

O'REILLY®

bhv®

5-е издание

Веб- ДИЗАЙН для начинающих

HTML, CSS, JavaScript и веб-графика

Дженнифер Нидерст Роббинс

Fifth Edition

LEARNING WEB DESIGN

A BEGINNER'S GUIDE TO HTML, CSS,
JAVASCRIPT, AND WEB GRAPHICS

Jennifer Niederst Robbins



Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

5-е издание

Веб-ДИЗАЙН для НАЧИНАЮЩИХ

HTML, CSS, JavaScript и веб-графика

Дженнифер Нидерст Роббинс

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.43
ББК 32.973-018.1
P58

Роббинс Дж.

P58 Веб-дизайн для начинающих. HTML, CSS, JavaScript и веб-графика. — 5-е изд.;
пер. с англ. — СПб.: БХВ-Петербург, 2021. — 956 с.: ил.

ISBN 978-5-9775-4050-6

Книга поможет освоить веб-дизайн, не имея опыта. На практических примерах показано, как создать простой сайт и постепенно его совершенствовать. Рассказано о создании веб-страниц, содержащих текст, ссылки, изображения, таблицы и формы. Описано применение CSS для создания и выбора цвета, фона, форматирования текста, макетирования страниц и выполнения простой анимации. Даны основы языка JavaScript и подчеркнута его важность в веб-дизайне. Описано создание, оптимизация и сокращение времени загрузки веб-изображений.

В каждой главе представлены упражнения, которые позволяют освоить описанные методики, и краткие контрольные вопросы для закрепления ключевых понятий.

В пятом издании добавлен материал об использовании CSS Flexbox и Grid для создания сложных и гибких макетов страниц, тонкостях адаптивного веб-дизайна для отображения веб-страниц на экранах любых устройств, о работе с командной строкой, Git и другими инструментами веб-дизайнера, а также применении SVG-изображений

Для веб-дизайнеров

УДК 004.43
ББК 32.973-018.1

Группа подготовки издания:

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Екатерина Сависте</i>
Компьютерная верстка	<i>Людмила Гауль</i>
Оформление обложки	<i>Карина Соловьевой</i>

© 2021 BHV

Authorized Russian translation of the English edition of *Learning Web Design 5th edition* ISBN 9781491960202

© 2018 O'Reilly Media Inc.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

© 2021 BHV

Авторизованный перевод с английского языка на русский издания *Learning Web Design 5th edition*
ISBN 9781491960202 © 2018 O'Reilly Media Inc.

Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc.

«БХВ-Петербург», 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-1-491-96020-2 (англ.)
ISBN 978-5-9775-4050-6 (рус.)

© O'Reilly Media, Inc., 2018
© Перевод на русский язык, оформление, ООО «БХВ-Петербург»,
ООО «БХВ», 2021

ОГЛАВЛЕНИЕ

НАПУТСТВИЕ от Джен Симмонс	9
ПРЕДИСЛОВИЕ	11
Структура книги	14
Типографские соглашения	16
Благодарности	16
Об авторе	17
Как с нами связаться?	18
ЧАСТЬ I. НАЧАЛО РАБОТЫ.....	19
Глава 1. Знакомство с веб-дизайном	21
С чего начать?	22
Распределение ролей при разработке сайтов	23
Подготовка к веб-разработке	36
Подведем итоги	41
Контрольные вопросы	42
Глава 2. Как все это работает?.....	43
Связь между Интернетом и Всемирной паутиной	43
Порядок обработки информации	44
Пара слов о браузерах	45
Адреса веб-страниц: URL	47
Анатомия веб-страницы	51
Собираем все вместе	57
Контрольные вопросы	60
Глава 3. Важные концепции	61
Множественность устройств	62
Соблюдение интернет-стандартов	64
Постепенное улучшение	64
Адаптивный веб-дизайн	66
Один Интернет для всех (доступность сайтов)	69
Потребность в скорости (быстродействие сайта)	72
Контрольные вопросы	75
ЧАСТЬ II. РАЗМЕТКА HTML, ПРИМЕНЯЕМАЯ ДЛЯ СТРУКТУРИРОВАНИЯ ВЕБ-СТРАНИЦ	77
Глава 4. Создание простой веб-страницы	79
Пошаговое создание веб-страницы	79
Запуск текстового редактора	80
Шаг 1. Начнем с ввода контента	84
Шаг 2. Добавление структуры в HTML-документ	87

Шаг 3. Добавление тегов в HTML-документ.....	92
Шаг 4. Добавление изображений.....	96
Шаг 5. Изменение внешнего вида текста с помощью таблицы стилей.....	100
Устранение проблем при разработке веб-страниц	101
Валидация документов.....	103
Контрольные вопросы	105
Глава 5. Разметка текста	107
Абзацы.....	108
Заголовки.....	109
Тематические разрывы (горизонтальные линейки).....	111
Списки	111
Дополнительные элементы разметки контента	116
Организация контента на странице.....	120
Обзор строчных элементов	127
Обобщающие элементы (<i>div</i> и <i>span</i>).....	141
Улучшение доступности документа с помощью ARIA.....	148
Экранирование символов.....	151
Объединяем все вместе.....	154
Контрольные вопросы	158
Обзор элементов: текстовые элементы	158
Глава 6. Добавление ссылок	161
Атрибут <i>href</i>	162
Ссылки на веб-страницы в Интернете	164
Ссылки в пределах сайта.....	165
Ссылка на фрагмент другого документа	176
Открытие новых целевых окон браузера.....	178
Почтовые ссылки	179
Телефонные ссылки	180
Контрольные вопросы	181
Обзор элементов: ссылки.....	182
Глава 7. Добавление изображений	183
Раздел первый: добавление изображений с помощью элемента <i>img</i>	184
Раздел второй: добавление SVG-изображений	194
Раздел третий: разметка адаптивных изображений	203
Конец — делу венец	218
Контрольные вопросы	219
Обзор элементов: изображения	221
Глава 8. Таблицы	223
Использование таблиц.....	223
Минимальная структура таблицы	224
Заголовки таблиц.....	227
Растяжение ячеек.....	228
Доступность таблиц	231

Группировка строк и столбцов.....	233
Собираем все воедино	235
Контрольные вопросы	237
Обзор элементов: таблицы.....	238
Глава 9. Формы.....	239
Как работают формы?.....	239
Элемент form	241
Переменные и контент.....	245
Элементы управления формой	246
Функции доступности формы.....	270
Макет и дизайн формы.....	275
Контрольные вопросы.....	277
Обзор элементов: формы.....	278
Глава 10. Встроенные мультимедийные объекты.....	285
Окно в окне: элемент iframe	285
Многоцелевое средство для встраивания: object	288
Видео и аудио.....	290
Холст.....	301
Контрольные вопросы	305
Обзор элементов: встроенные мультимедийные объекты	306
ЧАСТЬ III. ПРАВИЛА CSS ДЛЯ ПРЕДСТАВЛЕНИЯ HTML-ДОКУМЕНТОВ	309
Глава 11. Введение в каскадные таблицы стилей	311
Преимущества CSS.....	312
Мощь CSS.....	312
Как работают таблицы стилей?	313
Важные понятия.....	320
Единицы измерения CSS.....	330
Инструменты вашего браузера для веб-дизайна	334
Вперед — вместе с CSS.....	336
Контрольные вопросы	337
Глава 12. Форматирование текста	339
Основные свойства шрифтов.....	340
Расширенная CSS3-тиографика.....	361
Свойство color (изменение цвета текста)	363
Еще о типах селекторов.....	365
Настройки текстовых строк	371
Изменение маркированного и нумерованного списков	383
Контрольные вопросы	387
Обзор CSS: свойства font и text	388
Глава 13. Цвета и фоны, а также еще о селекторах и внешних таблицах стилей ..	391
Указание цветовых значений	391
Свойство color (цвет переднего плана).....	398

Свойство <i>background-color</i> (цвет фона)	399
Свойство <i>background-clip</i> (обрезка фона).....	400
Свойство <i>opacity</i> (управление непрозрачностью)	401
Селекторы псевдокласса	403
Селекторы псевдоэлементов.....	408
Селекторы атрибутов	411
Фоновые изображения.....	413
Все цвета радуги (градиенты)	431
И, в завершение, внешние таблицы стилей	438
Подводим итоги	442
Контрольные вопросы	442
Обзор CSS: свойства цвета и фона	444
Глава 14. Блочная модель	445
Блок элемента	445
Свойства <i>width</i>, <i>height</i> и <i>box-sizing</i> (указание размеров блока)	446
Отступы.....	451
Стили границ.....	457
Поля.....	468
Присваивание типов отображения	474
Тени для блока.....	476
Контрольные вопросы	477
Обзор CSS: свойства блока.....	478
Глава 15. Плавающие элементы и позиционирование	481
Нормальный поток	481
Плавающие элементы.....	482
Свободное обтекание текстом CSS-форм.....	494
Основы позиционирования	502
Относительное позиционирование	504
Абсолютное позиционирование	505
Фиксированное позиционирование.....	514
Контрольные вопросы	516
Обзор CSS: свойства плавающих элементов и позиционирования	517
Глава 16. CSS-макетирование с помощью Flexbox и Grid	519
CSS Flexbox: создание гибких контейнеров	520
Модуль CSS Grid Layout.....	549
Контрольные вопросы	584
Обзор CSS: свойства макета.....	588
Глава 17. Адаптивный веб-дизайн	591
Зачем нужен АВД?	591
Рецепт адаптивности	593
Настройка области просмотра.....	594
Гибкие сетки (текущие макеты)	595
Выбор точек прерывания.....	603

