);
}
a:hover #img3, a:focus #img3 {
transform: rotate(-7deg);
}
a:hover #img4, a:focus #img4 {
transform: rotate(2deg);
}
```

3. Теперь также немного увеличим фотографии, чтобы они были лучше видны посетителям. Добавьте функцию **scale(1.5)** к каждому из значений свойства **transform**. Ниже представлено первое свойство, вы допишете остальные.

```
a:hover #img1 {
transform: rotate(-3deg) scale(1.5);
}
```

Важно отметить, что файлы изображения создаются сначала большого размера, а затем уменьшаются до величины миниатюры. Если бы мы начали с маленьких изображений и увеличили их масштаб, они выглядели бы отвратительно.

4. Поскольку мы создаем впечатление, будто приподнимаем фотографии на экране, давайте сделаем так, чтобы казалось, будто тень расположена чуть дальше, увеличив значения смещения и размытия, а также осветлив оттенок серого. У всех изображений должен быть одинаковый эффект, поэтому добавьте одно правило, используя значение **a:hover img** в качестве селектора.

```
a:hover img {
box-shadow: 6px 6px 6px rgba(0,0,0,.3);
}
```

Сохраните файл и проверьте его в браузере. Изображения должны наклоняться и увеличиваться при наведении на них указателя мыши. Но действие происходит как-то рывками. Исправим это поведение с помощью перехода.

5. Добавьте сокращенное свойство **transition** к нормальному состоянию изображения **img** (то есть не к состояниям **:hover** или **:focus**). Свойство, для которого мы хотим создать переход, в данном случае — **transform**. Задайте продолжительность в .3 секунды и используйте **linear**.

```
img {
…
transition: transform 0.3s linear;
}
```

Повторюсь, что в полной версии кода к свойству преобразования также необходимо добавлять вендорные префиксы. Например версия Webkit будет выглядет так:

```
-webkit-transition: -webkit-transform .3s linear;
```

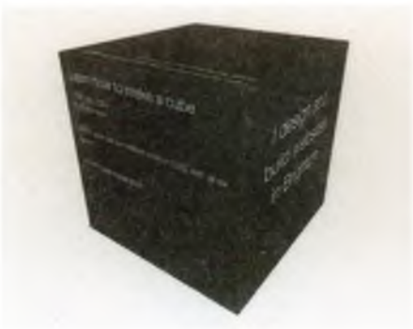
И это все! Вы можете поэкспериментировать с различной продолжительностью и функциями тайминга или попробовать изменить преобразования или их исходные позиции, чтобы увидеть, на что способны другие эффекты.

ПРИМЕЧАНИЕ

Обратите внимание, что я опускаю версии с префиксами, но вам понадобится префикс **-webkit-** для просмотра изменений в браузерах Chrome и Safari.

Трехмерные преобразования

Помимо функций двухмерных преобразований, в спецификации преобразований CSS также описывается система для создания ощущения пространства и перспективы. В сочетании с переходами трехмерные преобразования можно использовать для реализации насыщенных интерактивных интерфейсов, таких как карусели изображений, переворачивающиеся карты или вращающиеся кубики! (В данный момент во Всемирной паутине не существует сокращенного свойства для создания кубов посредством CSS, это станет хорошим проектом для практики.) На рис. 17.14 показано несколько примеров интерфейсов, созданных с помощью трехмерных преобразований. В прошлом, если бы вы увидели подобные трехмерные интерфейсы, вы бы предположили, что для их создания использовали Flash. Сейчас это встроенные возможности браузера и каскадные таблицы стилей.



3D-куб Пола Хэйеса (www.paulhayes.com/experiments/cube-3d/touch.html)

Веб-галерея технологий браузера Safari (developer.apple.com/safaridemos/showcase/gallery/)

Проект Чарльза Енга (www.satine.org/research/webkit/snowleopard/snowstack.html)

Рис. 17.14. Несколько примеров трехмерных преобразований

Трехмерные преобразования не обязательно изучать тем, кто только начинает осваивать веб-дизайн, поэтому я не буду вдаваться во все детали, но дам вам почувствовать, что значит добавить дизайну третье измерение. Если вы хотите узнать больше, для начала подойдут следующие публикации:

- «3D-трансформации средствами CSS» (**habrahabr.ru/post/151300/**)
- «3D-домик на CSS3» (**habrahabr.ru/post/151176/**)
- «CSS 3D Meninas» (**www.romancortes.com/blog/css-3d-meninas/** и **www.romancortes.com/blog/3d-meninas-explained/**)

В качестве очень простого примера я использую изображения из упражнения 17.2 и размещу их так, будто бы они находятся в трехмерной галерее, работающей по типу карусели (рис. 17.15).

Разметка — тот же неупорядоченный список, который использовался в предыдущем упражнении.

```
<ul>

<li><a href=""><img src="anchovies.jpg" id="img1" alt=""></
a></li>
```

```
<li><a href=""><img src="jellyfish1.jpg" id="img2" alt=""></
a></li>
<li><a href=""><img src="bluejellyfish.jpg" id="img3"
alt=""></a>
</li>
<li><a href=""><img src="seadragon.jpg" id="img4"alt=""></
a></li>
</ul>
```

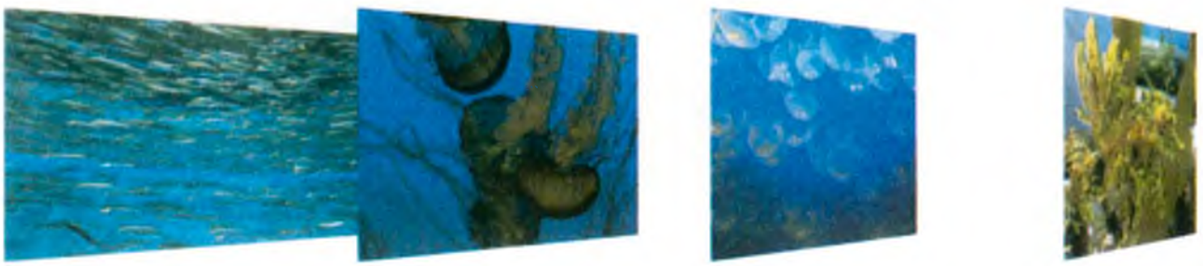


Рис.17.15. Изображения аквариума расположены в пространстве

Первый шаг заключается в добавлении содержащему элементу некоторой «перспективы» с помощью свойства **perspective**. Оно сообщает браузеру, что дочерний элемент должен вести себя так, будто он находится в трехмерном пространстве. Значением свойства **perspective** является некоторое целое число больше нуля, определяющее расстояние от исходного элемента по оси Z. Чем меньше значение, тем резче перспектива. Я выяснила, что значения между 300 и 1500, приемлемы, но вам придется повозиться, прежде чем добьетесь желаемого эффекта.

```
ul {
width: 1000px;
height: 100px;
list-style-type: none;
padding: 0;
margin: 0;
-webkit-perspective: 600;
-moz-perspective: 600;
perspective: 600;
}
```

Свойство **perspective-origin** (не показано) описывает положение ваших глаз относительно преобразованных элементов. Значениям являются положения по горизонтали (**left, center, right**, а также значения длины или процентные) и по вертикали (**top**, **bottom**, **center**, а также значения длины или процентные). По умолчанию (как показано на рис. 17.15) изображение выровнено по центру по вертикали и горизонтали (**perspective-origin:50% 50%**). Последнее свойство,

имеющее отношение к преобразованиям — **backface-visibility**, определяет, будет ли видима обратная сторона элемента при его вращении.

Задав трехмерное пространство, примените одну из функций трехмерных преобразований к каждому элементу **li** внутри неупорядоченного списка **ul**. Трехмерные функции включают в себя: **translate3d**, **translateZ**, **scale3d**, **scaleZ**, **rotate3d**, **rotateX**, **rotateY**, **rotateZ** и **matrix3d**. Вам должны быть знакомы некоторые из этих понятий. Функции *** Z** определяют ориентацию объекта по отношению к оси Z (представьте, что она проходит от вашего носа к странице, на которой лежат оси X и Y).

В нашем примере на рис. 17.15, каждый элемент **li** будет повернут на 45 градусов вокруг своей оси Y (вертикальная ось) с помощью функции **rotateY**. Сравните результат с рис. 17.16, на котором каждый элемент **li** вращается вокруг своей оси X (горизонтальная ось) с помощью функции **rotateX**.

```
li {
float: left;
margin-right: 10px;
-webkit-transform: rotateX(45deg);
-moz-transform: rotateX(45deg);
transform: rotateX(45deg);
}
```

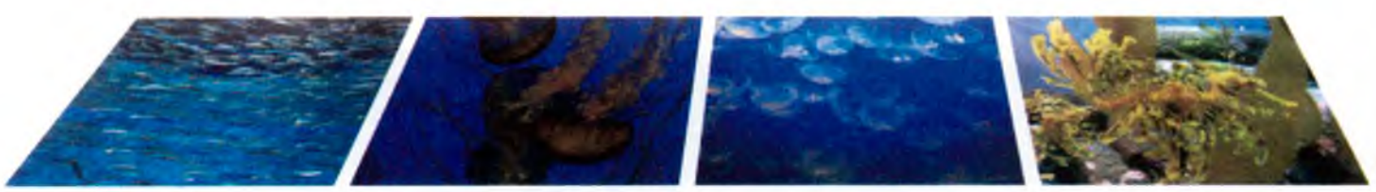


Рис. 17.16. *Тот же список изображений, повернутых по горизонтальной оси с помощью функции* **rotateX**

Очевидно, я охватила лишь небольшую часть того, что можно сделать, используя трехмерные преобразования, но этого должно быть достаточно, чтобы у вас сложилось правильное понимание их работы. Далее я расскажу о более сложном способе приведения страниц в движение.

ПРИМЕЧАНИЕ

Анимация по ключевым кадрам известна как *явная анимация* потому, что вы программируете ее поведение. Переходы, напротив, являются примерами *скрытой анимации*, потому что они срабатывают только при изменении свойства.

Анимация по ключевым кадрам

Модуль анимации CSS позволяет веб-дизайнерам создавать самую настоящую анимацию по ключевым кадрам. В отличие от переходов, в которых одно состояние сменяется другим, анимация по ключевым кадрам позволяет четко указать другие состояния в определенных точках процесса, предоставляя более полный контроль над действием.

Эти «точки процесса» устанавливаются *ключевыми кадрами*, которые определяют начало или конец фрагмента анимации. CSS-переходы представляют собой анимацию с двумя ключевыми кадрами: начальное и конечное состояния. Более сложная анимация требует множества ключевых кадров для управления изменениями свойства в последовательности. Создание анимации по ключевым кадрам — это сложный процесс, который невозможно полностью охватить его в данной книге. Но мне хотелось бы, чтобы вы имели некоторое представление о том, как он работает, поэтому я вкратце опишу минимальные подробности. Следующие публикации послужат хорошим стартом для получения дополнительной информации:

- «Свойство @keyframes CSS3» (**xhtml.co.il/ru/CSS3/keyframes**)
- «CSS3 Анимация» (**uroki-css.ru/css3/css3_animations.php**)
- «Шпаргалка блогера: Анимация CSS» (**shpargalkablog.ru/2012/03/animaciya-css.html**)
- «Анимация с ключевыми кадрами CSS3» (**www.thevista.ru/page16030-animatsiya_s_klyuchevymi_kadrami_css3**)
- «Циклическое слайд-шоу на чистом CSS3» (**habrahabr.ru/post/143025/**)
- «Введение в CSS3 анимации по ключевым кадрам» (**bot.kz/2011/05/13384**)
- «Анимации» (**msdn.microsoft.com/ru-ru/library/ie/hh673530**)
- «Модернизация страницы с помощью анимаций CSS» (**msdn.microsoft.com/ru-ru/library/ie/jj665792**)
- «Мастер-класс по CSS-анимации» (**www.dejurka.ru/css/masterclass-css-animations/**)
- **Anthonycalzadilla.com**. Мой друг Энтони Кальзадилла проделал революционную работу в области CSS анимации, включая изображения шагающих AT-AT и человека (рис. 17.17) посредством CSS3, опередившие свое время. На его сайте представлены ссылки на примеры анимации и основные новости из мира CSS.

Инструменты анимации

Если вы хотите применить CSS-анимацию, но вам не хватает необходимых средств, чтобы научиться писать код самостоятельно, существуют инструменты, представляющие интерфейс временной шкалы для создания анимации и генерирования кода HTML и CSS. Ниже приведены некоторые из таких программ:

- Tumult Hype, **tumultco.com/hype/** (только OS X)
- Sencha Animator, **www.sencha.com/products/animator/**
- Adobe Edge, **labs.adobe.com/technologies/edge/**

Шагающий AT-AT с использованием только CSS3, Энтони Кальзадилла (www.anthonycalzadilla.com/css3-ATAT/)

Шагающий человек с использованием только CSS3, Энтони Кальзадилла (www.optimum7.com/css3-man/)

Анимация «Как я научился ходить» Эндрю Хойера (andrew-hoyer.com/blog/2010/10/21/walking/)

Воссоздание вступления к фильму «Звездные войны» Гильермо Эстевеса (www.gesteves.com/experiments/starwars.html)

Рис. 17.17. Примеры анимации с использованием исключительно CSS

Установка ключевых кадров

Процесс анимации состоит из двух этапов: задавания ключевых кадров с помощью правила **@keyframes**, а затем — добавления специальных свойств к элементу, который будет анимирован. Ниже приведен весьма упрощенный набор ключевых кадров, который изменяет цвет фона элемента с течением времени. Это не очень активная анимация, но она должна дать вам общее представление о том, что делает правило **@keyframes**.