Стратегии адаптивной разработки.....	606
Несколько слов о тестировании.....	622
Дополнительные справочные ресурсы по адаптивному веб-дизайну.....	624
Контрольные вопросы	626
Глава 18. Переходы, трансформации и анимация.....	627
CSS Transitions.....	628
Модуль CSS Transforms	638
Анимация по ключевым кадрам.....	650
Несколько завершающих слов	655
Контрольные вопросы	657
Обзор CSS: переходы, трансформации и анимация.....	659
Глава 19. Дополнительные сведения о CSS-методиках	661
Стайлинг форм.....	661
Специальные свойства для форматирования таблиц.....	665
Очистка стилей: сброс CSS и таблица стилей Normalize.css	669
Методика подмены текста изображением.....	671
CSS-спрайты	673
Проверка поддержки свойств CSS	675
Таблицы стилей: послесловие	680
Контрольные вопросы	680
Обзор CSS: свойства таблиц.....	682
Глава 20. Современные инструменты для веб-дизайна	683
Знакомство с командной строкой.....	683
Универсальные CSS-инструменты (процессоры).....	689
Grunt и Gulp: инструменты для сборки кода.....	697
Контроль версий с помощью Git и GitHub.....	701
Заключение.....	710
Контрольные вопросы	711
ЧАСТЬ IV. ПРИМЕНЕНИЕ JAVASCRIPT ДЛЯ УПРАВЛЕНИЯ ПОВЕДЕНИЕМ ВЕБ-СТРАНИЦ.....	713
Глава 21. Введение в JavaScript.....	715
Что есть JavaScript?	715
Добавление кода JavaScript на веб-страницу.....	719
Анатомия сценария	721
Объект браузера.....	737
События	737
Объединяя все вместе	740
Дополнительные сведения о JavaScript	741
Контрольные вопросы	743
Глава 22. Использование JavaScript и DOM	745
Встречайте DOM.....	745
Полифили	755

Библиотеки JavaScript.....	758
Промежуточный финиш.....	763
Контрольные вопросы	764
ЧАСТЬ V. ИЗОБРАЖЕНИЯ ДЛЯ ВСЕМИРНОЙ ПАУТИНЫ	765
Глава 23. Веб-графика: основные понятия.....	767
Источники изображений	767
Знакомство с форматами изображений	772
Размер и разрешение изображения	785
Стратегия работы с изображениями.....	789
Фавиконы.....	793
Подведем итоги...	797
Контрольные вопросы.....	798
Глава 24. Создание веб-изображений	799
Сохранение изображений в веб-форматах	799
Работа с прозрачностью	806
Советы по созданию адаптивных изображений	813
Оптимизация изображения.....	823
Контрольные вопросы	834
Глава 25. SVG-изображения.....	835
Рисование с помощью XML	837
Возможности SVG, связанные с XML	846
Инструменты SVG	853
Советы по созданию SVG	856
Адаптивные изображения в формате SVG.....	860
Дальнейшее изучение SVG	868
Контрольные вопросы	868
И ... мы это сделали!.....	869
ЧАСТЬ VI. ПРИЛОЖЕНИЯ.....	871
Приложение 1	873
Приложение 2	889
Приложение 3	891
Приложение 4	897
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	906

НАПУТСТВИЕ

от Джен Симmons

Если вы когда-либо окажетесь в Кремниевой долине, где расположены главные штаб-квартиры самых известных интернет-компаний мира, то сможете посетить Музей развития компьютерной техники. Прогуляйтесь по залам этого музея, мимо древних мэйнфреймов и витрин со старинными перфокартами, и перед вами в полный рост встанет вся история становления Всемирной паутины. Одними из интереснейших музейных экспонатов того времени являются дискета с копией браузера Mosaic, прилагаемая к описывающей его книге, коробка с браузером Netscape Navigator и нечто под названием «Интернет в коробке» — самое продаваемое тогда интернет-решение № 1 для Windows. Представлены в экспозиции музея и некоторые из самых ранних, наиболее известных в свое время и наиболее важных веб-сайтов, в том числе и сайт Global Network Navigator, созданный в далеком 1993-м году. Так вот, его разработчиком как раз и является автор этой книги — Дженифер Роббинс. Задолго до того, как большинство из нас узнали о существовании глобальной Сети, или, вполне вероятно, задолго до вашего рождения, Джен уже занималась созданием коммерческих сайтов. Она принимала самое активное участие в зарождении и развитии Интернета, вела преподавательскую работу и описывала каждый этап эволюции глобальной Сети.

Перед вами 5-е издание ее книги «Веб-дизайн для начинающих. HTML, CSS, JavaScript и веб-графика», содержащее по сравнению с предшествующими изданиями огромное количество нового и самого современного материала.

Нередко приходится слышать вопрос: «Какие ресурсы лучше всего использовать при освоении интернет-технологий?» Лично я знакомлюсь с новыми технологиями, читая описывающие их книги. Можно, конечно, просматривать сообщения в соответствующих блогах, но если требуется всестороннее и исчерпывающее описание какой-либо темы, то лучше обратиться к книгам. На заре появления интернет-технологий книги, посвященные этим вопросам, предназначались исключительно для новичков. В них описывались HTML, ссылки URL, а также излагались основы работы с браузерами. Когда появились CSS, авторы таких книг предполагали, что их читатель уже имеет опыт работы с HTML, и его нужно научить пользоваться новыми методиками. Затем появились CSS3, и книги стали описывать новые свойства CSS, отсутствовавшие в CSS2. Конечно, книги для начинающих издавались всегда, но они

содержали лишь небольшой набор базовых сведений и не включали описание профессиональных методик, применяемых опытными пользователями. То есть при подготовке каждого следующего поколения книг предполагалось, что читатели владеют умениями и навыками, связанными с предыдущими технологиями. Подобный подход отлично подходит для профессионалов, но у тех, кто только начинает свой путь в мир интернет-технологий, может вызывать определенные затруднения. Можно ли самостоятельно освоить технические новинки, появлявшиеся на протяжении двух десятилетий, отсеивая устаревшие сведения и выбирая ценную информацию? Как построить карьеру, опираясь лишь на базовые понятия и не будучи знакомыми с практическими приемами профессионалов, используемыми ими в повседневной работе?

Вряд ли все это возможно без помощи хорошей книги. Поэтому, когда меня спрашивают о книге, которую я могла бы порекомендовать, у меня есть только один ответ — выберите эту книгу.

Книга, которую вы сейчас держите в руках, не требует каких-либо предварительных знаний. Вам не нужно обладать опытом разработки веб-страниц или знать, где можно найти редактор кода. Изложение материала ведется с самого начала и иллюстрируется примерами. И, тем не менее, в отличие от других книг для начинающих, эта книга поможет вам быстро освоиться в теме. Джен подробно поясняет каждый следующий шаг на этом пути с привлечением весьма продвинутых концепций. Она наполнила эту книгу самыми передовыми знаниями от ведущих экспертов отрасли.

Честно говоря, я не знаю секрета успеха Дженифер. Каким образом можно одновременно преподавать основы и доходчиво излагать достаточно сложные понятия? Чтобы овладеть всем этим на профессиональном уровне, в обычных условиях вам потребовались бы годы. Однако Джен быстро поднимет вас с вашего теперешнего уровня до высот профессионализма. Всем нам, кто состоит в рабочей группе CSS (сообщество людей, работающих над созданием новых стилей CSS), — в том числе и мне — есть чему поучиться, читая эту книгу. И всякий раз, открывая ее, я нахожу для себя в ней что-либо полезное.

Обращайте внимание на сопровождающие текст врезки и примечания. Посещайте сайты и смотрите видеоматериалы, которые Джен вам рекомендует, — это кратчайшие пути к сети профессиональных советов. Не пренебрегайте опытом людей, которых она упоминает. Просматривайте материалы, доступные по ссылкам, которые она приводит. Вполне возможно, что в будущем их авторы станут вашими коллегами. Не бойтесь мечтать о встречах с ними. Как бы там ни было, но это реальные люди, представители нашего маленького мира веб-дизайна, общающиеся с вами в Твиттере, и вы также можете стать его частью. Эта книга поможет вам встать на этот путь.

Джен Симмонс,
дизайнер и представитель группы разработчиков в компании Mozilla,
Член рабочей группы CSS
Апрель 2018

ПРЕДИСЛОВИЕ

Добро пожаловать в пятое издание книги «Веб-дизайн для начинающих. HTML, CSS, JavaScript и веб-графика»

На протяжении десятилетий я занимаюсь написанием книг, посвященных веб-дизайну и интернет-технологиям, и не перестаю удивляться тому, как изменяется веб-ландшафт с появлением каждого следующего издания книги. И новое — пятое издание — не является исключением из этого правила. Мало того, что оно содержит почти на 200 страниц больше, чем предыдущее, но еще и включает достойные упоминания важные обновления и дополнения.

Прежде всего, некоторые технологии и методы, которые на время выхода предыдущего издания книги были новыми или носили экспериментальный характер, уже достаточно хорошо освоены. В наши дни стандарт HTML5 получил повсеместное распространение, а благодаря применению модульного подхода происходит дальнейшее развитие CSS (каскадных таблиц стилей), обеспечивающее последовательное появление и внедрение новых технологий. Кроме того, в настоящее время производители техники нацелены на разработку практически бесконечного ассортимента новых устройств. Адаптивный веб-дизайн, который совсем недавно был новинкой, теперь является стандартом де-факто, применяемым при разработке сайтов. Поэтому в новом издании книги адаптивному веб-дизайну посвящена отдельная глава 17.

Если в предыдущем издании книги шла речь лишь о методике обработки адаптивного изображения, в настоящем издании новые адаптивные элементы изображения уже стандартизированы и хорошо поддерживаются (см. главу 7). Я думаю, что вы по достоинству оцените возможности, представленные в этой главе.

Я была свидетелем многих тектонических сдвигов в веб-дизайне, имевших место за эти годы, и последние из них — это Flexbox и Grid, которые фундаментальным образом изменили подход к веб-дизайну. Подобно тому как CSS избавляют от утомительных хлопот по созданию макетов на основе таблиц и однопиксельных разделятельных GIF-файлов, благодаря Flexbox и Grid наконец-то появилась возможность отказаться от использования устаревших техник подготовки плавающих макетов. Этот новый подход можно сравнить с настоящей революцией, которая позволила найти достойное решение проблем верстки. В это издание книги добавлена новая (и просто огромная!) глава 16, — целиком посвященная созданию полноценного макета страницы с помощью Flexbox и Grid.

РЕСУРС В ИНТЕРНЕТЕ: СОПРОВОЖДАЮЩИЙ КНИГУ ВЕБ-САЙТ

При рассмотрении материала книги не забывайте посещать сопровождающий ее веб-сайт, доступный по адресу: learningwebdesign.com. На этом сайте опубликованы материалы для упражнений, загружаемые статьи, списки ссылок из книги, контактная информация, а также другие сведения.

Несмотря на то что основным условием для успешной реализации процесса веб-разработки является знание HTML, CSS и JavaScript, этого уже недостаточно. По мере развития веб-дизайна появляются новые инструментальные средства, и с некоторыми из них (процессоры CSS, идентификация свойств, командная строка, обработчики задач и Git) вы познакомитесь в главе 20. Конечно, этими инструментами еще нужно научиться пользоваться, но ваши усилия не пропадут даром. В качестве награды вы получите отлаженный и более эффективный рабочий процесс.

Лично для меня наибольшим сюрпризом стало осознание изменений процесса создания и применения изображений в Интернете по сравнению с процессом, описанным в четвертом издании книги. Если не учитывать введение формата PNG, главы книги, посвященные описанию веб-графики, оставались практически неизменными в течение 20 лет (с момента появления на свет первого издания книги). Но все течет, все меняется. Графический формат GIF, который верой и правдой служил веб-разработчикам более 20 лет, в настоящее время практически не используется. На смену ему пришел графический формат PNG, используемый по умолчанию благодаря несомненным преимуществам в производительности и наличию новых инструментов, которые позволяют включать несколько уровней прозрачности в небольшие по размерам 8-битные PNG-файлы. На смену PNG уже спешит формат WebP, позволяющий создавать файлы, имеющие небольшие размеры и обладающие оптимизированными свойствами. Однако наиболее крупным достижением в области развития веб-графики за этот период является появление формата SVG (Scalable Vector Graphics, масштабируемая векторная графика). Созданию и широкому распространению формата SVG способствовала расширенная поддержка этого формата распространенными браузерами. Формат SVG будет подробно рассмотрен в главе 25.

Как и предыдущие четыре издания книги, настоящее издание является руководством, помогающим решать конкретные задачи и проблемы, возникающие перед разработчиками с разным уровнем подготовки. Эта книга будет полезной как для опытных графических дизайнеров, стремящихся расширить свои навыки программистов, так и для всех пользователей, желающих научиться создавать веб-сайты. При подготовке книги я приложила все усилия, для того чтобы на основе опыта преподавания веб-дизайна для начинающих, написать книгу с упражнениями и тестами, позволяющую вам получить практический опыт и проверить имеющийся уровень знаний и навыков.

Независимо от того, читаете ли вы только эту книгу, или же используете ее в качестве дополнения к курсу веб-дизайна, я надеюсь, что она даст вам хороший старт и вы получите удовольствие от самого процесса веб-дизайна.

СТРУКТУРА КНИГИ

Книга состоит из пяти частей, в каждой из которых рассматриваются важные аспекты процесса веб-разработки.

► *Часть I. Начало работы.*

В этой части рассматриваются основы всех тем, которым посвящена книга. Изложение начинается с рассмотрения важной общей информации о среде веб-дизайна, с описания функций, выполняемых веб-дизайнерами, с рассмотрения разных изучаемых технологий и доступных инструментов. Вы сразу жезнакомитесь с HTML, CSS и принципами функционирования Всемирной паутины и веб-страниц, а также с основными понятиями и профессиональными навыками, присущими современным веб-дизайнерам.

► *Часть II. Разметка HTML, применяемая для структурирования веб-страниц.*

В главах, относящихся к этой части, описываются всевозможные аспекты каждого элемента и атрибута, обеспечивающие семантическую структуру контента. Рассматривается разметка текста, ссылок, изображений, таблиц, форм и внедренных медиаэлементов.

► *Часть III. Правила CSS для представления HTML-документов.*

В главах *части III* начинается изучение основ каскадных таблиц стилей, которые применяются для изменения представления текста, создания многоколоночных макетов, а также для добавления анимации и интерактивных возможностей, носящих динамический характер. Здесь же вы найдете введение в адаптивный веб-дизайн, а также описание инструментов и методов, которые являются частью рабочего процесса современного разработчика.

► *Часть IV. Применение JavaScript для управления поведением веб-страниц.*

Автор глав этой части Мэт Маркус начинает здесь с краткого изложения синтаксиса JavaScript, что позволит вам легко отличать переменные от функций. Вы познакомитесь с некоторыми подходами к использованию JavaScript (включая сценарии DOM) и инструментами JavaScript — такими, как полифили и библиотеки, которые позволяют быстро применять JavaScript, даже если вы еще не можете писать собственный код «с нуля».

► *Часть V. Изображения для Всемирной паутины.*

В этой части описываются различные форматы файлов изображений, подходящие для использования во Всемирной паутине, предлагаются стратегии их отбора в рамках адаптивного рабочего процесса и описывается, каким образом оптимизировать изображения, чтобы максимально сократить размер графических файлов. В эту часть также включена глава, посвященная SVG-графике. Это графика обеспечивает серьезные преимущества при ее использовании в процессе адаптивного и интерактивного дизайна.

► *Часть VI. Приложения.*

В этой части приводятся справочные материалы, такие, как ответы на контрольные вопросы, списки глобальных атрибутов HTML и селекторов CSS. Здесь также представлен обзор HTML5 и краткая историческая справка по HTML5.

ТИПОГРАФСКИЕ СОГЛАШЕНИЯ

- ▶ *Шрифт Arial — используется для указания имен файлов и каталогов.*
- ▶ *Курсив — используется для выделения специальных определяемых терминов.*
- ▶ **Полужирный** — используется для указания интернет-адресов (*URL*) и адресов электронной почты.
- ▶ *Моноширинный текст — используется для указания примеров кода и команд клавиатуры.*
- ▶ **Черный моноширинный** — используется для выделений в примерах кода.
- ▶ **Курсивный моноширинный** — используется для выделения заполняемых значений свойств атрибутов и таблиц стилей в примерах кода.
- ▶ → — указывает, что строка кода, которая разделена в тексте на несколько частей, на самом деле является одной строкой.

ЦВЕТНАЯ ВКЛЕЙКА

Большинство иллюстраций книги удобнее рассматривать в цветном формате, однако по понятным причинам в тексте книги они представлены в черно-белом варианте. Тем не менее наиболее важные для понимания материала книги иллюстрации вынесены на цветную вклейку (в тексте ссылки на них помечены префиксом «ЦВ»).

БЛАГОДАРНОСТИ

Очень много умных и хороших людей поддержали меня при подготовке этого издания.

Хочу отдельно поблагодарить моих двух просто удивительных технических рецензентов. Я очень признательна Элике Дж. Этемад (*fantasai*) (Elika J. Etemad), которая, будучи членом рабочей группы W3C CSS, помогла сделать это издание максимально точным и соответствующим стандартам. С Эликой работать нелегко, но результаты этой работы впечатляют. Петтер Десне (Petter Dessne) привнес свой опыт работы в области компьютерных наук, а также ценный опыт как профессора, так и читателя, для которого английский язык является вторым языком. Спасибо ему за хороший настрой, а фотографии его дома в Швеции будут оценены по достоинству!

Также благодарю звезд веб-дизайна первой величины, имена которых приведены списком (в алфавитном порядке), — они внимательно просмотрели отдельные главы и части этого издания книги, относящиеся к областям их деятельности: Амелия Беллами-Ройдс (Amelia Bellamy-Royds) — SVG-графика, Брент Бир (Brent Beer) — инструменты для веб-разработчиков, Крис Коуэр (Chris Coyier) — SVG-графика, Теренс Иден (Terence Eden) — аудио/видео, Брэд Фрост (Brad Frost) — адаптивный веб-дизайн, Лиза Дэнжер Гарднер (Lyza Danger Gardner) — инструменты для

разработчиков), Джейсон Григсби (Jason Grigsby) — веб-изображения, Вэл Хед (Val Head) — анимация, Даниэль Хенгевельд (Daniel Hengeveld) — инструменты для разработчиков, Мэт Маркус (Mat Marquis) — адаптивные изображения, Эрик Мейер (Eric Meyer) — CSS-верстка, Джейсон Палменталь (Jason Pamental) — веб-шрифты, Дэн Роуз (Dan Rose) — изображения, Арсению Сантос (Arsenio Santos) — внедренные медиаэлементы, Джен Симмонс (Jen Simmons) — CSS-верстка, Адам Симпсон (Adam Simpson) — инструменты для разработчика и Джеймс Уильямсон (James Williamson) — структурированные данные.