```
@keyframes colors {
0% { background-color: red; }
20% { background-color: orange; }
40% { background-color: yellow; }
60% { background-color: green; }
80% { background-color: blue; }
100% { background-color: purple; }
}
```

В правиле **@keyframes** сообщается следующее: создать анимированную последовательность с именем «colors». В ее начале свойство **background-color** элемента должно быть красным, спустя 20% от времени продолжительности анимации цвет фона должен стать оранжевым, и так далее, пока анимация не закончится. Браузер заполняет все оттенки цвета между каждым ключевым кадром. Я представила этот процесс на рис. 17.18.

Каждое процентное значение и определение свойство/значение указывают ключевые кадры в последовательности анимации. Помимо процентных значений вы можете также использовать зарезервированные слова **from** в начале анимации и **to** для обозначения ее конца. Ниже представлено правило **@keyframes**, содержащее только собственный синтаксис.

```
@keyframes имя анимации {
ключевой кадр { свойство: значение; }
ключевой кадр { свойство: значение; }
```

Рис. 17.18. Создание анимации для смены цветов радуги с помощью ключевых кадров

Для правила
[также необх

Для правила **@keyframes** также необходимы вендорные префиксы, например

```
@-webkit-keyframes
```

Добавление свойств анимации

Теперь мы можем применить эту последовательность к одному или нескольким элементам в документе с помощью набора свойств анимации, очень похожего на набор свойств перехода, которые вы уже знаете. Я применю анимацию радуги к элементу **#magic div** в моем документе.

```
<div id="magic">Магия!</div>
```

В правиле CSS для элемента **#magic** я могу принять несколько решений, касающихся анимации, которую я хочу применить:

- Какую анимацию применить (**animation-name**)
- Как долго она должна продолжаться (**animation-duration**)
- Каким образом она должна воспроизводиться (тайминг) (**animation-timing-function**)
- Задерживать ли ее начало (**animation-delay**)

Кажутся знакомыми? Кроме того, также существуют несколько специфических свойств анимации:

- Сколько раз анимация должна повторяться (**animation-iteration-count**).
- Воспроизводится ли она в прямом, обратном направлении, или направления воспроизведения чередуются (**animation-direction**).
- Будет ли она продолжать воспроизводиться или остановится. Состояние воспроизведения можно включить и выключить с помощью JavaScript или при наведении указателя мыши (**animation-play-state**).
- Должна ли анимация отменять настройки по умолчанию, запрещающие применение данного свойства вне времени запуска (**animation-fill-mode**).

Свойство **animation-name** сообщает браузеру, какую последовательность ключевых кадров применить к элементу **#magic div**. Я также задала продолжительность и функции времени, и использовала свойство **animation-iteration-count**, чтобы повторять анимацию бесконечное число раз. Можно, конечно, указать определенное числовое значение, например 2, чтобы воспроизвести ее дважды, но разве весело, если радуг всего две? И, забавы ради, я установила значение **alternate** свойства **animation-direction**, чтобы заставить анимацию проигрываться в обратном порядке после нормального воспроизведения. Другие варианты — **forward** или **reverse**. Ниже приведено правило, которое в результате получилось для анимированного элемента **div**.

```
#magic {
...
animation-name: colors;
animation-duration: 5s;
```

```
animation-timing-function: linear;
animation-iteration-count: infinite;
animation-direction: alternate;
}
```

Код правила становится немного длинноватым, особенно если учесть, что каждое свойство необходимо повторять с вендорными префиксами. Вы также можете использовать сокращенное свойство анимации, чтобы объединить значения, как мы это делали для перехода.

```
##magic {
animation: colors 5s linear infinite alternate;
}
```

Такова схема создания ключевых кадров или применения анимации к элементу на странице. Чтобы элементы перемещались (то, что мы обычно считаем анимацией), примените ключевые кадры для изменения положения элемента на экране с помощью преобразования **translate** или свойств **top**, **right**, **bottom**, **left**. Когда ключевые кадры объединены в анимацию, объект плавно перемещается из одной позиции в другую. Вы также можете анимировать иные методы преобразования. Надеюсь, я помогла вам осмыслить, как можно использовать CSS, чтобы добавить на страницы немного анимации и плавности. Это классные вещи, но помните, что очень важно применять их умеренно и только как улучшение к нормальным сайтам. Если вы демонстрируете анимацию, было бы неплохо попросить пользователей заменить их браузеры на версии, поддерживающие ее.

ПРИМЕЧАНИЕ

По спецификации, значение свойства **animation-name** в правиле **@keyframe** и свойстве **animation** должно быть указано в одинарных кавычках. Сокращенное определение будет записано следующим образом:

```
animation: 'colors' 5s linear
infinite reverse;
```

Однако разработчики на данный момент опускают кавычки, чтобы избежать ошибок при применении в браузере Firefox.

Резюме

Ниже в алфавитном порядке приведен обзор свойств, рассмотренных в этой главе.

Свойство	Описание
animation	Сокращенное свойство, объединяющее свойства анимации
animation-name	Определяет имя последовательности анимации
animation-duration	Длительность анимации
animation-timing-function	Описывает ускорение (тайминг) анимации
animation-iteration-count	Количество повторов анимации
animation-direction	Указывает, воспроизводится ли анимация в прямом, обратном направлении, или же они чередуются
animation-play-state	Воспроизводится ли анимация или остановлена
animation-delay	Промежуток времени перед запуском анимации
animation-fill-mode	Отменяет ограничения на применение свойств анимации

Свойство	Описание
`backface-visibility`	Определяет, будет ли видима обратная сторона элемента при трехмерных преобразованиях
`perspective`	Определяет элемент как трехмерное пространство и указывает воспринимаемую глубину
`perspective-origin`	Определяет положение ваш угол зрения в трехмерном пространстве
`transform`	Указывает, что отображение элемента должно быть изменено с помощью одной из функций двухмерного или трехмерного преобразования
`transform-origin`	Точка, вокруг которой происходит преобразование элемента
`transform-style`	Используется для сохранения трехмерного контекста, когда преобразуемые элементы являются вложенными
`transition`	Сокращенное свойство, объединяющее свойства перехода
`transition-property`	Определяет, к какому свойству CSS будет применен переход
`transition-duration`	Продолжительность анимации перехода
`transition-timing-function`	Описывает характер перехода (изменения скорости)
`transition-delay`	Период времени перед началом перехода

ГЛАВА 18

ТЕХНИЧЕСКИЕ ПРИЕМЫ CSS

На данный момент у вас уже есть прочный фундамент для написания таблиц стилей. Вы можете задавать стили тексту и блокам элементов, создавать макеты страниц при помощи плавающих элементов и даже добавлять ненавязчивые эффекты анимации в свои проекты. Но есть еще несколько распространенных технических приемов CSS, с которыми я хочу вас познакомить перед тем, как мы перейдем к изучению языка JavaScript.

Эта глава полна самых разных сведений. Она начинается с описания техник, входящих в базовый набор инструментов веб-разработчика: удаления стилей браузера с помощью сброса стилей CSS, использования изображений вместо текста (только при необходимости!) и сокращение числа запросов на сервер благодаря CSS-спрайтам. Затем мы перейдем к общим подходам и специальным свойствам для форматирования веб-форм и таблиц. Лучшее я оставила напоследок — вы будете применять медиазапросы в пошаговых упражнениях для создания адаптивного сайта.

В этой главе

- Сброс стилей CSS
- Замена текста изображением
- Применение CSS-спрайтов
- Стилизация веб-форм
- Стилизация таблиц
- Использование медиазапросов в адаптивном дизайне

Сброс стилей CSS

Как вы знаете, у браузеров есть собственные встроенные таблицы стилей (так называемые таблицы стилей *пользовательского агента*) для отображения HTML-элементов. Например, если вы не предоставите стили для элемента `h1`, можете быть уверены, что он отобразится крупным жирным текстом с интервалами сверху и снизу. Но размер шрифта и величина интервала могут варьироваться от браузера к браузеру, приводя к непредсказуемым результатам. Кроме того, даже если вы предоставите собственные таблицы стилей, элементы документа могут наследовать определенные стили из таблицы стилей пользовательского агента, что также дает неизвестный результат. Поэтому многие дизайнеры используют так называемый *сброс CSS* — набор правил стилей, который отменяет все стили пользовательского агента и создает как можно более нейтральную отправную точку. После этого необходимо четко указать стили шрифта, текста, полей и отступов для каждого элемента в документе, но вы можете быть уверены, что ни один из стилей браузера не будет им мешать.

Самый популярный сброс был написан Эриком Мейером (автором многих книг по CSS). Его код представлен ниже. Кроме того, я включила копию этого кода в подборку материалов для этой главы, чтобы вы могли его использовать.

```
/* http://meyerweb.com/eric/tools/css/reset/
v2.0 | 20110126 Лицензия: нет (общий домен)*/
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
margin: 0;
padding: 0;
border: 0;
font-size: 100%;
font: inherit;
vertical-align: baseline;
}
/* сброс типа отображения HTML5 для старых версий браузеров
*/
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
display: block;
}
body {
line-height: 1;
}
ol, ul {
list-style: none;
}
blockquote, q {
```

```
quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
content: '';
content: none;
}
table {
border-collapse: collapse;
border-spacing: 0;
}
```

Чтобы применить сброс, поместите эти правила в верхней части собственной таблицы стилей. Их можно использовать точно так же, как показано здесь, или настроить в соответствии с требованиями вашего проекта. Кроме того, я рекомендую прочитать публикацию на сайте **habrahabr.ru/post/45296/**.

Сброс CSS — не для всех. Возможно, вы решите, что стоит отдать на откуп браузеру некоторые основные стили, чтобы не приходилось писать их для каждого мелкого фрагмента контента. Но если вы хотите быть уверены, что все отображающиеся стили — ваши, сброс окажется наиболее верным способом.

ПРИМЕЧАНИЕ

Существует еще один сброс, который стал доступен благодаря разработчикам в компании Yahoo!. Для его использования вставьте следующую строку в раздел заголовка своего HTML-документа:

```
<link rel="stylesheet"
type="text/css" href="http://
yui.yahooapis.com/3.5.1/
build/cssreset/cssreset-min.
css">
```

Однако перед тем, как это сделать, обязательно ознакомьтесь с инструкцией на сайте **yuilibrary.com/yui/docs/cssreset/**.

Технические приемы замены текста изображением

До того как появились веб-шрифты, нам приходилось использовать изображения каждый раз, когда требовалось сделать буквы более вычурными, чем в шрифтах Times New Roman или Arial. К счастью, это больше не проблема, и у нас могут быть очень стильные заголовки и оформление текста без дополнительного бремени изображений. Время от времени, однако, даже веб-шрифт не подходит, и приходится вместо нескольких слов текста использовать изображение. Например, вы можете вставить стилизованный логотип вместо названия вашей компании или знакомые иконки вместо текстовых ссылок. Удаление текста полностью и замена его элементом **`img`** — плохая идея, поскольку ценный контент пропадет навсегда. Лучше использовать *приемы замены текста изображением* на основе CSS, которые размещают изображение в качестве фона элемента, а затем смещают текст, чтобы он не отображался на странице. Графические браузеры отображают фоновое изображение, а текстовый контент находится в файле для поисковых систем, программ экранного доступа и других вспомогательных устройств — таким образом выигрывают все.

Элегантная техника замены текста изображением была придумана Скоттом Келламом. В ней используется свойство **text-indent** для вытеснения абзаца текста вправо за пределы видимого блока элемента (рис. 18.1).

Рис. 18.1. *Техника замены текста изображением Скотта Келлама скрывает HTML-текст, сдвигая его за пределы видимого блока элемента с помощью свойства* **text-indent**

В этом примере я заменю HTML-текст «Малышок» элемента **h1** замечательным логотипом. Разметка проста:

```
<h1 id="logo">Малышок</h1>
```

Правило стилей следующее:

```
h1#logo {
width: 450px;
height: 80px
background: url(malishok.png) no-repeat;
text-indent: 100%;
white-space: no-wrap;
overflow: hidden;
}
```

Здесь следует отметить несколько моментов. Во-первых, элемент **h1** отображается в виде блока по умолчанию, поэтому мы можем просто указать его ширину и высоту в соответствии с размерами изображения, используемого в качестве фона. Свойство **text-indent** вытесняет слово «Малышок» вправо на всю ширину (100%) элемента. Для свойства **white-space** задано значение **no-wrap**, которое гарантирует, что длинные строки текста не будут перенесены и не появятся снова в элементе

окна. Наконец, свойство **overflow: hidden** указывает браузеру, что весь контент, выходящий за пределы блока элемента (как наш текст **h1**), не должен отображаться.

Фактически существует около дюжины технических приемов замены текста изображением, которые появились за эти годы. Одним из самых популярных является прием Phark, который использует очень большие отрицательные значения свойства **text-indent** (обычно -9999px), чтобы сместить HTML-текст влево за пределы области просмотра.