Также спасибо Мэту Маркусу (Mat Marquis) за его вклад в две подробные главы о JavaScript, которые вряд ли я смогла бы написать на столь высоком уровне, и Джен Симмонс (Jen Simmons) за предваряющее книгу напутствие и за постоянную помощь в процессе обучения веб-дизайну.

Я хочу поблагодарить мою просто потрясающую команду сотрудников из O'Reilly Media: Мег Фоли (Meg Foley) — рецензента издательства, Джека Бльеля (Jeff Bleiel) — редактора по развитию, Кристен Браун (Kristen Brown) — редактора по производству, Рэйчел Монаган (Rachel Monaghan) — редактора, Шэрон Уилки (Sharon Wilkey) — корректора и Люси Хаскинс (Lucie Haskins) — составителя предметных указателей. Особую благодарность хотелось бы выразить InDesign и эксперту по производству книг Рону Билодо (Ron Bilodeau), который превратил мой дизайн в шаблон и набор инструментов, благодаря чему производство книг превратилось в приятное занятие. Также хотелось бы выразить особую признательность Эди Фридману (Edie Freedman) за красивый дизайн обложки и проведенную в дружеских отношениях половину жизни, а также — за отличное руководство.

Наконец, никакие благодарности не были бы полными без глубокой признательности за любовь и поддержку моих дорогих Джека (Jeff) и Арло (Arlo).

ОБ АВТОРЕ

Дженнифер Роббинс (Jennifer Robbins) начала работать в области веб-дизайна в 1993-м году как графический дизайнер первого коммерческого сайта Global Network Navigator. Помимо этой книги, ее перу принадлежит несколько других книг, посвященных веб-дизайну: «Web Design in a Nutshell» и «HTML5 Pocket Reference» и вышедших в издательстве О'Рейли. Дженнифер является основателем и организатором конференции Artifact Conference, посвященной вопросам, связанным с веб-дизайном для мобильных устройств. Она выступала на многих конференциях и преподавала веб-дизайн для начинающих в Университете Джонсона и Уэльса в Провиденсе, штат Род-Айленд. В свободное от работы время Дженнифер занимается рукоделием, увлекается инди-роком, приготовлением пищи, путешествует и воспитывает детей.

КАК С НАМИ СВЯЗАТЬСЯ?

Пожалуйста, направляйте комментарии и вопросы относительно этой книги издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Севастополь, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Имеется и веб-страница этой книги, где отмечаются ошибки, приводятся примеры, а также содержится любая дополнительная информация по теме книги. Для получения доступа к этой странице обратитесь по адресу: bit.ly/learningWebDesign_5e.

Для того чтобы оставить комментарий или задать технические вопросы об этой книге, отправьте электронное письмо по адресу: bookquestions@oreilly.com

Для получения дополнительной информации о наших книгах, курсах, конференциях и новостях посетите наш веб-сайт: www.oreilly.com.

Наша страница в Facebook: facebook.com/oreilly.

Подписывайтесь в Твиттере: twitter.com/oreillymedia.

Смотрите наш канал на YouTube: www.youtube.com/oreillymedia.

ИЗДАТЕЛЬСТВО «БХВ-ПЕТЕРБУРГ»

Оставить свои отзывы и замечания о русском издании книги вы сможете на странице книги на сайте издательства «БХВ-Петербург» по адресу: <https://bhv.ru/>.

ЧАСТЬ I

НАЧАЛО РАБОТЫ

ГЛАВА 1

ЗНАКОМСТВО С ВЕБ-ДИЗАЙНОМ

В этой главе...

- ▶ *Дисциплины, связанные с контентом*
- ▶ *Специальности, связанные с дизайном*
- ▶ *Фронтенд-разработка*
- ▶ *Бэкенд-разработка*
- ▶ *Рекомендованное оборудование*
- ▶ *Программное обеспечение, связанное с Всемирной паутиной*

Всемирная паутина существует уже более 25 лет. За это время она пережила бурный рост на начальном этапе, падение, вызванное экономическим кризисом, подстегнутое инновациями возрождение и постоянное эволюционное развитие. Как бы там ни было, но одно можно заявить вполне определенно: Интернет присутствует во всех сферах нашей жизни. Подключиться к Интернету можно практически с любого современного устройства — такого, как смартфон, планшет, телевизор и многие другие подобные устройства. Ну а тесно связанный с Интернетом веб-дизайн подразумевает использование самых современных технологий.

Благодаря опыту преподавания курсов и семинаров по веб-дизайну, у меня была возможность встретиться с людьми из всех слоев общества, которые заинтересованы в изучении способов разработки веб-страниц. Позвольте мне представить вам всего несколько типичных представителей:

«Я работаю дизайнером печати в течение 17 лет и теперь осознаю, что придется заняться веб-дизайном».

«У меня многолетний опыт работы программистом, но мне все же хочется заняться веб-дизайном, поскольку в моем регионе есть хорошие возможности работы по этой специальности».

«Еще в старших классах школы я занимался разработкой веб-страниц и думаю, что это может стать делом всей моей жизни».

«Я разработал несколько сайтов с помощью тем WordPress, но мне хотелось бы расширить свои навыки и разработать собственные сайты для малого бизнеса».

Какова бы ни была мотивация заняться веб-дизайном, первый вопрос всегда один и тот же: «С чего мне начать?». Изначально может показаться, что существует куча вещей, которые надо изучить, и не всегда понятно, с чего именно начать. Но с чего-то все же начать нужно.

Итак, прежде чем мы перейдем к созданию сайтов, познакомимся с профессиями, связанными с веб-разработкой. Сначала мы узнаем о ролях и обязанностях разработчиков веб-сайтов, чтобы можно было определить, какая роль подходит именно вам. Вы также узнаете об оборудовании и программном обеспечении, которое вы, вероятно, будете использовать, — иными словами, об инструментах веб-разработки.

С ЧЕГО НАЧАТЬ?

Возможно, вы читаете эту книгу в рамках полного курса по веб-дизайну и веб-разработке. А, может быть, купили ее, чтобы заняться самообразованием. Либо просто открыли ее из любопытства. В любом случае, эта книга — хорошее пособие, чтобы начать изучать веб-дизайн.

Существует множество уровней участия в веб-дизайне: от создания небольшой личной страницы до разработки полнофункционального корпоративного сайта. Вы можете быть как универсальным веб-разработчиком, так и узкоспециализированным, владеющим каким-либо одним навыком. То есть имеется великое множество путей, по которым вы можете пойти.

Если вы хотите заниматься веб-дизайном или намерены профессионально разрабатывать веб-сайты, вам придется поднять уровень своих навыков в области веб-дизайна до соответствующей отметки. Ваши работодатели могут и не спросить вас о наличии диплома веб-дизайнера, но они непременно захотят посмотреть на рабочие образцы сайтов, которые продемонстрируют им ваши знания и опыт. Эти сайты могут появиться в результате выполнения классных заданий, личных проектов или же явиться плодом усилий по разработке сайта для малого бизнеса или организации. Важно, чтобы они имели профессиональный вид, чтобы код HTML был чистым и хорошо написанным, а таблицы стилей и сценарии незаметно выполняли свою работу.

Если ваше участие в веб-разработке имеет скромные масштабы — допустим, что вы хотите создать и опубликовать один-два сайта, то вы вполне можете начать с использования шаблона, доступного на сайте, предоставляющем услуги по веб-дизайну (см. врезку «Я просто хочу иметь свой сайт»). Большинство этих шаблонов позволяют настроить базовый код, поэтому знания, полученные из этой книги, помогут вам настроить такой шаблон по своему вкусу.

Я ПРОСТО ХОЧУ ИМЕТЬ СВОЙ САЙТ

Чтобы начать разработку сайтов, вовсе не обязательно становиться веб-дизайнером или разработчиком. Существует множество служб хостинга веб-сайтов, которые предоставляют пользователям шаблоны и интерфейсы с поддержкой перетаскивания, облегчающие создание сайта без каких-либо знаний кода. Эти шаблоны можно использовать для чего угодно: от полнофункциональных решений для электронной коммерции до небольших персональных сайтов (хотя некоторые шаблоны лучше подходят для одних целей, а некоторые — для других).

Вот несколько наиболее популярных сервисов по созданию сайтов, которые существовали на момент подготовки этой книги:

- *WordPress* — www.wordpress.com;
- *Squarespace* — squarespace.com;
- *Wix* — wix.com;
- *SiteBuilder* — sitebuilder.com;
- *Weebly* — weebly.com.

Существует множество других подобных сервисов, поэтому поищите в Интернете то, что подходит именно вам.

РАСПРЕДЕЛЕНИЕ РОЛЕЙ ПРИ РАЗРАБОТКЕ САЙТОВ

Когда я смотрю на сайт, то вижу множество решений и областей знаний, которые были задействованы при его создании. Сайты — это нечто больше, чем просто код и картинки. Создание сайта зачастую начинается с составления бизнес-плана или выполнения другой определенной миссии. Прежде чем запустить сайт, нужно создать и организовать контент, выполнить соответствующее исследование, тщательно проанализировать дизайн и написать код. Причем, все эти действия должны быть согласованы с процессами, происходящими на сервере, и тогда создание сайта увенчается успехом.

Большие, известные сайты создаются командами, состоящими из десятков, сотен или даже тысяч участников. Впрочем, существуют и такие сайты, которые создаются и поддерживаются немногочисленной командой. Также вполне возможно создавать отличный сайт собственноручно. Это разнообразие и придает всю прелест Всемирной паутине.