```
h1#logo {
width: 450px;
height: 100px
background: url(malishok.png) no-repeat;
text-indent: -9999px;
}
```

Недостатком этого подхода является то, что браузеры вынуждены рассчитывать и рисовать широкие элементы блока, даже если они и не отображаются, что снижает производительность. Но если вам встретится пример текста с фоновым изображением и отступом -9999px, вы будете знать, что происходит.

Недостатком любого подобного приема являются дополнительные запросы на сервер для каждого используемого изображения. В следующем разделе мы рассмотрим способ, как свести их к минимуму.

ПРИМЕЧАНИЕ

С помощью приемов замены можно также замещать одни изображения другими, например стандартный веб-рисунок — изображением с высоким разрешением, если страница распечатывается или просматривается на экранах с высокой плотностью пикселов (Retina). Аарон Густафсон описывает этот подход в своем блоге по адресу: **v2.easy-designs.net/articles/iIR** и **blog.easy-designs.net/archives/2012/04/16/iir-redux**.

CSS-спрайты

Когда ранее, в главе 3, я говорила о производительности сайта, то заметила, что вы можете улучшить ее за счет сокращения числа запросов, которые страница отправляет на сервер (то есть запросов HTTP). Одна из стратегий, призванных уменьшить их количество, состоит в том, чтобы объединить все маленькие изображения в один графический файл так, чтобы браузер запрашивал только одно изображение. Большое изображение, содержащее в себе несколько других, известно как *спрайт*. Этот термин появился на заре индустрии компьютерной графики и видеоигр. Спрайт размещается в элементе с помощью свойства **background-position** таким образом, что видна только нужная его часть. Пример поможет вам понять суть приема.

На сайте Velocity Conference присутствуют девять часто встречающихся значков социальных медиа, как показано на рис. 18.2. Чтобы повысить производительность сайта, Тони Кванторолло и Зебулон Янг превратили эти графические значки в один спрайт, объединив их все в одно изображение и оставив между ними расстояние в два пиксела. Таким образом количества http-запросов уменьшилось.

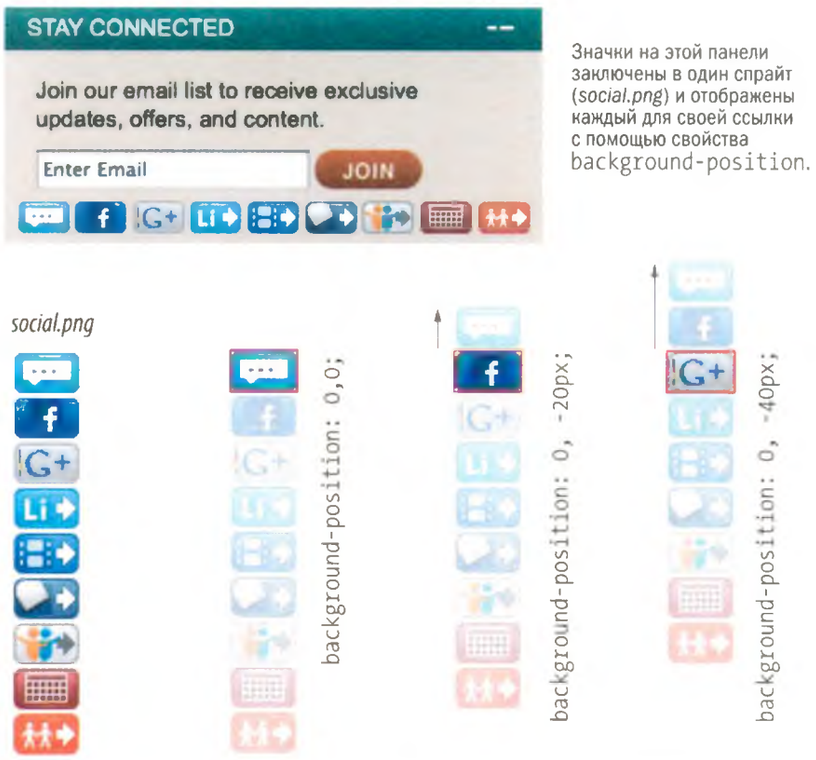


Рис. 18.2. *Замена отдельных графических файлов одним спрайтом сокращает число HTTP-запросов на сервер и улучшает производительность сайта*

Приведенные ниже стили и разметка являются упрощенным вариантом кода, используемого на сайте конференции, но дают тот же результат.

ПРИМЕЧАНИЕ

Чтобы узнать больше о CSS-спрайтах, прочитайте статью «CSS спрайты: основные техники и полезные инструменты» (**habrahabr.ru/post/159027/**). В ней приведены замечательные примеры спрайтов во Всемирной паутине.

Разметка

```
<ul>
<li><a href="" class="hide twitter">Twitter</a></li>
<li><a href="" class="hide fb">Facebook</a></li>
<li><a href="" class="hide gplus">Google+</a></li>
<li><a href="" class="hide linkedin">LinkedIn</a></li>
<li><a href="" class="hide blip">BlipTV</a></li>
<li><a href="" class="hide lanyrd">Lanyrd</a></li>
<li><a href="" class="hide slides">Slideshare</a></li>
<li><a href="" class="hide sched">Расписание</a></li>
<li><a href="" class="hide attendees">Список участников</
a></li>
</ul>
```

Стили

```
.hide {
text-indent: 100%;
```

```
white-space: nowrap;
overflow: hidden;
}
li a {
display: block;
width: 29px;
height: 18px;
background-image: url(social.png);
}
li a.twitter { background-position: 0 0;}
li a.fb { background-position: 0 -20px;}
li a.gplus { background-position: 0 -40px;}
li a.linkedin { background-position: 0 -60px; }
li a.blip { background-position: 0 -80px; }
li a.lanyrd { background-position: 0 -100px; }
li a.slides { background-position: 0 -120px; }
li a.sched { background-position: 0 -140px; }
li a.attendees { background-position: 0 -160px; }
```

В разметке у каждого элемента по два значения класса. Значение **hide** используется в качестве селектора для замены текста изображением, о чем говорилось в предыдущем разделе.

Другое имя класса — индивидуальное для каждой ссылки на социальную сеть. Уникальные значения класса позволяют нам позиционировать спрайт соответствующим образом для каждой ссылки.

В верхней части таблицы стилей находятся стили замены текста изображением. Обратите внимание, что в следующем правиле все элементы ссылки (**a**) используют изображение *social.png* в качестве фонового.

Наконец мы добрались до стилей, которые выполняют основную работу. Свойство **background-position** имеет разные значения для каждой ссылки в списке, и видимый блок элемента работает как маленькое окошко, демонстрируя часть фонового изображения. Значение для первого элемента `"0,0"`; оно помещает левый верхний угол изображения в левом верхнем углу блока элемента.

Чтобы сделать видимым значок Facebook, нам нужно переместить изображение вверх на 20 пикселов, поэтому его положение по вертикали будет задано как –20px (положение 0 по горизонтали подойдет). Для каждой ссылки изображение перемещается верх с шагом в 20 пикселов, открывая области изображения в спрайте все ниже и ниже.

В данном примере все значки имеют одинаковые размеры и аккуратно выстроены, но это не обязательно.

Вы можете комбинировать в одном спрайте изображения с различными размерами. Процесс задавания размера элемента, а затем его выравнивания с помощью свойства **background-position** выглядит точно так же.

Генераторы спрайтов

Существует несколько инструментов, которые автоматически создают файлы спрайтов и соответствующие им стили. Ниже представлены лишь некоторые из них:

- SpriteMe (**spriteme.org**). SpriteMe — это инструмент для конвертации изображений на существующем сайте в спрайты и правила стилей. Перейдите на ваш веб-сайт, нажмите кнопку SpriteMe, и программа проанализирует страницу, указывая, какие изображения можно объединить в спрайты.
- CSS Sprite Generator (**spritegen.website-performance.org**). CSS Sprite Generator представляет собой веб-сервис, позволяющий загружать отдельные изображения, которые необходимо преобразовать в спрайт, и создает CSS-код для управления ими.

ПРЕДУПРЕЖДЕНИЕ

Спрайты нельзя использовать для мозаичных фоновых изображений (по крайней мере, если не использовать некоторые хитрые приемы). Применяйте их к одиночным фоновым изображениям.

Языки Sass и LESS

Я знаю, вы только привыкаете к регулярному написанию правил CSS, но существуют несколько очень эффективных альтернатив, о которых я хочу вам рассказать. Посчитав обычный синтаксис CSS повторяющимся, разработчики Хэмптон Кэтлин и Нэйтан Вейзенбаум создали новый синтаксис таблиц стилей, который для экономии времени использует преимущества инструментов, заимствованных у языков написания сценариев. Они назвали свой новый синтаксис *Sass* (Syntactically awesome style sheets — Синтаксически удивительная таблица стилей). Его более поздняя версия, известная как SCSS (Sassy CSS — Дерзкий CSS) основана на исходном синтаксисе с отступами Sass, но также позволяет применять и обычный синтаксис CSS.

В документах со стилями Sass можно делать то же, что и в сценариях, например задавать имя переменной для значения, которое вы планируете часто использовать. Например на сайтах компании O'Reilly неоднократно используется один и тот же оттенок красного цвета, так что они могли бы создать переменную с именем «oreilly-red» и использовать ее имя для обозначения цвета. Таким образом, если в будущем им понадобится скорректировать оттенок, будет нужно только изменить значение переменной в одном месте. Ниже показано, как выглядит задавание и использование переменной в Sass:

```
$oreilly-red: #900;
a { border-color: $oreilly-red; }
```

Вы даже можете многократно использовать целый набор стилей с помощью условного обозначения под названием *смешивание* (**mixin**). В следующем примере сохраняется сочетание стилей фона, цвета и границы в смешивании с именем **special**. Чтобы применить это сочетание стилей, добавьте его в определение с помощью правила **@include** и укажите его имя.

```
@mixin special {
color: #fff;
background-color: #befc6d;
border: 1px dotted # 59950c;
}
a.nav {
@include special;
}
```

Кроме того, язык Sass позволяет использовать вложенные правила, выполняет математические операции, а также математически настраивает цвета, и это только несколько функций, заимствованных из языков сценариев. Браузеры не знают, как интерпретировать синтаксис файла *.sass* или *.scss*, поэтому вам необходимо будет использовать компилятор Sass (написанный на Ruby), который запускается на сервере. Прежде чем файл Sass доставляется в браузер, компилятор преобразует его в стандартный синтаксис CSS.

LESS — еще один синтаксис CSS, обладающий способностями, напоминающими языки сценариев. Он очень похож на Sass, но в нем отсутствуют некоторые возможности и имеются незначительные отличия в синтаксисе (например, переменные в LESS обозначены символом **@** вместо **$**, например, **@oreilly-red**). Другое отличие в том, что файл LESS преобразуется в обычный синтаксис CSS с помощью сценария JavaScript (*less.js*), а не Ruby. Обратите внимание, что компиляция файла LESS в CSS сильно загружает процессор и будет замедлять работу браузера. По этой причине лучше преобразовать файл в CSS перед отправкой его на сервер. Для этого рекомендуется использовать инструмент CodeKit (**incident57.com/less/**) или другие аналогичные приложения.

Как только вы приобретете немного практического опыта и почувствуете, что готовы выводить свои таблицы стилей на новый уровень, изучите некоторые из этих ресурсов (статей) о языках Sass и LESS:

- Веб-сайт Sass (**sass-lang.com**)
- Веб-сайт LESS (**lesscss.org**)
- Compass, полнофункциональная основа для создания кода CSS, использующая Sass (**compass-style.org**)
- Статья «Знакомство с LESS и сравнение с Sass» (**www.coolwebmasters.com/frameworks-and-platforms/1972-an-introduction-to-less-and-comparison-to-sass.html**)
- Статья «Краткий обзор отличий LESS от SASS» (**habrahabr.ru/post/130886/**)
- Статья «SASS против LESS» (**habrahabr.ru/post/144309/**)

Стилизация веб-форм

Только что созданные веб-формы без примененных к ним стилей могут быть похожи на сборную солянку из полей (рис. 18.3), так что вы, конечно, захотите, придать им более профессиональный внешний вид с помощью CSS. Формы не только выглядят лучше, но и, как показывают исследования, гораздо проще и быстрее в использовании, когда их метки и поля красиво выровнены. В этом разделе мы рассмотрим, как можно стилизовать различные элементы формы, и как их выровнять без табличной верстки.

Оформление форм с помощью стилей — занятие довольно проблематичное, так как браузеры обрабатывают их элементы самыми различными способами. Но усилия, затраченные на то, чтобы заставить формы выглядеть так же профессионально, как и весь сайт, того стоят.

Не существует специальных свойств для стилизации веб-форм; используйте стандартные свойства цвета, фона, шрифта, границ, полей

Рис. 18.3. Веб-формы, состоящие только из HTML (слева), как правило, некрасивые, и их трудно использовать. Немного CSS может значительно изменить ситуацию (справа). В текущем разделе вы шаг за шагом рассмотрите оформление этой веб-формы с помощью стилей

и отступов, которые вы изучили в предыдущих главах. Ниже приведен краткий список того, что можно сделать с каждым элементом веб-формы.

Поля ввода текста (`text`, `password`, `email`, `search`, `tel`, `url`)

Позволяют изменить внешний вид поля с помощью свойств **`width`**, **`height`**, **`background-color`**, **`background-image`**, **`border`**, **`border-radius`**, **`margin`**, **`padding`** и **`box-shadow`**. Вы также можете добавить стили к тексту в поле ввода, применив **`color`** и другие свойства шрифтов.

Элемент `textarea`

Может быть оформлен стилями так же, как поля ввода текста. В элементах **`textarea`** по умолчанию используется моноширинный шрифт, поэтому вам нужно будет изменить его в соответствии с другими вашими полями ввода текста. Так как элемент допускает ввод нескольких строк текста, можно также указать свойство **`line-height`**. Обратите внимание, что некоторые браузеры отображают маркер в нижнем правом углу окна текстовой области, что позволяет изменять ее размер, но это можно отменить, добавив стиль **`resize: none;`**.

Кнопка (`submit`, `reset`, `button`)

Примените любое из свойств поля к кнопкам отправки данных и сброса (**`width`**, **`height`**, **`border`**, **`background`**, **`margin`**, **`padding`** и **`box-shadow`**). Однако обратите внимание, что для кнопок по умолчанию задана модель **`border-box`**, поэтому значения ширины и высоты применяются от одной границы до другой. Большинство браузеров также по умолчанию добавляют небольшой отступ, но это можно отменить значением свойства **`padding`**. Вы можете также применить стили к тексту, отображающемуся на кнопке.

Переключатели и флажки

Оптимальный способ работы с флажками и переключателями — оставить их в покое. В лучшем случае браузер Internet Explorer отобразит вокруг блока некоторый цвет, что выглядит неуклюже. Если вы упорны, можете использовать JavaScript, чтобы полностью изменить эти элементы.

Раскрывающиеся и прокручиваемые списки

Вы можете указать ширину и высоту элемента **`select`**, но обратите внимание, что по умолчанию для него используется модель задавания размера элемента **`border-box`**. Некоторые браузеры позволяют применять свойства **`color`**, **`background-color`**, а также свойства шрифта к элементам **`option`**, но, вероятно, лучше оставить их отображение браузеру и операционной системе.

Элементы `fieldset` и `legend`

Вы можете обращаться с элементом **`fieldset`** так же, как с любым другим блоком элемента, настраивая границу, фон, поля и отступы. Пол-

ное отключение границы — один из способов поддержания аккуратного внешнего вида формы при сохранении ее семантики и доступности. Элемент **legend** по умолчанию располагается с отступом, выровнен по центру по вертикали по верхней границе элемента **fieldset**, и, к сожалению, браузеры не позволяют легко изменить это поведение. Некоторые разработчики для получения более предсказуемых результатов используют элементы **span** или **b** внутри элемента **legend** и применяют к ним стили.

Теперь мы знаем, что можем применить стили к отдельным элементам формы, но главная цель — сделать ее более организованной и удобной в использовании. В прошлом для выполнения этой задачи использовался табличный дизайн, но теперь желательно придерживаться CSS, особенно выполняя выравнивание. Я покажу, как написать стили для веб-формы, показанной на рис. 18.3, и проведу вас к результату шаг за шагом.

Разметка

Ниже приведена разметка веб-формы заказа. Каждый элемент заключен в пункт списка и для всех представлены текстовые метки. Также в форме имеются два элемента **fieldset**, группирующие переключатели и флажки.

```
<form action="" method="">
<h2>Форма заказа</h2>
<ul>
   <li>
      <label for="form-name">Имя:</label>
      <input type="text" name="username" id="form-name"
class="textinput">
   </li>
   <br>
   <li>
      <label for="form-email">Email:</label>
      <input type="email" name="emailaddress" id="form-
email" class="textinput">
   </li>
   <br>
   <li>
      <label for="form-tel">Телефон:</label>
      <input type="tel" name="telephone" id="form-
tel"class="textinput">
   </li>
   <br>
   <li>
      <label for="form-story">Примечание:</label>
```

ПРИМЕЧАНИЕ

Вы можете обратить внимание, что веб-форма в этом примере похожа на ту, что вы создавали в главе 9. Я немного упростила ее, чтобы сократить объем кода.

```
      <textarea name="story" maxlength="300" id="form-
story" rows="3" cols="30" placeholder="Не более 300 симво-
лов..."></textarea>
   </li>
   <br>
   <li>
      <label for="sizes">Размер:</label>
      <select name="size">
         <option>35</option>
         <option>36</option>
         <option>37</option>
         <option>38</option>
         <option>39</option>
         <option>40</option>
         <option>41</option>
         <option>42</option>
         <option>43</option>
         <option>44</option>
         <option>45</option>
      </select>
      <em>Стандартные российские размеры</em>
   </li>
   <br>
   <li>
      <fieldset id="colors">
         <legend>Цвет</legend>
         <ul>
            <li><label><input type="radio" name="color"
value="red"> Красный</label> </li>
            <li><label><input type="radio" name="color"
value="blue"> Синий</label> </li>
            <li><label><input type="radio" name="color"
value="black"> Черный</label></li>
            <li><label><input type="radio" name="color"
value="silver"> Серебрянный</label></li>
         </ul>
      </fieldset>
   </li>
   <br>
   <li>
      <fieldset id="features">
      <legend>Характеристики</legend>
         <ul>
            <li>
               <label><input type="checkbox"
name="feature" value="laces"> Глянцевые ободки</label>
```

```
            </li>
            <li>
               <label><input type="checkbox"
name="feature" value="logo" checked> Металлическая бляшка</
label>
            </li>
            <li>
               <label><input type="checkbox"
name="feature" value="heels"> Светящаяся подошва</label>
            </li>
            <li>
               <label><input type="checkbox"
name="feature" value="mp3"> Mp3-проигрыватель</label>
            </li>
         </ul>
      </fieldset>
   </li>
   <br>
   <li class="buttons">
      <input type="submit" value="Заказать!">
      <input type="reset">
   </li>
</ul>
</form>
```

Шаг 1: Добавление основных стилей

Первый набор стилей влияет на основное оформление документа, в том числе элементов **body**, **h2**, а также на некоторые стандартные стили элемента **ul** для удаления маркеров списка. Я создала правило для элемента **form**, задав его ширину, цвет фона, скругленные углы, тень и небольшой отступ. Так как известно, что его контент по большей части будет плавающим, я добавила значение **overflow:hidden;** в качестве контейнера плавающего элемента. Точно так же правило **ul li** содержит значение **clear:both;** в ожидании плавающих элементов. Чтобы сэкономить немного пространства, ниже приведены только стили, имеющие отношение к форме. Результат показан на рис. 18.4.

```
ul li {
clear: both;
list-style-type: none;
…
}
form {
width: 40em;
border: 1px solid #666;
```

```
border-radius: 10px;
box-shadow: .2em .2em .5em #999;
background-color:#d0e9f6;
padding: 1em;
overflow: hidden;
}
```

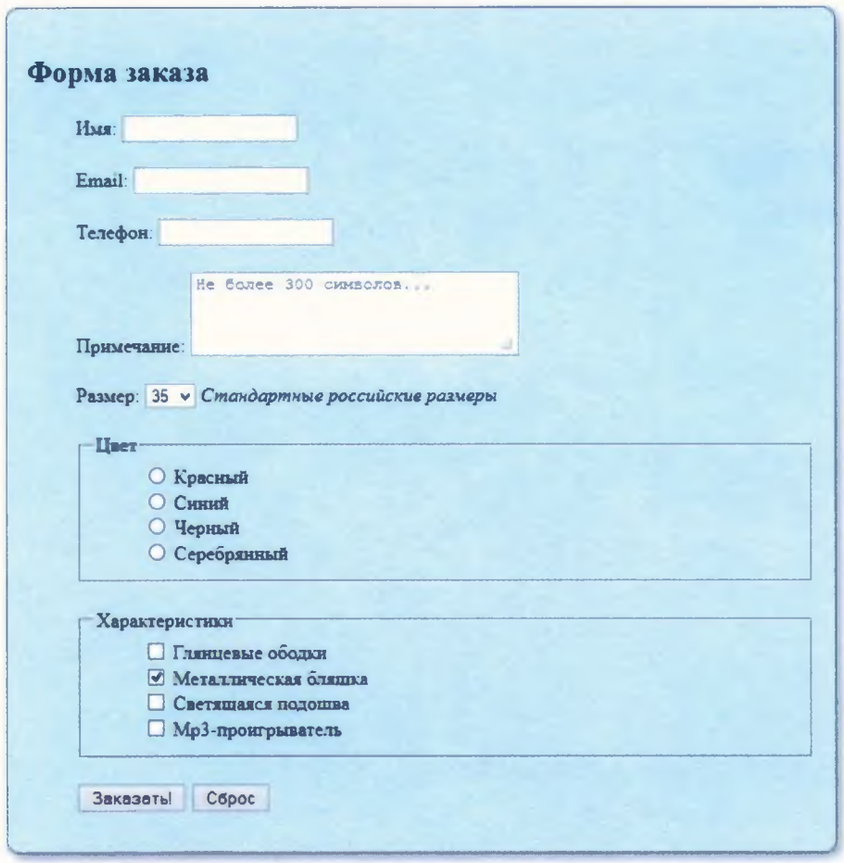


Рис. 18.4. Результат после добавления стилей к элементам формы

Шаг 2: Выравнивание текста и полей ввода

Теперь мы добрались до интересного! Обратите внимание, что на снимке «после» на рис. 18.3 все надписи и поля ввода выровнены аккуратными колонками. Чтобы так сделать, задайте элементам **label** определенную ширину, сместите их влево с помощью свойства **float**, а затем выровняйте текст надписей по правому краю таким образом, чтобы они находились вблизи соответствующих им полей ввода. Вы должны увидеть работу всех этих стилей в правиле элемента **label**, приведенном ниже.

```
label {
display: block;
float: left;
width: 10em;
```

```
text-align: right;
margin-right: .5em;
color: #04699d;
}
```

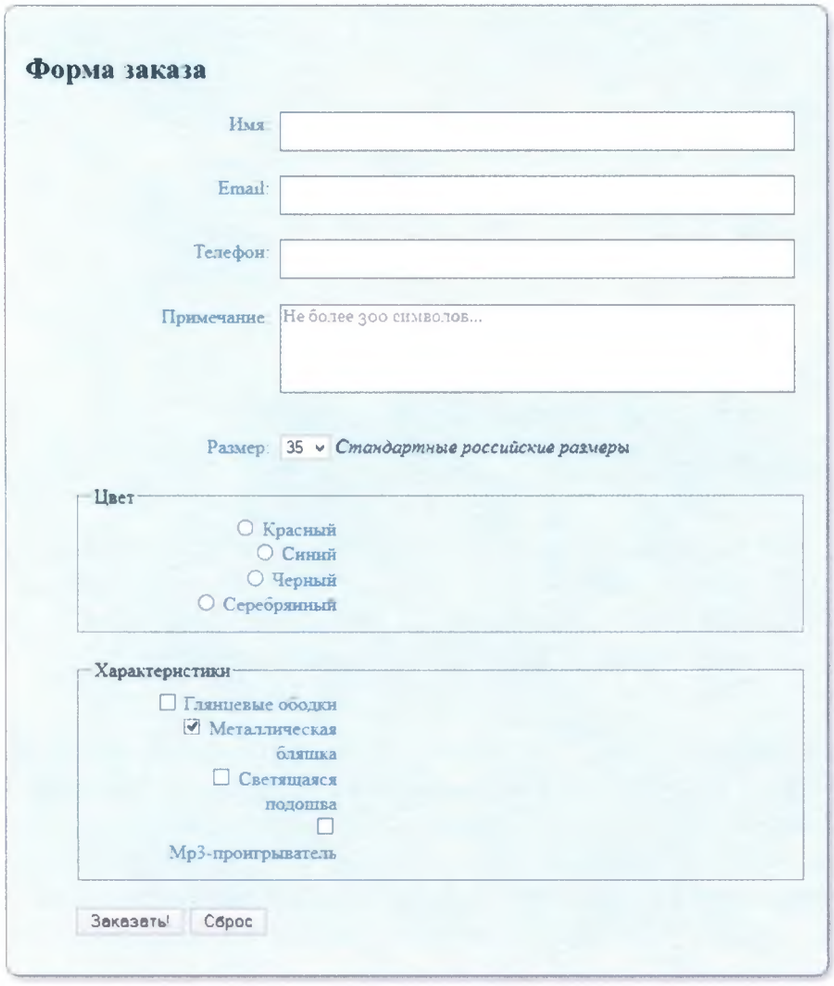


Рис. 18.5. Результат после выравнивания надписей и полей ввода

Полям ввода текста и элементу **textarea** заданы значения ширины и высоты, а также простая граница толщиной в 1 пиксел. Кроме того, к обоим применяются схожие свойства шрифта. Я удалила возможность изменять размер элемента **textarea**, задав значение **none** свойства **resize**. Форма на рис. 18.5 выглядит немного лучше, но теперь у нас проблемы с надписями переключателя и флажков, которые необходимо исправить.