В этом разделе вы познакомитесь с различными дисциплинами, которые способствуют созданию сайта, включая роли, связанные с созданием контента, разработкой дизайна и кода. В конечном итоге вы можете специализироваться только в одной области знаний, работая в составе группы специалистов. Если же вы разрабатываете сайты самостоятельно, вам придется быть мастером на все руки. Это то же самое, что и выполнение обязанностей по дому, подразумевающее работу неполный рабочий день шеф-поваром, уборщицей, бухгалтером, дипломатом, садовником и строителем. Разрабатывая сайт самостоятельно, вы овладеете многими специальностями, связанными с Интернетом, но вам будет просто казаться, что вы всего лишь создаете сайт.

Контент — всему голова

Каждый, кто использует термин «веб-дизайнер», должен знать, что все, что мы делаем, направлено на процесс передачи контента, сообщений или функциональных возможностей нашим пользователям. Кроме того, качественный веб-дизайн поможет в создании более эффективного пользовательского интерфейса: от надписей на кнопках до сообщений об ошибках.

Конечно же, кто-то должен создать весь этот контент и поддерживать его, и все это потребует немалых ресурсов. Хорошие писатели и редакторы контента являются важной частью команды. Кроме того, я хочу обратить ваше внимание на двух специалистов по контенту, принимающих участие в современной веб-разработке: информационный архитектор (ИА) и контент-стратег.

Информационная архитектура

Информационный архитектор (также называемый *информационным дизайнером*) организует контент логически, обеспечивая простоту его поиска. Эти специалисты несут ответственность за функциональность поиска, структуру сайтов и организацию контента и данных на сервере. Информационная архитектура неизбежно переплетается с проектированием взаимодействия и проектированием пользовательского интерфейса (эти понятия будут определены чуть позже), а также с управлением контентом. Если вам нравится все организовывать или же вы склонны к систематизации, вам будет по душе заняться информационной архитектурой.

Дополнительные сведения по этой теме, относящиеся к Всемирной паутине, можно найти в книге издательства О’Реили: «Information Architecture: For the Web and Beyond», написанной Луи Розенфельдом (Louis Rosenfeld) и Питером Морвиллем (Peter Morville).¹

Стратегия контента

Если контент некорректен, сайт вряд ли будет полноценным. Контент-стратег следит за тем, чтобы каждый фрагмент текста на сайте: от длинного пояснительного текста до надписей на кнопках, поддерживал идентичность бренда и маркетинговые цели организации. Стратегия контента может также распространяться на моделирование данных и управление контентом в большом и постоянном масштабе — например, при планировании повторного использования контента и обновлении расписаний. В обязанности контент-стратегов также может входить представление организации в социальных сетях.

Дополнительные сведения по этой теме можно найти в книге издательства New Riders «Content Strategy for the Web», 2-е издание, написанной Кристиной Халворсон (Kristina Halvorson) и Мелиссоу Рич (Melissa Rich).

¹ Следует иметь в виду, что многие книги, упоминаемые автором по ходу изложения материала, имеются в продаже в русских переводах — стоит только выполнить по их названиям соответствующий поиск (Прим. ред.).

Все грани дизайна

Ах, дизайн! Это слово звучит весьма просто и кажется всем знакомым, но даже дизайн простого сайта требует ряда отдельных специализаций. Далее приведено несколько описаний обязанностей, связанных с разработкой сайта, но имейте в виду, что они часто пересекаются, и человек, называющий себя «дизайнером», часто отвечает за выполнение нескольких (если не всех) этих обязанностей.

Пользовательский опыт, взаимодействие и дизайн пользовательского интерфейса

Часто, когда мы думаем о проектировании (дизайне), мы подразумеваем приданье внешнего вида чему-либо. При разработке сайта первой задачей является определение того, как этот сайт будет работать. Прежде чем выбрать цвета и шрифты, важно определить цели сайта, то, как он будет использоваться и каким образом посетители сайта будут перемещаться по его разделам. Эти задачи относятся к таким видам профессиональной деятельности, как *проектирование пользовательского опыта* (UX, User Experience), *проектирование взаимодействия* (IxD, Interaction Design) и *проектирование интерфейса пользователя* (UI, User Interface). Эти виды деятельности имеют много общего, и нередко бывает так, что их все выполняет один человек или команда.

Дизайнер пользовательского опыта должен придерживаться целостного взгляда на процесс проектирования, обеспечивая удобство работы с сайтом в целом. Дизайн пользовательского опыта основан на глубоком понимании интересов пользователей и их потребностей, почерпнутом на основе наблюдений и интервью. Согласно Дональду Норману (Donald Norman), которому принадлежит этот термин, дизайн пользовательского опыта подразумевает «все аспекты взаимодействия пользователя с продуктом: как он воспринимается, усваивается и используется». В случае веб-сайта или приложения этот термин подразумевает визуальный дизайн, пользовательский интерфейс, качество и содержание сообщения и даже общую производительность сайта. Залог успешности пользовательского опыта — соответствие бренду и бизнес-целям организации.

Цель дизайнера взаимодействия — сделать сайт максимально простым, эффективным и приятным в использовании. С проектированием взаимодействия тесно связано проектирование пользовательского интерфейса, которое, как правило, более узко ориентировано на функциональную организацию страницы, а также на конкретные инструменты: кнопки, ссылки, меню и т. п., которые пользователи используют для навигации по контенту или выполнения задач.

Дизайнеры пользовательского опыта, интерфейса пользователя и взаимодействия пользователя могут создавать следующие документы:

- ▶ **отчеты по изучению мнений пользователей и отчеты по тестированию разрабатываемого сайта** — понимание потребностей, желаний и ограничений пользователей имеет решающее значение для успеха дизайна сайта или веб-приложения.

Подход к проектированию, ориентированный на потребности пользователя, называется *проектированием, ориентированным на пользователя* (UCD, *User-Centered Design*). Именно этот подход занимает центральное место в современном веб-дизайне. Дизайн сайта часто начинается с *изучения мнений пользователей*, включая интервью с ними и наблюдения за ними. Это позволяет лучше понять, каким образом сайт может обеспечивать решение проблем пользователей или каким образом он будет вообще использоваться. Для проектировщиков характерно проведение раунда пользовательского тестирования сайта на каждом этапе процесса его проектирования, чтобы обеспечить удобство использования созданных сайтов. Ежели пользователю будет трудно понять, где найти контент или как перейти к следующему шагу в процессе, тогда он вернется к «чертежной доске» и продолжит свою работу вручную;

- ▶ **каркасные схемы** — каркасная схема демонстрирует структуру веб-страницы с помощью контуров, иллюстрирующих каждый тип контента и виджета (рис. 1.1). Цель каркасной схемы — указать, как разделена область экрана и где размещены функциональные элементы и контент: элементы навигации, поля поиска, элементы формы и т. п. Цвета, шрифты и другие элементы визуальной идентификации на каркасной схеме намеренно опускаются, чтобы не отвлекать внимания от структуры страницы. Эти схемы обычно снабжены инструкциями о том, как все должно работать, чтобы команда разработчиков имела представление о создаваемом сайте;

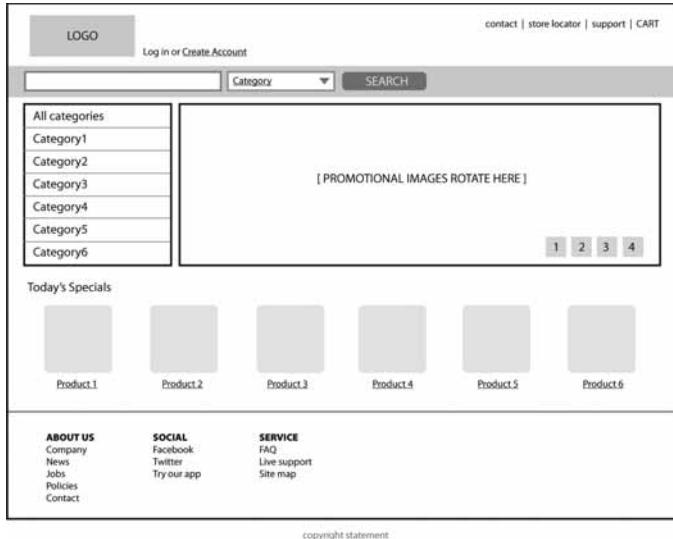


Рис. 1.1. Каркасная схема

- ▶ **схемы сайта** — схема сайта иллюстрирует структуру сайта в целом и то, как отдельные страницы связаны друг с другом. На рис. 1.2 показана очень простая схема сайта. Некоторые схемы сайтов настолько велики, что занимают несколько страниц;

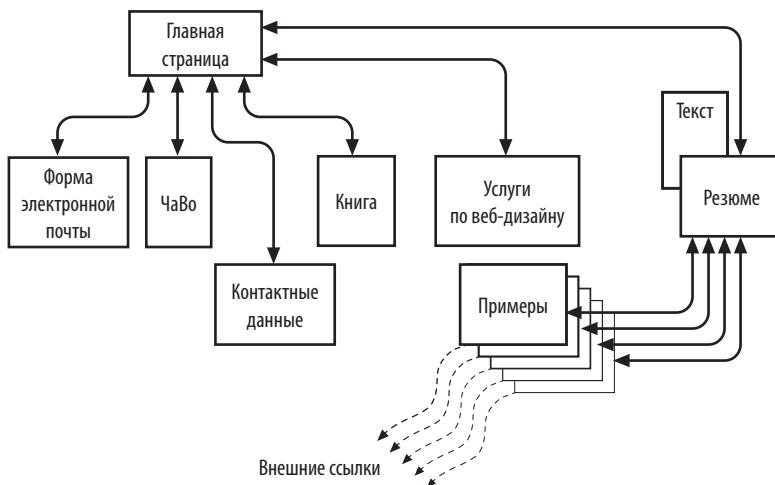


Рис. 1.2. Простая схема сайта

- **раскадровки и пользовательские блок-схемы** — раскадровка отслеживает путь типичного пользователя (персоны — на языке дизайнёров опыта взаимодействия) через сайт или приложение. Обычно она включает сценарий и «сцены», состоящие из представлений экрана или пользователя, взаимодействующего с экраном. Раскадровка предназначена для демонстрации шагов, необходимых для выполнения задач, описания возможных вариантов, а также введения некоторых стандартных типов страниц. На рис. 1.3 показана простая раскадровка.

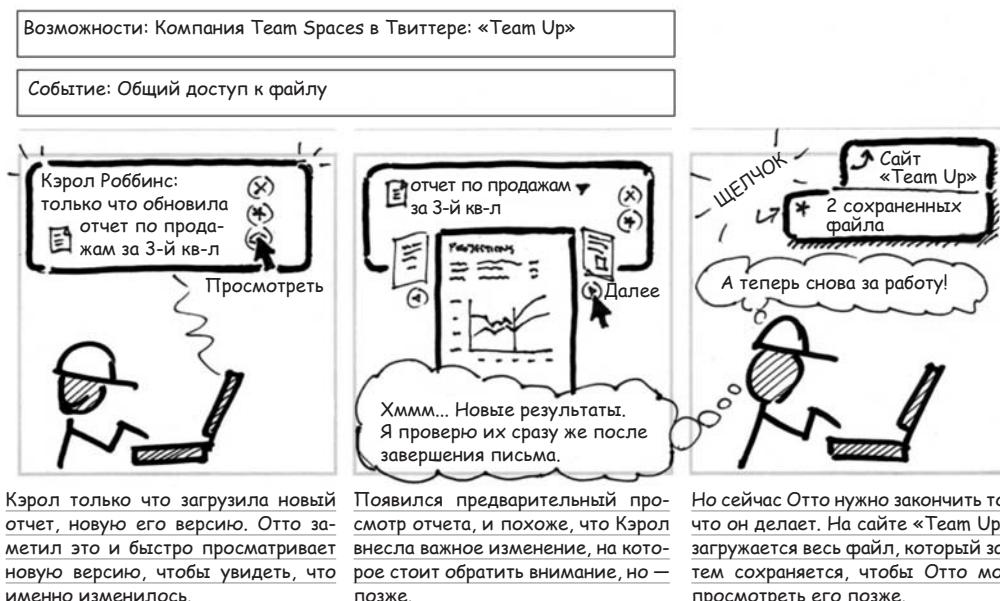


Рис. 1.3. Типичная раскадровка (нарисована Брэндоном Шоером и любезно представлена компанией Adaptive Path)

Пользовательская блок-схема — это еще один метод, показывающий, как связаны части сайта или приложения, но здесь упор делается на технических деталях, а не на повествовании. Например, «когда пользователь делает это, он выполняет такую функцию на сервере». Обычно дизайнеры создают схему действий пользователя, осуществляющих реализацию шагов процесса, — таких, как регистрация участника или выполнение платежей в Интернете.

Выпущено немало книг по проектированию пользовательского опыта, взаимодействию пользователя и дизайну интерфейса пользователя. В следующем списке приведено несколько классических книг подобного рода, которые помогут вам начать освоение этой темы:

- «The Elements of User Experience: User-Centered Design for the Web and Beyond» — написана Джесси Джеймсом Гарреттом (Jesse James Garrett) и издана в издательстве New Riders;
- «Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability» — написана Стивом Кругом (Steve Krug) и увидела свет в издательстве New Riders;
- «The Design of Everyday Things» — написана Доном Норманом (Don Norman) и увидела свет в издательстве Basic Books;
- «About Face: The Essentials of Interaction Design», 4-е издание — написана Аланом Купером (Alan Cooper), Робертом Райманном (Robert Reimann), Дэвидом Кронином (David Cronin) и Кристофером Ноесселом (Christopher Noessel) и выпущена издательством Wiley;
- «Designing Interfaces», 2-е издание — написана Дженнифер Тидвелл (Jenifer Tidwell) и выпущена издательством O'Reilly;
- «100 Things Every Designer Needs to Know about People» — написана Сьюзен Вайнштенк (Susan Weinschenk) и вышла в свет в издательстве New Riders;
- «Designing User Experience: A Guide to HCI, UX and Interaction Design» — написана Дэвидом Беньоном (David Benyon) и увидела свет в издательстве Pearson.

Визуальный (графический) дизайн

Поскольку Интернет является визуальной средой, веб-страницы требуют внимания к их визуальному представлению. Отношение к сайту определяют первые впечатления пользователя. Графический дизайнер формирует «внешний вид» сайта: логотипы, графику, тип, цвета, макет и т. п. таким образом, чтобы сайт производил хорошее первое впечатление и согласовывался с брендом и идеей организации, которую он представляет.

Существует множество способов и документов, которые можно использовать для представления визуального дизайна клиентам и заинтересованным сторонам. Наиболее традиционными являются эскизы или макеты (созданные в Photoshop или с помощью аналогичной программы), иллюстрирующие внешний вид сайта, — например макеты домашней страницы, показанные на рис. 1.4.

Теперь, когда сайты отображаются на экранах всех размеров, многие дизайнеры предпочитают рассматривать визуальную идентификацию, применяющуюся при разработке сайтов (цвета, шрифты, стиль изображения и т. п.), не привязывая ее к конкретной компоновке — такой, как типичные рабочие столы, показанные на рис. 1.4. Идея состоит в том, чтобы договориться о визуальном языке, используемом при создании сайта, до начала производства самого этого сайта.

Одним из вариантов отделения стиля от размера экрана является использование плиток стиля (см. также врезку «Коллажи элементов»). Эта техника предложена Самантой Уоррен (Samantha Warren). Плитки стилей включают примеры цветовых схем, элементов брендинга, трактовки пользовательского интерфейса, трактовки текста и настроения (рис. 1.5). Как только детали определены, они могут быть внедрены в рабочие прототипы и конечный сайт.

Чтобы узнать больше об этой технике, посетите отличный сайт Саманты: styleit.es, где можно загрузить соответствующий шаблон.

Графические дизайнеры могут быть также привлечены к созданию графических ресурсов для сайта. Им нужно будет знать, как оптимизировать изображения для максимальной быстрой доставки и как удовлетворить требования к различным размерам экрана. Разработчики тоже обычно занимаются оптимизацией изображений, но я думаю, что этим навыком должен обладать каждый графический дизайнер (оптимизация изображений будет рассмотрена в главе 24).

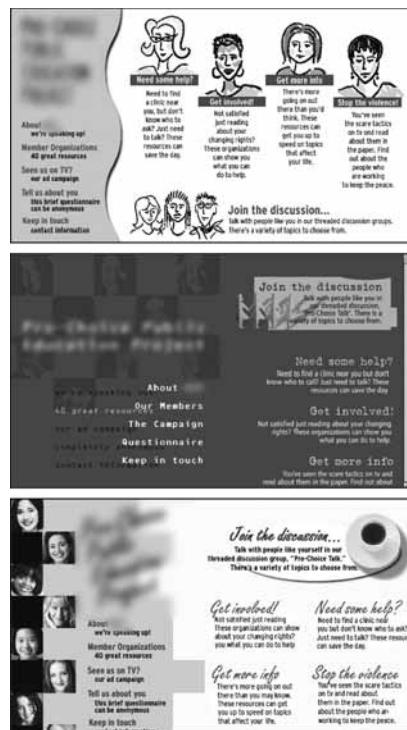


Рис. 1.4. Эскизы (макеты), иллюстрирующие внешний вид простого сайта

КОЛЛАЖИ ЭЛЕМЕНТОВ

Дизайнер Дэн Молл (Dan Mall) использует аналогичный подход, который он называет «коллажи элементов». Коллаж элементов — это набор элементов дизайна, которые придают сайту уникальный внешний вид, но, как и стилевые фрагменты, не привязаны к определенному макету экрана. Прочтайте его статью, доступную на сайте: v3.danielmall.com/articles/rif-element-collages/.



Рис. 1.5. Техника, предусматривающая использование плиток стиля (предложена Самантой Уоррен)

НУЖНО ЛИ ДИЗАЙНЕРАМ УЧИТЬСЯ ПРОГРАММИРОВАТЬ?

Если коротко, то да. Базовое знакомство с HTML и CSS теперь является обязательным требованием для любого, кто присоединяется к команде веб-дизайнеров. Вы как веб-дизайнер не можете нести ответственность за создание окончательного рабочего кода для сайта, но поскольку HTML и CSS являются «родными» языками для вашей среды, вам нужно знать, как с ними быть. Некоторые дизайнеры также изучают и JavaScript, а некоторые отказываются это делать, отдавая разработчику прерогативу заниматься программированием.

Код становится все более важным для рабочего процесса дизайнера визуального контента. Если когда-то в Photoshop было все, что нужно для макетирования веб-страницы и ее передачи в производство, то сейчас макеты с фиксированным размером не соответствуют описанию страницы, которую придется адаптировать в соответствии с размерами экрана. По этой причине дизайнеры создают свои собственные рабочие прототипы в качестве результирующих документов, которые позволяют получить представление о том, как будет выглядеть и вести себя дизайн, попавший в руки пользователей.

На дизайнеров также могут быть возложены обязанности и по созданию руководства по стилю, которое документирует выбор стилей, — таких, как шрифты, цвета и другие стили, позволяющие обеспечить постоянную согласованность сайта.