```
input.textinput {
width: 30em;
height: 2em;
border: 1px solid #666;
}
textarea {
```

ПРИМЕЧАНИЕ.

Я добавила атрибут **class="textinput"** к различным типам ввода текста (**text**, **tel** и т. д.), чтобы можно было выбрать только поля ввода текста. Также можно было использовать для каждого из них селекторы атрибутов (например, **input[type="tel"]**), но они не поддерживаются некоторыми версиями браузера Internet Explorer. Я выбрала более надежный метод **class**, так как хочу, чтобы все пользователи увидели эти стили.

```
display: block;
width: 30em;
height: 5em;
border: 1px solid #666;
margin-bottom: 1em;
line-height: 1.25;
overflow: auto;
resize: none;
}
input.textinput, textarea {
font-family: Georgia, "Times New Roman", Times, serif;
font-size: .875em;
}
```

Шаг 3: Настройка групп элементов и текстовых надписей формы

Следующее, что я собираюсь сделать, это переопределить заданные по умолчанию стили элементов **fieldset**, чтобы они стали менее заметны. Я также собираюсь оформить элемент **legend** в каждой группе элементов формы с помощью тех же стилей, что были применены к надписям.

В результате добавления стилей в шаге 2, элементы **label** переключателей и флажков внутри элементов **fieldset** оформлены так же, как основные надписи, но мне это не нравится. Я написала стили, которые сбрасывают цвет, размеры и плавающие элементы, специально для элементов **label**, расположенных внутри групп элементов формы «Цвет» и «Характеристики». Наконец я отобразила элементы списка в разделе «Цвет» как встроенные, так что они будут появляться на одной строке и сэкономят пространство. Флажки в разделе «Характеристики» требуют небольшой коррекции, такой как добавление левого поля, чтобы выровнять их по отношению к остальным элементам формы (**margin-left:11em**), и сброс свойства **clear**, чтобы первый пункт списка флажков не начинался под плавающим элементом **legend** (**clear:none**). Результат показан на рис. 18.6.

```
fieldset {
margin: 0;
padding: 0;
border: none;
}
legend {
display: block;
width: 10em;
float: left;
margin-right: .5em;
```

```
text-align: right;
color: #04699d;
}
#features label, #colors label {
color: #000;
display: inline;
float: none;
text-align: inherit;
width: auto;
font-weight: normal;
background-color: inherit;
}
#colors ul li {
display: inline;
margin-bottom: 0;
}
#features ul {
margin-left: 11em;
}
#features ul li {
margin-bottom: 0;
clear: none;
}
```

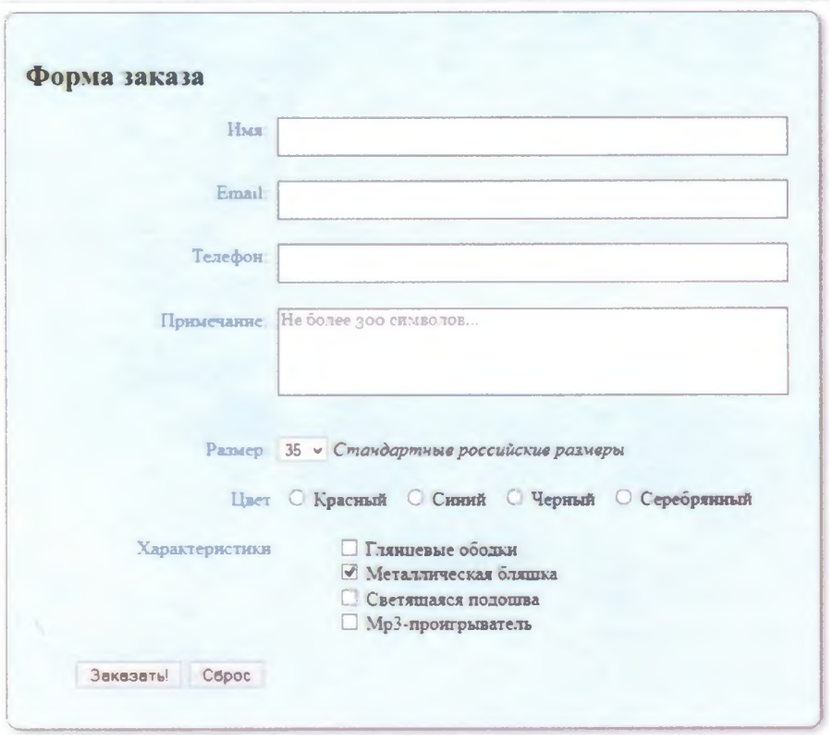


Рис. 18.6. Размещение надписей рядом с флажками и переключателями

```
margin-left: 10.5em;
margin-right: 1em;
color: #C00; /* Текст кнопки отправки данных красного цвета
для привлечения внимания */
}
```

В этом примере я уделила основное внимание стилям, используемым для выравнивания, добавления цветов и оформления текста. В собственных веб-формах вы, возможно, решите добавить стили для интерактивности, такие как стили состояния **:hover** к кнопкам и стили состояния **:focus** к выделенным полям ввода текста.

Стилизация таблиц

Мы уже охватили подавляющее большинство свойств стилей, которые вам понадобятся, чтобы задать стили контенту в таблицах. Например внешний вид и выравнивание контента внутри ячеек, как и любого другого текстового элемента, можно изменить с помощью различных свойств шрифта, текста и фона. Кроме того, вы можете обработать саму таблицу и ячейки при помощи отступов, полей и границ.

Но есть еще несколько свойств CSS, которые были созданы специально для таблиц. Некоторые из них довольно сложны для понимания и кратко представлены во врезке «Расширенные свойства таблиц». Этот раз-

Расширенные свойства таблиц

Существует несколько свойств, относящихся к табличной модели CSS, которые вам вряд ли понадобятся, если вы только приступили к веб-дизайну.

Макет таблицы

Свойство **table-layout** позволяет верстальщикам задавать один из двух методов вычисления ширины таблицы. Значение **fixed** определяет ширину таблицы на основе значений **width**, заданных для таблицы, столбцов или ячеек. Значение **auto** вычисляет ширину таблицы на основе минимальной ширины ее контента. В последнем случае таблица может выводиться медленнее, потому что браузер должен вычислить значение ширины каждой ячейки, прежде чем перейти к ширине таблицы.

Значения свойства отображения таблиц

В главе 14 было введено свойство **display**, используемое, чтобы задать вида блока, генерируемого элементом в макете. CSS спроектирован для работы со всеми языками XML, не только HTML и XHTML. Похоже, что другие языки испытывают потребность в табличных макетах, но не имеют в своих словарях элементов наподобие **table**, **tr** или **td**.

Для отображения таблиц существует множество связанных с ними значений свойства **display**, которые позволяют авторам, использующим языки XML, присвоить поведение макета таблицы любому элементу. Такими свойствами **display** являются **table**, **inline-table**, **table-row-group**, **table-header-group**, **table-footer-group**, **table-row**, **table-column-group**, **table-column**, **table-cell** и **table-caption**. Вы можете присвоить данные типы отображения другим HTML-элементам. Но обычно такой подход не одобряется, поскольку не все браузеры могут полностью поддерживать эти значения.

дел ориентируется на свойства, которые непосредственно затрагивают отображение таблиц; особенно обработку границ.

Раздельные и сливающиеся границы

CSS предоставляет два метода для отображения границ между ячейками таблицы: *раздельные* или *сливающиеся*. Когда границы раздельные, они рисуются со всех четырех сторон каждой ячейки, и вы можете задавать пространство между ними. В таблице со сливающимися границами они совмещаются так, что видна только одна граница, а промежуток удаляется (рис. 18.8).

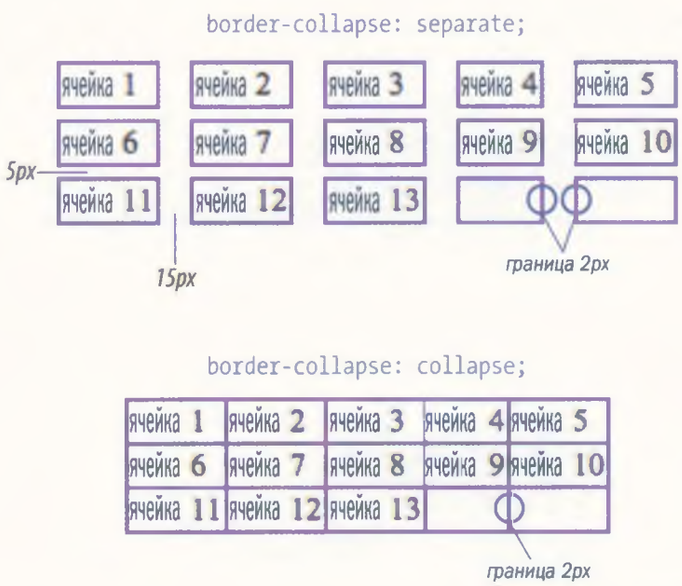


Рис. 18.8. *Раздельные (верхнее изображение) и сливающиеся границы (нижнее изображение)*

Свойство **`border-collapse`** позволяет верстальщикам выбирать, какой из этих методов отображения границ использовать.

`border-collapse`

Принимаемые значения: **`separate`** | **`collapse`** | **`inherit`**

Значение по умолчанию: **`separate`**

Применение: к табличным и встроенным табличным элементам

Наследование: да

Модель раздельных границ

Таблицы отображаются с раздельными границами по умолчанию, как показано на рис. 18.8 (вверху). Вы можете задать размер промежутка между ячейками при помощи свойства **`border-spacing`**.

`border-spacing`

Принимаемые значения: длина | **`inherit`**

Значение по умолчанию: `0`

Применение: к табличным и встроенным табличным элементам

Наследование: да

Для определения свойства **border-spacing** указываются два значения длины. Значение по горизонтали указывается первым и применяется между столбцами. Второе применяется между строками. Если вы зададите одно значение, оно будет использоваться как по горизонтали, так и по вертикали. Значением по умолчанию является 0, вызывающее слияние внутренних границ таблицы.

Ниже приведены правила стилей, используемые для создания пользовательских расстояний между границами, показанных на рис. 18.8 (вверху).

```
table {
border-collapse: separate;
border-spacing: 15px 3px;
border: none; /* нет границ вокруг самой таблицы */
}
td {
border: 2px solid purple; /* границы вокруг ячеек */
}
```

Модель сливающихся границ

Когда выбрана модель сливающихся границ, только одна из них появляется между ячейками. Ниже приведена таблица стилей, создающая таблицу, показанную на рис. 18.8 (внизу).

```
table {
border-collapse: collapse;
border: none; /*нет границ вокруг самой таблицы */
}
td {
border: 2px solid purple; /* границы вокруг ячеек */
}
```

Обратите внимание, что, хотя каждая ячейка таблицы имеет 2-пиксельную границу, итоговая толщина границы между ячейками будет два пиксела, а не четыре. Границы ячеек центрированы между ними, так что если задана 4-пиксельная граница, два пиксела выпадут на одну ячейку и два пиксела — на другую. Для нечетных пиксельных значений браузер решает, на какой стороне отобразится лишний пиксел.

В примерах, где соседние ячейки имеют разные стили границ, используется довольно запутанный алгоритм, чтобы определить, какую границу отображать. Если свойство **border-style** имеет значение **hidden** для одной из ячеек, граница не будет отображаться. Далее рассматри-

ПРИМЕЧАНИЕ

Хотя значением по умолчанию свойства **border-spacing** является ноль, браузеры добавляют два пиксела пространства для атрибута **cellspacing**. Если хотите увидеть эффект дублирования границ, вам нужно установить для атрибута **cellspacing** значение 0 в элементе **table**.

вается ширина границ: более широкие имеют преимущество над более узкими. Наконец, при прочих равных, все сводится к вопросу стилей. Разработчики CSS расположили по ранжиру стили границ от наиболее до наименее приоритетных: **double**, **solid**, **dashed**, **dotted**, **ridge**, **outset**, **groove** и (с самым низким приоритетом) **inset**.

Пустые ячейки

В таблицах с раздельными границами вы можете обозначить при помощи свойства **empty-cells**, отображать ли фон и границы для пустых ячеек..

empty-cells

Принимаемые значения: **show** | **hide** | **inherit**

Значение по умолчанию: **show**

Применение: к элементам ячеек таблицы

Наследование: да

Для того чтобы считаться «пустой», ячейка не должна содержать никакого текста, изображения или неразрывного пробела. Она может содержать только возврат каретки и символы пробела.

На рис. 18.9 показан предыдущий пример таблицы с раздельными границами и пустыми ячейками (под номерами 14 и 15), для которых установлено значение **hide**.