Список примеров, статей, книг и подкастов о руководствах по веб-стилю можно найти на странице **Website Style Guide Resources**, доступной по адресу: styleguides.io.

Разработка кода

Значительная часть процесса разработки веб-сайта включает создание и корректировку документов, таблиц стилей, сценариев и изображений, которые образуют сайт. В фирмах, предоставляющих услуги веб-дизайна, команда, которая занимается созданием файлов, составляющих сайт (или шаблонов для страниц, которые собираются динамически), обычно называется отделом *разработки* или *производства*.

Разработка сводится к двум широким категориям: *фронтенд-разработка* и *бэкенд-разработка*. Еще раз — эти задачи могут быть переданы специалистам, но их может выполнять один человек или команда.

Фронтенд-разработка

Термин «фронтенд» относится к любому аспекту процесса проектирования веб-сайта, который напрямую связан с браузером. Этот процесс включает в себя использование HTML, CSS и JavaScript, для чего — если вы действительно хотите заниматься веб-разработкой — вам понадобятся серьезные знания. Давайте рассмотрим каждый из этих аспектов по отдельности.

ФРОНТЕНД-РАЗРАБОТКА

Фронтенд-разработчики обычно используют следующие веб-технологии:

- язык гипертекстовой разметки (HTML, HyperText Markup Language);
- каскадные таблицы стилей (CSS, Cascading Style Sheets);
- написание сценариев JavaScript и DOM, включая фреймворки, основанные на AJAX и JavaScript.

Авторинг — разметка с помощью HTML

Авторинг (Authoring) — это процесс подготовки контента для доставки через Интернет, или, более конкретно, разметка контента с помощью HTML-тегов, которые описывают его содержимое и функции.

Язык HTML (HyperText Markup Language, гипертекстовый язык разметки) — это язык авторинга, используемый для создания веб-страниц. Текущая версия (и версия, использованная в этой книге) — HTML 5.2 (в *приложении 4* описана история HTML и приведены характеристики, которые делают HTML5 уникальным).

Надо подчеркнуть, что язык HTML — это именно *язык разметки*, а не программирования, то есть он применяется для идентификации и описания различных компонентов документа — таких, как заголовки, абзацы и списки. Разметка определяет базовую структуру документа — эта структура напоминает детализированную машиночитаемую схему. Чтобы создавать код HTML, не нужны программистские навыки — достаточно располагать терпением и здравым смыслом.

Лучший способ научиться работать с HTML — создать несколько веб-страниц вручную, как мы будем делать в упражнениях второй части этой книги.

Стайлинг — создание представлений с помощью CSS

В то время как HTML используется для описания контента (содержимого) веб-страницы, *каскадные таблицы стилей* (CSS, Cascading Style Sheets) описывают, как этот контент должен выглядеть (см. врезку «Каскадные таблицы стилей»). Внешний вид веб-страницы называется *ее представлением*. С помощью CSS можно управлять шрифтами, цветами, фоновыми изображениями, межстрочным интервалом, макетом страницы и т. п. На свою веб-страницу можно даже добавить специальные эффекты и базовую анимацию.

Спецификация CSS также предоставляет методы для управления представлением документов в контекстах, отличных от браузера, — таких, как печать или чтение вслух программой чтения с экрана, но в нашей книге эти методы подробно не рассматриваются.

КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ

Когда в этой книге используется термин «таблицы стилей», он всегда относится к *каскадным таблицам стилей* — стандартному языку таблиц стилей для World Wide Web. Таблицы стилей (включая значение термина «каскадирование») будут подробно рассмотрены в части III книги.

Хотя можно публиковать веб-страницы лишь с помощью HTML, лучше использовать таблицы стилей, чтобы не зависеть от стилей браузера, заданных по умолчанию. Если вы занимаетесь профессиональным проектированием сайтов, будь вы дизайнер или разработчик, знание CSS обязательно.

Написание сценариев JavaScript и DOM

Язык JavaScript — это язык сценариев, который добавляет к веб-страницам интерактивность и поведение, включая следующие элементы (и это лишь некоторые из них):

- проверка записей формы на корректность;
- обмен стилями между элементом и всем сайтом;
- автоматическая загрузка информационных каналов с большим количеством контента;
- обеспечение запоминания браузером информации о пользователях;
- создание виджетов интерфейса — таких, как встроенные видеоплееры или специальные формы ввода.

Вы также в связи с JavaScript могли слышать термин «сценарий DOM». Аббревиатура DOM расшифровывается как *объектная модель документа* (Object Document Model) и относится к стандартизированному списку элементов веб-страниц, доступ к которым можно получить с помощью JavaScript (или другого языка сценариев).

От фронтенд-разработчиков также может потребоваться знание фреймворков JavaScript (таких, как React, Bootstrap, Angular и других), которые автоматизируют большую часть производственного процесса. Скорее всего, им также понадобится AJAX (Asynchronous JavaScript And XML, Асинхронный JavaScript и XML) — метод, используемый для загрузки контента в фоновом режиме, который позволяет гладко обновлять страницу без перезагрузки (подобно тому как автоматически на веб-странице обновляются каналы).

КОНСОРЦИУМ WORLD WIDE WEB

Консорциум World Wide Web (сокращенно W3C) — это организация, которая занимается разработкой таких веб-технологий, как HTML, CSS и JavaScript. Группа была основана в 1994-м году в Массачусетском технологическом институте (MIT) Тимом Бернерс-Ли (Tim Berners-Lee), создателем Интернета.

Вначале группа W3C занималась главным образом разработкой протокола HTTP и развитием HTML. Теперь W3C закладывает основу для будущего Интернета, создавая десятки технологий и протоколов, которые должны работать вместе, формируя цельную инфраструктуру.

Чтобы получить исчерпывающий ответ на любой вопрос о веб-технологиях, посетите сайт W3C, доступный по следующему адресу: www.w3.org.

Для получения дополнительной информации о W3C и о том, что он делает, вы также можете посетить и следующую полезную страницу: www.w3.org/Consortium/.

Написание сценариев для Всемирной паутины определенно требует некоторого традиционного компьютерного мастерства. Хотя многие веб-разработчики имеют ученые степени в области компьютерных наук, нередки случаи, когда они являются самоучками. Несколько разработчиков, которых я лично знаю, начинали с копирования и адаптации существующих сценариев, а затем постепенно — с каждым новым проектом — приобретали навыки программирования. Тем не менее, если у вас нет опыта работы с языками программирования, начальная кривая обучения может оказаться весьма крутой.

Если вы собираетесь заниматься веб-разработкой профессионально, вам придется освоить JavaScript. Дизайнеры извлекут пользу из понимания того, что может делать JavaScript, но, возможно, им не придется учить этот язык, если они работают вместе с командой разработчиков (подробнее о JavaScript будет рассказано в главе 21). Если же вы сами хотите профессионально освоить JavaScript, обратитесь к книге издательства O'Reilly «Learning JavaScript», написанной Итаном Брауном (Ethan Brown).

Бэкенд-разработка

Бэкенд-разработчики сосредоточены на создании серверных программ, включая приложения и базы данных, которые выполняются на сервере. Они могут также заниматься установкой и настройкой серверного программного обеспечения (подробнее серверы будут рассмотрены в главе 2). *Им, безусловно, потребуется знать, по крайней мере, один, а возможно, и несколько серверных языков программирования — таких, как PHP, Ruby, .NET (или ASP.NET), Python или JSP, чтобы создавать приложения, предоставляющие функциональность, требуемую для сайта: обработку форм, управление контентом с помощью соответствующих систем (CMS, Content Management Systems), выполнение покупок в Интернете — и это лишь некоторые из них.*

Кроме того, бэкенд-разработчики должны быть знакомы с настройкой и поддержкой баз данных, в которых хранятся все данные для сайта: контент, который добавляется в шаблоны, учетные записи пользователей, товарные запасы и т. п. В число распространенных языков управления базами данных входят MySQL, Oracle и SQL Server.

Бэкенд-разработка выходит за рамки этой книги, но важно понимать, какие задачи решаются на уровне сервера. Следует также знать, что такую функциональность, как корзины для покупок, списки рассылки и многое другое, можно получить в виде готовых решений от вашей хостинговой компании без необходимости программировать ее с нуля.

БЭКЕНД-РАЗРАБОТКА

Бэкенд-разработчики обычно используют следующие технологии:

- серверное программное обеспечение (Apache, Microsoft IIS);
- языки веб-приложений (PHP, Ruby, Python, JSP, ASP.NET);
- программное обеспечение баз данных (MySQL, Oracle, SQL Server).

УНИВЕРСАЛЬНЫЕ РАЗРАБОТЧИКИ И «ЕДИНОРОГИ»

В процессе поиска работы в области веб-разработки вы часто будете видеть сообщения, авторы которых пишут, что ищут универсальных разработчиков. Это означает, что претендент на эту работу должен свободно владеть как фронтенд- (HTML, CSS, JavaScript), так и бэкенд-технологиями (серверные приложения, базы данных).

Существует редкая категория веб-дизайнеров, которые могут справиться со всеми упомянутыми ранее задачами: от контент-стратегии до проектирования пользовательского опыта и фронтенд-разработки на сервере. Эти люди известны в своей среде как «единороги». Мне доводилось встречаться с некоторыми из них!