```
table {
border-collapse: separate;
border-spacing: 15px 3px;
empty-cells: hide;
border: none;
}
td {
border: 1px solid purple;
}
```

Рис. 18.9. Сокрытие пустых ячеек с помощью свойства **empty-cells**

Основы адаптивного веб-дизайна

Адаптивный веб-дизайн — это метод, в котором CSS используется для изменения макета страницы в зависимости от размера экрана. Это только одна из стратегий, применяемых, чтобы справиться с разнообразием размеров экранов устройств.

Конечно, адаптивный дизайн — большая, объемная, сложная тема, предназначенная для отдельной книги. Здесь я познакомлю вас с основными ингредиентами адаптивного сайта, чтобы вы почувствовали, как его создавать. Подход, представленный в этой книге, во многом основан на методе адаптивного дизайна, описанном Итаном Маркоттом в его знаменитой книге «Отзывчивый веб-дизайн» (Манн, Иванов и Фербер, 2012). К тому времени, как вы ее прочтете, я уверена, появится еще много замечательных работ на эту тему, не говоря уже об обилии информации во Всемирной паутине (см. также врезку «Для дополнительного чтения» в конце этой главы). И можно сказать, что после завершения упражнений этого раздела ваш путь к овладению адаптивным веб-дизайном только начнется.

Простой пример

В этом разделе мы вместе поработаем над превращением страницы сайта «Малышок» в адаптивную. На рис. 18.10 показано, как одна и та же HTML-страница будет выглядеть на небольшом экране смартфона, на экране планшета в книжной и альбомной ориентации и на большом мониторе компьютера.

Рис. 18.10. Новый адаптивный дизайн сайта «Малышок»

ПРИМЕЧАНИЕ

Более вдохновляющие адаптации можно увидеть в галерее медиазапросов на сайте **mediaqueri.es**.

На экране смартфона у страницы очень узкие боковые поля. На планшетных компьютерах в книжной ориентации снизу появляется внушительный объем пустого пространства. При альбомной ориентации планшета контент прекрасно вписывается в размеры страницы. На настольном компьютере с очень высоким разрешением ширина контента ограничивается свойством `max-width`, гарантирующим, что длина строк не выйдет из-под контроля. Это очень скромные настройки по сравнению с профессионально разработанными адаптивными сайтами, но их должно быть достаточно, чтобы показать вам такой дизайн в действии.

Как это работает?

Адаптивный дизайн, впервые предложенный господином Маркоттом, состоит их трех основных компонентов:

Жидкий макет

Вы узнали все о жидких макетах в главе 16, и к счастью, сайт «Малышок» уже разработан как жидкий.

Гибкие изображения

Когда масштаб макета уменьшается, изображения и другие встроенные мультимедийные элементы должны уменьшаться вместе с ним, иначе они исчезнут из виду. Мы убедимся, что изображения на сайте «Малышок» масштабируются до нужных размеров.

Медиазапросы CSS

Медиазапросы — это способ применения стилей на основе среды, с помощью которой отображается документ. Запросы начинаются с вопросов типа «Будет ли документ распечатываться? Тогда используйте стили для печати». Или «Документ отображается на экране, размер которого не менее 1024 пикселов в ширину, и в альбомной ориентации? Тогда примените *эти* стили». Чуть позже я покажу, как такие запросы выглядят в синтаксисе CSS.

К этому списку ингредиентов я хотела бы добавить элемент области просмотра `meta`, который приводит ширину страницы в соответствие с шириной экрана, и с этого мы начнем наш адаптивный проект.

Настройка области просмотра

Чтобы уместить стандартные веб-сайты на небольших экранах, мобильные браузеры визуализируют страницу на холсте, называемом *областью просмотра*, а затем сжимают эту область до размеров *ширины экрана устройства*. Например на смартфоне iPhone браузер Safari задает ширину области просмотра 980 пикселов так, что веб-страница отображается, словно в окне браузера шириной 980 пикселов, открытом на настольном компьютере. Но отображение сжимается до 640 пикселов

в ширину, запихивая большой объем информации в крошечное пространство, когда iPhone находится в книжной ориентации.

В браузере Safari на устройствах под управлением операционной системы iOS был введен тег **meta** для исходной области просмотра, который позволяет разработчикам управлять ее размером. Вскоре другие мобильные браузеры последовали этому примеру, что стало важным первым шагом на пути к адаптивному дизайну. Добавьте следующий элемент **meta** в раздел заголовка HTML-документа:

```
<meta name="viewport" content="width=device-width, initial-
scale=1">
```

Эта строка указывает браузеру задать ширину области просмотра равную ширине экрана устройства (**width=device-width**), какой бы она ни была. Атрибут **initial-scale** задает уровень масштабирования — 1 (100%).

Теперь, кажется, самое время придать сайту «Малышок» черты адаптивного. Мы будем совершать по одному шагу за раз, начиная с упражнения 18.1.

ПРИМЕЧАНИЕ

Элемент **meta** области просмотра также позволяет применять атрибут **maximum-scale**. Присвоение ему значения 1 (**maximum-scale=1**) не позволит пользователям масштабировать страницу, но настоятельно рекомендуется этого не делать, так как изменение размера имеет важное значение для обеспечения доступности и удобства использования.

УПРАЖНЕНИЕ 18.1. НАСТРОЙКА РАЗМЕРА ОБЛАСТИ ПРОСМОТРА

В этом упражнении вы познакомитесь с материалами сайта «Малышок» и настроите область просмотра, прежде чем мы перейдем к редактированию стилей. Файлы *malishok-rwd.html* и *malishok.css* находятся в папке с материалами к данной главе. Возможно, вы узнали страницу из предыдущих упражнений, но я немного изменила стили, чтобы вам проще было начать.

1. Откройте файл *malishok-rwd.html* в браузере. Таблица *malishok.css* охватывает основные стили, такие, как фон, цвета, границы и стили текста, обеспечивая хорошее первое впечатление. Измените размер окна на очень узкий, чтобы приблизить его к ширине экрана смартфона. Вы должны увидеть что-то похожее на снимок экрана iPhone, показанный на рис 18.10, за исключением того, что графический логотип «Малышок» выходит за пределы правого края экрана. Прокрутите вниз, и вы увидите, что блоки **#products** и **#testimonials** сместились к самому краю окна.
2. Теперь измените размер окна на максимально широкий. Вы увидите, что страница некрасиво растянулась и текст не обтекает изображения товаров. К этому проекту явно нужно отнестись иначе, чтобы он смотрелся лучше в широком окне браузера.
3. Давайте добавим элемент **meta** области просмотра. Откройте файл *malishok-rwd.html* в текстовом редакторе и добавьте стандартный элемент **meta**, как показано здесь:

```
<meta name="viewport" content="width=device-width,
initial-scale=1">
```

Сохраните файл. Поскольку это версия для мобильных устройств, вы не заметите изменений, когда снова просмотрите страницу в браузере настольного компьютера, но будете знать, что основа для улучшений заложена.

ПРИМЕЧАНИЕ

Браузер Internet Explorer 8 и более ранние версии программы не поддерживают медиазапросы, которые мы применим далее.

Адаптивный макет

В качестве альтернативного подхода, особенно когда нет времени или средств для по-настоящему адаптивной переработки сайта, некоторые дизайнеры решают вместо этого создать *адаптивный макет*. Такие макеты состоят из двух или трех различных фиксированных макетов, ориентированных на наиболее распространенные устройства. Возможно, их создание проходит быстрее и не столь трудоемко, но преимущества жидких макетов будут потеряны. Некоторые считают, что адаптивные макеты — это, скорее, временное решение, чем долгосрочная стратегия мобильного дизайна.

Жидкие макеты

Поскольку жидкие макеты являются основополагающими для адаптивного дизайна, я думаю, не помешает их краткое описание. Жидкие макеты создаются с помощью процентных значений ширины так, что размер элементов изменяется пропорционально, чтобы заполнить доступную ширину экрана или окна.

Невозможно создать дизайны для всех возможных вариантов ширины устройства, на котором будет просмотрена ваша страница. Веб-дизайнеры обычно готовят два или три проекта (иногда несколько), ориентированных на основные классы устройств, таких как смартфоны, планшеты и настольные браузеры, полагая, что жидкий макет справится со всеми промежуточными размерами. Жидкие макеты стараются не оставлять неудобно большое ненужное пространство и не позволяют обрезать правую часть страницы.

Поскольку я выбрала для этого проекта стили жидкого макета из предыдущих упражнений, нам ничего не нужно менять в стилях сайта «Малышок». Для ваших собственных проектов, однако, обязательно создавайте гибкие дизайны. И если говорить о гибком, давайте сделаем что-нибудь с логотипом!

Обеспечение гибкости изображений

Иногда решение оказывается простым. Возьмем, например, правило стилей, необходимое, чтобы уменьшать размер изображений в соответствии с их контейнером:

```
img {
max-width: 100%;
}
```

Когда макет уменьшается, изображения в нем масштабируются до ширины контейнера, в котором они находятся. Если контейнер больше, чем изображение, например, в макетах планшетных или настольных компьютеров, то оно не увеличивается, а останавливается на 100% от своего начального размера. При применении свойства **max-width** убедитесь, что в элементе **img** HTML-документа не указаны атрибуты **width** и **height**, иначе изображение не будет изменяться пропорционально.

Но подождите, не бывает все так легко, верно? Я боюсь, что, хотя правило стилей простое, в целом проблема отображения изображений на мобильных устройствах более сложная. Даже в нашем скромном примере мы отправляем на смартфон изображение большего размера, чем нужно, а значит, передаются лишние данные. Я собираюсь вернуться к этой задаче чуть позднее, в разделе «Адаптивные изображения». А пока просто помните о ней.

Прежде чем перейти к упражнению, я должна также отметить, что с помощью свойства **max-width** вы можете уменьшать масштаб и других встроенных мультимедийных элементов таких, как **object**, **embed** или **video** (см. примечание).

ПРИМЕЧАНИЕ

Для сохранения пропорций уменьшенного видеоролика придется потрудиться. Тьерри Кобленц описал эту стратегию в своей статье на сайте **www.alistapart.com/articles/creatingintrinsic-ratios-for-video**. Кроме того, решение с использованием JavaScript представлено на сайте **fitvidsjs.com**.

УПРАЖНЕНИЕ 18.2. ОБУЗДАЙТЕ ИЗОБРАЖЕНИЯ

Это упражнение короткое. Откройте файл *malishok.css* и добавьте правило для изменения размера изображений в таблицу стилей сразу после набора правил для элемента **body**.

```
img { max-width: 100%; }
```

Сохраните файл и перезагрузите страницу в браузере. Теперь, когда вы сделаете окно очень узким, размер логотипа уменьшится вместе с ним (рис. 18.11). С изображениями товаров произойдет то же самое, но, возможно, у вас не получится сделать область просмотра достаточно узкой, чтобы это заметить.

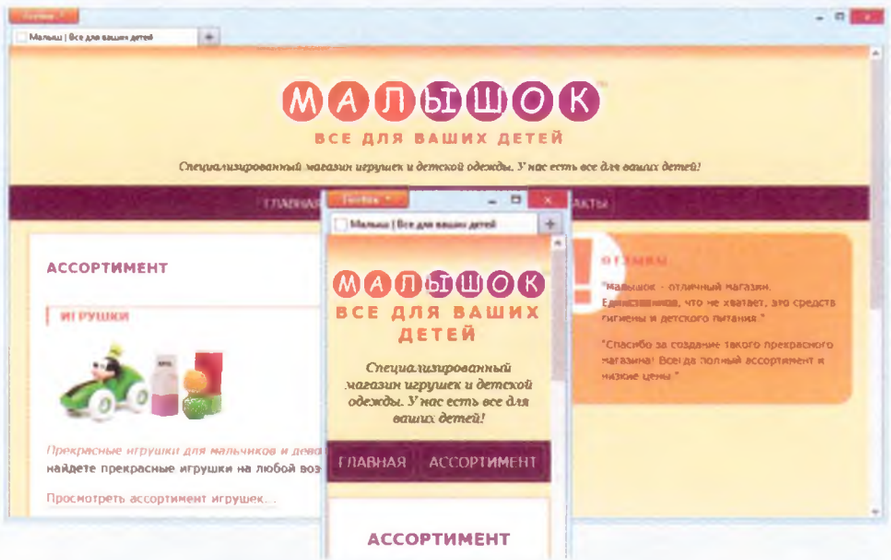


Рис. 18.11. *Свойство* **max-width** *заставляет изображение пропорционально сокращаться при уменьшении его контейнера*

Использование медиазапросов

Теперь мы перейдем к сути адаптивного дизайна: медиа-запросам. Они позволяют дизайнерам применять стили в зависимости от типа устройства. К известным типам относятся **print**, **speech**, **handheld**, **braille**, **projection**, **screen**, **tty** и **tv**. Зарезервированное слово **all** указывает, что стиль применяется ко всем типам устройств. Медиазапросы могут также определить характеристики устройств, например, **device-width**, **orientation** и **resolution**. У большинства характеристик можно выявить максимальные и минимальные значения, используя префиксы **min-** и **max-** соответственно. Например **min-width: 480px** проверяет, составляет ли ширина экрана хотя бы 480 пикселов. Экраны шириной 768 пикселов проходят проверку, и к ним применяются стили, а экраны шириной 320 пикселов — нет.

Полный список характеристик устройств, которые можно определить с помощью медиазапросов, приведен в табл. 18.1.

Медиазапросы можно добавить в таблицу стилей. Ниже приведен пример таблицы стилей с медиа-запросом, который определяет, является ли устройство экраном и составляет ли его ширина как минимум 480 пикселов:

```
@media screen and (min-width: 480px;) {
/* поместите стили для устройств и браузеров, отвечающих требованиям, внутри фигурных скобок */
}
```

Запрос начинается с **@media**, затем указывается зарезервированное слово, обозначающее тип целевого устройства (в данном случае **screen**). Характеристика устройства и тестируемое значение заключены в скобки.

Табл. 18.1. Характеристики устройств, которые можно определить с помощью медиазапросов

Характеристика	Описание
width	Ширина области отображения (области просмотра)
height	Высота области отображения (области просмотра)
device-width	Ширина отображающей поверхности устройств (весь экран)
device-height	Высота отображающей поверхности устройств (весь экран)
orientation	Установлена ли на устройстве книжная или альбомная ориентация (не принимает префиксы **min-**/**max-**)
aspect-ratio	Соотношение сторон области просмотра, полученное делением ширины на высоту (ширина/высота)
device-aspect-ratio	Соотношение ширины и высоты всего экрана (отображающей поверхности)
color	Битовая глубина отображения, например, значение **color: 8** проверяет, доступен ли на экране устройства хотя бы 8-битный цвет
color-index	Количество цветов в таблице цветов
monochrome	Число фрагментов, содержащихся в одном пикселе на монохромном устройстве
resolution	Плотность пикселов в устройстве. Эта характеристика приобретает все большее значение для обнаружения экранов с высоким разрешением
scan	Определяет, используется ли в типе носителя **tv** прогрессивное или чересстрочное сканирование (не принимает префиксы **min-**/**max-**)
grid	Определяет, применяет ли устройство отображение с сетчатой основой, такое как шрифт фиксированной ширины (не принимает префиксы **min-**/**max-**)

Правила стилей для браузеров, отвечающих требованиям, указываются между фигурными скобками.

Ниже представлен еще один пример, который проверяет два значения характеристики: составляет ли ширина экрана менее 700 пикселов и находится ли устройство в альбомной ориентации. Обратите внимание, что каждая пара «характеристика-значение» заключена в скобки. Слово «and» связывает различные требования вместе. Устройство должно

отвечать всем требованиям, чтобы к нему были применены стили, заключенные в фигурных скобках.

```
@media screen and (max-width: 700px;) and (orientation:
landscape;) {
/* поместите стили для устройств и браузеров, отвечающих
требованиям, внутри фигурных скобок */
}
```

Наконец в примере ниже медиазапрос выясняет, имеет ли устройство экран с высокой плотностью пикселов типа Retina (iPhone, iPad и новые версии MacBook Pro). В этом примере содержится не только стандартный запрос, но и вендорные префиксы. Отдельные запросы приведены в списке через запятую. Стили, заключенные в фигурных скобках, применяются, если удовлетворены условия любого из запросов.

```
@media screen and (-webkit-min-device-pixel-ratio: 2),
screen and (-moz-min-device-pixel-ratio: 2),
screen and (-o-min-device-pixel-ratio: 2),
screen and (-ms-min-device-pixel-ratio: 2),
screen and (min-device-pixel-ratio: 2) {
/* здесь приводятся стили, относящиеся к экранам с высоким
разрешением */
}
```

Медиазапросы в разделе заголовка документа

Запросы **@media**, которые мы рассматривали выше, включены в саму таблицу стилей. Медиазапросы могут также осуществляться с помощью атрибута **media** в элементе **link**, чтобы загружать отдельные файлы *.css*, если условия выполняются. В этом примере сначала запрашиваются основные стили для сайта, а затем — таблица стилей, которая будет использоваться только в том случае, если ширина экрана устройства превышает 780 пикселов (и если браузер поддерживает медиазапросы).

```
<head>
<link rel="stylesheet" href="styles.css">
<link rel="stylesheet" href="2column-styles.css"
media="screen and (min-width:780px)">
</head>
```

Некоторые разработчики считают этот метод полезным для управления модульными таблицами стилей, но у него есть недостатки — дополнительные http-запросы для каждого нового файла *.css*. Обязательно указывайте только необходимые ссылки (возможно, по одной для каждой основной контрольной точки) и прибегайте к правилам **@media** в таблицах стилей для внесения небольших корректив в промежуточные размеры*.

ПРЕДУПРЕЖДЕНИЕ

Браузер Internet Explorer 8 и его более ранние версии не поддерживают медиа-запросы. Я покажу вам обходной путь в упражнении 18.3.

* Этот метод был предложен Стефани Ригер в презентации «Pragmatic Responsive Design». Слайды можно просмотреть по адресу **www.slideshare.net/yiibu/pragmatic-responsive-design**

ПРИМЕЧАНИЕ

Хороший обзор подхода к дизайну по принципу «сначала мобильные» вы найдете в статье Брэда Фроста по адресу **www.html5rocks.com/en/mobile/responsivedesign/** и связанным с ней комментарием **bradfrostweb.com/blog/mobile/anatomy-of-a-mobilefirst-responsive-web-design/**, которые описывают размышления, предшествовавшие созданию каждого компонента в демонстрации. Это замечательная возможность заглянуть в мысли веб-дизайнера для мобильных устройств.

ПРЕДУПРЕЖДЕНИЕ

Убедитесь, что стили заключены в фигурные скобки и скобки закрыты правильно. Легко пропустить последнюю фигурную скобку, которой заканчивается медиазапрос.

«Сначала мобильные» медиазапросы

Лучшая практика для адаптивных сайтов — усвоить принцип «сначала мобильные». Это значит, что в первую очередь вы заботитесь о стилях для устройств с самым маленьким экраном и простыми характеристиками и используете медиазапросы для переопределения стилей, которые адаптируют дизайн по мере того, как становится доступно больше пространства экрана и характеристик. Медиазапросы по принципу «сначала мобильные» обычно начинаются с префикса **min-**, добавляя новые стили, когда ширина составляет *не менее* указанной или более. Это позволяет разработчикам наслаивать стили, основываясь на уже примененных более простых.

Помните, что стили, расположенные ниже в списке, переопределяют предшествующие им, будь то правила в одной таблице стилей или список элементов **link**. Должно получиться так, что основные стили идут первыми, за ними следуют стили для простых устройств с малой диагональю экрана, а затем улучшенные стили для более крупных окон браузеров. Именно так мы и поступим в упражнении 18.3.

УПРАЖНЕНИЕ 18.3. ДОБАВЛЕНИЕ МЕДИАЗАПРОСОВ

Теперь можно приступить к работе по добавлению стилей, которые будут изменять макет согласно ширине экрана. Я потрудилась над дизайном вместо вас. Вы можете скопировать готовые стили отсюда или взять их из *файла malishok-final.css* в папке материалов для этой главы.

1. Откройте файл *malishok.css* в текстовом редакторе. Текущая таблица стилей создает дизайн в одну колонку от края до края, который прекрасно работает на узких экранах, но выглядит скудно, когда становится шире. Я решила, что он замечательно подойдет для смартфонов в книжной и альбомной ориентации (до 960 пикселов в ширину), но при этом хочу добавить немного больше свободного пространства.
2. Я начну с добавления стилей для устройств шириной не менее 481 пиксела. Имея немного дополнительного пространства, я смогу переместить изображения продуктов влево с помощью свойства **float**, и применить свойство **clear** к следующим за ними ссылкам: «Подробнее». Я также добавила поля вокруг белого блока **#products** и применила скругленные углы и поля к блоку **#testimonials**, как мы это делали в упражнениях в главах 14 и 15 (рис. 18.12). Получившийся медиазапрос, показанный ниже, отправляется в конец таблицы стилей, так что он может выборочно переопределять свойства указанные перед ним.

```
@media screen and (min-width: 481px) {
#products img {
float: left;
margin: 0 6px 6px 0;
}
#products .more {
clear: left;
}
```

```
#products {
margin: 1em;
}
#testimonials {
margin: 1em 5%;
border-radius: 16px;
}
}
```

Рис. 18.12. Сайт Малышок после применения к нему стилей для планшетного компьютера

Обратите внимание, что мы тестируем ширину области отображения (**width**), а не ширину всего экрана (**device-width**), потому что вокруг веб-страниц, просматриваемых на мобильных устройствах, часто отображается окно браузера Chrome. Тестирование характеристики **width** даст нам более точные результаты.

3. Следующий набор стилей проявляется, когда ширина области отображения не менее 1024 пикселов. Стили в этом медиазапросе создают макет с двумя колонками, смещая плавающий элемент **#products div** влево и применяя широкое левое поле к элементу **#testimonials**. Абзацу с информацией об авторских правах запрещено обтекание с помощью свойства **clear**, и он отображается в нижней части страницы. Наконец я задала свойство **max-width** для элемента **#content div** так, чтобы ширина области контента никогда не превышала 1024 пикселов, даже если браузер будет гораздо шире (рис. 18.13). Этот медиазапрос должен быть указан в документе таблицы стилей под тем, который мы только что добавили.

```
@media screen and (min-width: 780px) {
#products {
float: left;
margin: 0 2% 1em;
clear: both;
width: 55%;
overflow: auto;
}
#testimonials {
margin: 1em 2% 1em 64%;
}
p#copyright {
clear: both;
}
#content {
max-width: 1024px;
margin: 0 auto;
}
}
```

4. Теперь вы можете сохранить документ и открыть его в браузере. Попробуйте изменить размер окна и посмотрите, как макет меняется на ходу. Вы видите свою первую адаптивную веб-страницу!

А что делать с Internet Explorer версии 8 и ниже?

Как я уже упоминала в примечании выше, браузер Internet Explorer версии 8 и более ранних не поддерживает медиазапросы, поэтому содержащиеся в них стили будут игнорироваться. Это значит, что пользователь с браузером Internet Explorer 8 на большом мониторе настольного компьютера увидит версию страницы в одну колонку с наименьшим общим знаменателем.

Решение — поместить стили для настольного браузера в отдельную таблицу стилей, обслуживающую только немобильные версии браузера Internet Explorer, более старые, чем версия 9 (это (**LTIE 9**) и (**!IEMobile**)).

Если хотите продолжить, скопируйте стили из медиазапросов (но не обозначение медиазапросов) и вставьте их в новый файл с именем *ie-layout.css*. Из первого медиазапроса удалите стили для плавающих изображений и скругления углов блока с отзывами. Все стили из второго медиазапроса применимы к настольному браузеру, так что вставьте и их тоже.

Специальный условный комментарий браузера Internet Explorer содержит ссылку на специальную таблицу стилей и должен указываться после ссылок на другие таблицы стилей. Вы можете добавить следующий код в раздел заголовка файла *malishok-rwd.html*.

```
<link rel="stylesheet" href="malishok.css">
<!--[if (lt IE 9)&(!IEMobile)]>
<link rel="stylesheet" href="ie-layout.css">
<![endif]-->
```

Рис. 18.13. *Сайт «Малышок» на различных экранах*

Проблемы

Сайт «Малышок» использует адаптивный дизайн, но очевидно, что он упрощен и представляет собой один из наилучших сценариев. Правильное применение адаптивности требует планирования. Поскольку мобильный Интернет относительно молод, сообщество разработчиков по-прежнему сталкивается с трудностями мобильного дизайна и преодолевает их. Я хочу ввести вас в курс дела касательно некоторых наиболее сложных аспектов и ограничений адаптивного дизайна и мобильного веб-дизайна в целом.

Выбор основных контрольных точек

Одно из первостепенных дизайнерских решений при создании адаптивного дизайна — это определение, на каких значениях ширины вносить существенные изменения в дизайн.

Точка, в которой медиазапрос передает новый набор стилей, известна как *контрольная*. У некоторых сайтов всего два макета, запускаемых

на одной контрольной точке. Чаще адаптивные веб-сайты используют три дизайна, ориентированные на типичную ширину экрана смартфона, планшетного и настольного компьютеров. Сколько вы выберете, зависит от дизайна вашего сайта.

Но как же выбрать контрольные точки? Один из способов заключается в использовании размеров (указанных в пикселах) экранов популярных устройств, как мы делали в упражнении с сайтом «Малышок». На рис. 18.14 показан график контрольных точек, где перечислены параметры экранов наиболее популярных устройств, как в книжной, так и в альбомной ориентации. Реальность такова, что постоянно появляются устройства с новыми значениями разрешения экрана, и не предполагается, что мы будем создавать отдельный дизайн для каждого. Поэтому в медиазапросах мы отошли от использования пиксельных значений в сторону любимой единицы измерения во Всемирной паутине — em. Многие разработчики позволяют контенту самому определять, где должны возникать контрольные точки, которые, если коротко, являются точками, где сайт начинает смотреться действительно некрасиво!

СОВЕТ

Веб-сайт **ResizeMyBrowser.com** делает именно то, что указано в его названии. Щелкните мышью по одному из размеров, перечисленных на экране (320×480, например) — и сайт изменит размер окна вашего браузера на выбранный. Это дает вам поле для тестирования своих макетов при различной ширине устройства. Имейте в виду, однако, что ничто не заменит теста на реальном устройстве! Это всего лишь удобный инструмент, используемый в процессе дизайна.

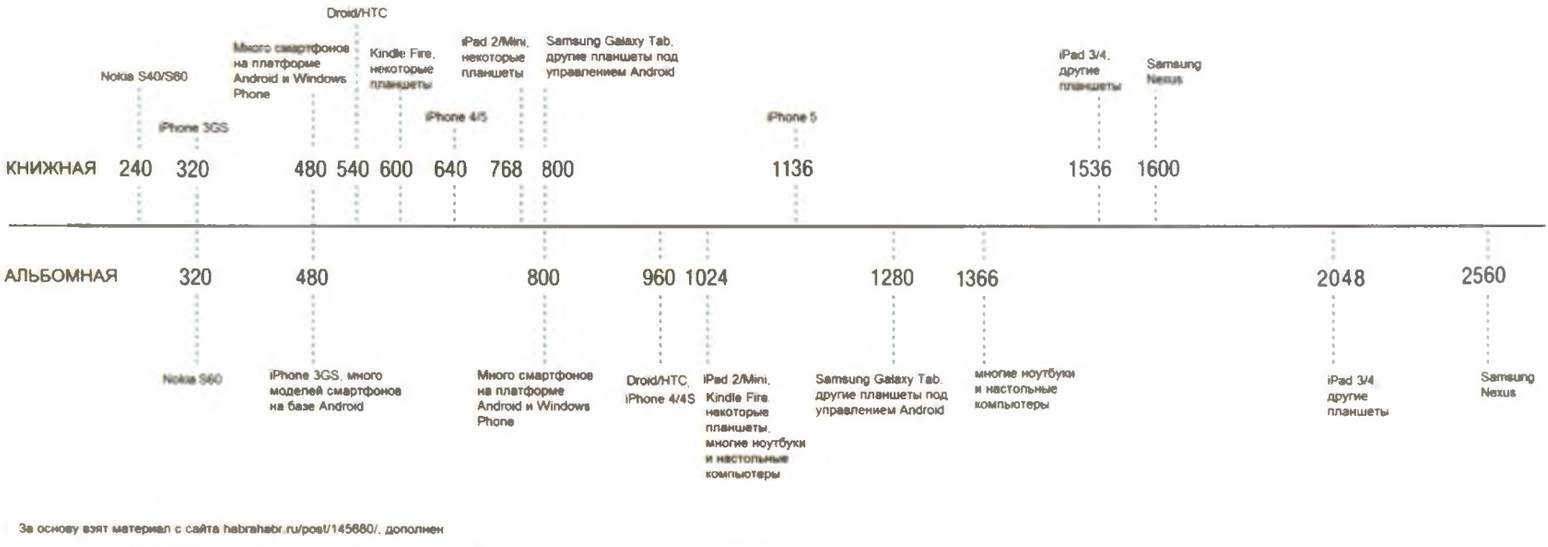


Рис. 18.14. *Этот график контрольных точек показывает в пикселах ширину экранов некоторых популярных устройств*

Учтите на будущее, что более удобным становится подход, заключающийся в продумывании дизайнов в одну колонку, в одну широкую колонку и в несколько колонок, а затем определение логических контрольных точек для каждого. Чтобы узнать больше, я рекомендую вам прочитать статью Лизы Гарднер по адресу **blog.cloudfour.com/the-ems-have-it-proportional-media-queries-ftw/**.

Адаптивные изображения

Одна из самых неприятных проблем, стоящих перед разработчиками сайтов под мобильные устройства, — это как добиться правильных изображений. В идеале устройство должно загружать изображения только того размера, который подходит для его величины и скорости работы

Изменение размеров изображения на сервере

Sencha.io Src — это сервис, который сжимает ваши изображения в реальном времени и передает их в нужном размере на запрашивающее устройство. Все, что вам нужно сделать, это добавить немного дополнительной разметки к тегу `img`, который отправляет изображение на сервер Sencha.io Src.

Сервер Sencha.io Src использует *строку пользовательского агента* (данные, которые используют браузеры, чтобы идентифицировать себя), чтобы найти в базе данных это устройство. Как только ширина его экрана определена, Sencha.io Src уменьшает масштаб изображения до данного значения и отправляет обратно файл меньшего размера. Подробнее см. на сайте **docs.sencha.io/0.1.3/index.html#!/guide/src**.

сети. Цель состоит в том, чтобы избежать загрузки ненужных данных, например изображения большего размера, чем нужно для маленького экрана, или двух версий изображения (с низким и высоким разрешением), когда требуется только одна.

Сложность с изображениями в том, что знание размера устройства не обязательно сообщит вам что-либо о скорости сети. Простые смартфоны могут использовать медленные сети EDGE или быстрые WiFi. Планшеты iPad с технологией Retina загружают большие изображения, но могут использовать для этого сеть 3G. Кроме того, возможно, вы просто не захотите уменьшать масштаб изображения для маленького экрана. В некоторых случаях предпочтительнее использовать совершенно другое изображение, обрезанное так, чтобы показать важные детали при меньших размерах.

На момент написания книги было больше споров на эту тему, чем решений. Кое-кто предлагал новую HTML-разметку, упрощающую указание файлов изображений на основе размеров и разрешения экрана. Некоторые считают, что сервер должен играть более значимую роль, особенно при обсуждении скорости сети. Другие полагают, что ответом является новый графический формат, который может содержать несколько версий одного и того же изображения. Внезапный рост популярности мобильной Всемирной паутины застал наши технологии врасплох.

Поиск во Всемирной паутине по запросу «адаптивные изображения» должен помочь вам получить последние новости о текущей ситуации. Также прочитайте следующие публикации:

- «HTML5: Адаптивные изображения» (**habrahabr.ru/post/145376/**)
- «Адаптивные фоновые изображения» (**codezona.com/adaptivnie-fonovie-izobragenia.html**)
- «Какое выбрать решение для адаптивных изображений?» (**css-live.ru/articles/kakoe-vybrat-reshenie-dlya-adaptivnyx-izobrazhenij.html**)
- «Адаптивные изображения для сайтов в стиле Responsive Web Design» (**wedeal.ru/blog/adaptivnyie-izobrazheniya-dlya-saytov-v-stile-responsive-web-design/**)
- «Восемь рекомендаций и одно правило для адаптивных изображений» (**webformyself.com/vosem-rekomendacij-i-odno-pravilo-dlya-adaptivnyx-izobrazhenij/**)
- «Краткий курс — «Адаптивный сайт за неделю»: изображения и видео» (**www.cmsmagazine.ru/library/items/graphical_design/build-responsive-site-week-images-and-video-part-3/**)

Один размер всем не подходит

Каскадные таблицы стилей отлично работают при замене стилей и перемещении элементов по экрану (или даже их сокрытии). Однако во многих случаях для небольших устройств лучше предоставить другой контент или тот же, но в другом порядке. Технология JavaScript может

справиться с некоторыми перестановками и предлагает способ условной загрузки контента. Настройка контента с помощью JavaScript выходит за рамки этого раздела, но вы должны знать, что такие настройки возможны и должны рассматриваться при дизайне для мобильных устройств.

Адаптивные ограничения

Для некоторых веб-сайтов, в частности, нагруженных текстом, таких как блоги, адаптивного перепроектирования достаточно, чтобы превратить их в сайты, которые приятно использовать на устройствах с малой диагональю экрана. Для других сайтов, однако, недостаточно просто настроить стили. Когда использование сайта или сервиса на мобильном устройстве существенно отличается от использования его в настольном браузере (на основе тестирования пользователями, конечно), может потребоваться создать отдельный мобильный сайт.

Но даже отдельные мобильные сайты могут выиграть от использования основных компонентов адаптивного дизайна, рассмотренных нами. Адаптивные методы доказали, что они имеют большое значение как важные навыки для любого веб-дизайнера.

Заключение

Мы подошли к концу изучения таблиц стилей. К этому времени вы должны свободно форматировать текст и даже компоновать макет страницы средствами CSS. Секрет овладения таблицами стилей, конечно, кроется в большом количестве практики и исследований. Если вы увлечетесь этим делом, то обнаружите, что существует много онлайн-ресурсов, позволяющих найти ответы на ваши вопросы.

В части IV мы поговорим об изображениях для Всемирной паутины. А в части V я уступлю клавиатуру эксперту в области JavaScript Мэту Маркусу, который познакомит вас с этим языком и его синтаксисом.

Резюме

Ниже приведена сводка свойств, рассмотренных в этой главе.

Свойство	Описание
`border-collapse`	Определяет, являются ли границы между ячейками раздельными или сливающимися
`border-spacing`	Задает пространство между ячейками при их раздельном отображении (`separate`)
`empty-cells`	Определяет, отображаются ли границы и фон пустых ячеек

Для дополнительного чтения

Всемирная паутина — лучшее средство, чтобы оставаться в курсе разработок в области адаптивного дизайна потому, что этот материал меняется с огромной скоростью. В качестве наиболее полного ресурса я рекомендую сайт Брэда Фроста (**mobilewebbestpractices.com**). В разделе «Resources» Брэд собрал списки лучших статей, книг, галерей, презентаций, сценариев и прочего, что имеет отношение к разработке веб-сайтов для мобильных устройств.

По этой теме существует множество книг, из которых я порекомендую «Сначала мобильные!» Люка Вроблевски (Манн, Иванов и Фербер, 2012), а также «Веб-программирование для мобильных устройств» Максимилиано Фиртмана (Рид Групп, 2012).

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Руководство для начинающих по HTML5, CSS3, JavaScript и графическим изображениям для веб

Независимо от того, новичок вы или опытный веб-дизайнер, эта книга научит вас основам современного веб-дизайна. Если вы интересуетесь веб-дизайном, эта книга — превосходное начало.

В этой книге вы найдете все, что необходимо знать для создания отличных веб-сайтов. Начав с изучения принципов функционирования Интернета и веб-страниц, к концу книги вы освоите приемы создания сложных сайтов, включая таблицы стилей CSS и графические файлы, и научитесь размещать страницы во Всемирной паутине. Книга включает упражнения, с помощью которых вы освоите разнообразные техники работы с современными веб-стандартами (включая HTML5 и CSS3).

Вы научитесь:

- создавать HTML-страницы и добавлять на них текст, ссылки, изображения, таблицы и формы
- работать с каскадными таблицами стилей — форматировать текст, устанавливать цвета и фон, компоновать макеты страниц и создавать простые анимационные эффекты
- использовать новые для HTML5 элементы, API-интерфейсы и свойства CSS3 при верстке веб-страниц
- подготавливать страницы для отображения на мобильных устройствах посредством применения техник адаптивного веб-дизайна.
- работать с языком JavaScript, разберетесь, почему он так важен для веб-дизайна
- создавать и оптимизировать графические изображения для веб

Вы хотите научиться самым современным технологиям, текущим стандартам создания сайтов и познакомиться с лучшими методами? Тогда эта книга для вас!

ISBN 978-5-699-67603-3

Об авторе

За плечами Дженнифер Роббинс более 20 лет опыта преподавания и разработки проектов в сфере веб-дизайна. Дженнифер спроектировала свой первый коммерческий веб-сайт в 1993 году и с тех пор стала признанным экспертом в области веб-дизайна. Кроме того, она читает веб-дизайна в Массачусетском колледже искусств в Бост Джонсона и Уэльса в Провиденсе в Род-Айленде.