Другие роли

Надеюсь, для вас не станет сюрпризом, что существует множество и других ролей, специалисты которых способствуют созданию и поддержке сайта. Вот несколько таких специализаций, в целом выходящих за рамки термина «веб-дизайн»:

- ▶ **менеджер по продукту** — менеджер по продукту веб-сайта или приложения направляет его дизайн и разработку таким образом, чтобы он соответствовал бизнес-целям. Этот член команды должен иметь полное представление о целевом рынке, а также о процессах, связанных с созданием самого сайта. Менеджеры по продуктам разрабатывают общую стратегию сайта с точки зрения маркетинга, включая то, в каком виде и когда он будет выпущен;
- ▶ **менеджер проекта** — менеджер проекта координирует дизайнеров, разработчиков и всех, кто работает над сайтом. Они управляют такими категориями, как сроки, подходы к разработке, результаты и т. п. Менеджер проекта работает с менеджером продукта и другими специалистами, чтобы обеспечить выполнение проекта вовремя и в рамках бюджета;
- ▶ **специалист по поисковой оптимизации** — мало создать веб-сайт или приложение, надо еще заявить о них всему миру. Поэтому очень важно, чтобы сайт был легко найден поисковыми системами. *Поисковая оптимизация* (SEO, *Search Engine Optimization*) — это дисциплина, сфокусированная на настройке структуры и кода сайта таким образом, чтобы повысить вероятность получения им высокого рейтинга в результатах поиска. Специалист по поисковой оптимизации может быть в штате, или же компания может принять решение нанять специалиста по поисковой оптимизации из сторонней компании. Поисковая оптимизация иногда воспринимается как черная магия, хотя существует много способов улучшить результаты поиска, которые всем доступны. На самом деле улучшить реализацию поисковой оптимизации можно при наличии хорошего контента с оптимизированной HTML-разметкой;
- ▶ **создатели мультимедиаконтента** — одна из замечательных особенностей Всемирной паутины — возможность добавлять на сайт мультимедийные элементы,

в том числе звук, видео, анимацию и даже интерактивные игры. Создание мультимедийных элементов обычно лучше предоставить художникам и техническим специалистам в этих областях, хотя они могут быть частью веб-команды, если видео, анимация или интерактивность являются основой миссии сайта.

На этом мы завершаем знакомство с виртуальной командой специалистов, вовлеченных в создание веб-сайта. Чем больше сайт, тем больше вероятность того, что у каждого члена команды будет узкая специализация и должности, — такие, как «Руководитель по поиску ошибок при проектировании пользовательского опыта». Скорее всего, все в команде будут обладать целым спектром навыков, и границы между дисциплинами окажутся размыты. Например, я занимаюсь проектированием взаимодействия, пользовательского интерфейса, графическим дизайном, HTML и CSS, но я не пишу сценарии JavaScript, не работаю на сервере и не занимаюсь организацией контента. В этой книге я стремлюсь дать вам основы фронтенд-технологий, которые подготовят вас к выполнению ряда рассмотренных нами ролей.

ГУМАНИТАРНЫЕ НАВЫКИ, В КОТОРЫХ НУЖДАЕТСЯ КАЖДЫЙ ВЕБ-РАЗРАБОТЧИК

До сих пор мы фокусировались на нескольких чисто технических навыках, полезных при создании сайтов. Я хотела бы упомянуть еще несколько навыков, часто упускаемых из виду, которые также важны для достижения успеха.

- *Прекрасные навыки общения* — при выполнении своей работы вам придется общаться лично, по телефону, по электронной почте и с помощью инструментов обмена текстовыми сообщениями с клиентами, членами команды и начальством. Будьте ясны, предупредительны и честны при выражении своих мыслей. Хорошее общение требует не только четкого самовыражения, но и хорошего слушателя. Убедитесь, что вы понимаете обсуждаемые вопросы, и не бойтесь просить разъяснений, если вы что-то не понимаете.
- *Гибкость* — будьте способны к быстрой переориентации, поскольку современные веб-технологии изменяются очень быстро, и это может привести к тому, что вы не сможете выполнять свою повседневную работу. Например, вы можете однажды прийти на работу и обнаружить, что клиент полностью изменил свои приоритеты. Вы можете обнаружить, что заказчик отказался от запланированного проекта. Вас могут попросить освоить новые навыки и изменить роль, выполняемую в команде. Возможность к быстрой адаптации является ключом к выживанию.
- *Критическое мышление и здравый смысл* — решение проблем занимает центральное место во всех дисциплинах, связанных с веб-дизайном, поэтому вам нужно уметь использовать навыки критического мышления, чтобы находить решения и всегда использовать здравый смысл.
- *Хорошее отношение* — создание сайтов означает быть частью команды, даже если вы работаете дома фрилансером. Помните, что отношение к вашей работе заразительно, поэтому старайтесь быть позитивным и дружелюбным членом команды.

ПОДГОТОВКА К ВЕБ-РАЗРАБОТКЕ

Неудивительно, что профессиональные веб-дизайнеры для выполнения своей работы требуют весьма много оборудования, аппаратного и программного обеспечения. Мне часто задают один вопрос: «Что мне нужно купить?». Я не могу сказать конкретно, что покупать именно вам, но приведу здесь обзор типичных инструментов, которые потребуются для выполнения веб-дизайна.

Оборудование

Для создания комфортной среды веб-разработки рекомендуется следующее оборудование:

- ▶ *надежный современный компьютер* — для выполнения веб-дизайна вполне пригодны компьютеры Macintosh, Windows или Linux, и вы можете использовать все, что у вас есть и с чем вам удобно. Творческие (т. н. креативные) отделы в профессиональных компаниях, занимающихся веб-разработкой, как правило, работают на Macintosh. Для бэкенд-разработки можно использовать Linux и Windows. И хотя очень приятно иметь супербыстрый компьютер, файлы, образующие веб-страницы, очень малы и не слишком обременительны для компьютеров. Если вы не занимаетесь редактированием звука и видео, не беспокойтесь, если ваш компьютер не самый современный и не самый лучший;
- ▶ *большой монитор* — хотя это и не обязательно, большой монитор облегчает жизнь. Чем больше размер у вашего монитора, тем больше окон и панелей управления можно на нем открывать одновременно. Вы также сможете видеть пространство за пределами создаваемой веб-страницы, что поможет вам принимать дизайнерские решения. Однако, если вы используете большой монитор, имейте в виду, что пользователи будут рассматривать разработанные вами веб-страницы на сравнительно небольших мониторах и экранах устройств;
- ▶ *второй компьютер для тестирования* — многие дизайнеры и разработчики предпочитают иметь тестовый компьютер, работающий на платформе, которая отличается от платформы компьютера, используемого для разработки (то есть, если вы проектируете на Macintosh, тестируйте в среде Windows). Поскольку браузеры выглядят на компьютерах Macintosh иначе, чем на компьютерах с Windows, очень важно протестировать свои страницы в максимально возможном количестве сред — особенно в текущей операционной системе Windows. Если вы не профессиональный веб-дизайнер и работаете дома, то можете проверить свои страницы на компьютере друга (пользователи компьютеров Macintosh могут обратиться ко врезке «Запуск Windows на компьютере Macintosh»);
- ▶ *мобильные устройства для тестирования* — Всемирная паутина стала мобильной! Это означает, что абсолютно необходимо протестировать внешний вид и производительность вашего сайта в браузерах на смартфонах и планшетах (тестирование устройств будет рассмотрено в главе 17);
- ▶ *сканер и/или камера* — если вы планируете создавать свои собственные изображения и текстуры, вам понадобятся и некоторые инструменты, предназначенные для их разработки.

ЗАПУСК WINDOWS НА КОМПЬЮТЕРЕ MACINTOSH

Если у вас есть компьютер Macintosh с процессором Intel, работающим под управлением macOS (Leopard или более поздней версии), вам не нужен отдельный компьютер для тестирования в среде Windows. Теперь можно запускать Windows прямо на вашем компьютере Macintosh, используя бесплатное приложение Boot Camp, которое позволяет переключаться на Windows при перезагрузке.

Существуют также несколько других продуктов, позволяющих создать на macOS виртуальную машину (Virtual Machine), дающую возможность переключаться между Macintosh и Windows:

- VMFusion (www.vmware.com/fusion) — это коммерческий продукт с бесплатным пробным периодом;
- Parallels Desktop for Mac (www.parallels.com) — это также коммерческий продукт с бесплатным пробным периодом;
- Oracle VirtualBox (virtualbox.org) — это бесплатная программа, которая позволяет запускать несколько гостевых операционных систем, включая Windows и несколько разновидностей UNIX.

Для работы с виртуальными машинами требуется купить копию Microsoft Windows, но это, безусловно, лучше, чем покупка целой машины.

Программное обеспечение для веб-дизайна

Существует множество программ, предназначенных для создания веб-страниц. На заре эры веб-дизайна мы вполне обходились приложениями, предназначенными для набора текста и печати. Сегодня же появились замечательные инструменты, созданные специально для веб-дизайна, которые делают этот процесс более эффективным. В этой книге приведен весьма скромный список таких программ, поскольку: а) их существует очень много; б) у каждого пользователя есть своя любимая программа; и в) новые инструменты появляются так быстро, что, безусловно, уже есть новые, более продвинутые, варианты программ, чем те, что имеются у вас. Причем, на момент подготовки книги многих таких программ просто не существовало.

Тем не менее вот общий обзор типов программ, применяемых для веб-дизайна, а также несколько конкретных упоминаний о самых популярных инструментах в каждом классе.

НИЧЕГО ПОКУПАТЬ НЕ ПРИДЕТСЯ...

Чтобы выполнить упражнения, описанные в этой книге, вам понадобится только текстовый редактор, который поставляется с вашей операционной системой, и бесплатное программное обеспечение для создания изображений. Вам не придется покупать какие-либо программы.

Инструменты кодирования

Хотя можно обойтись и простыми текстовыми редакторами, которые поставляются с вашим компьютером, специальный редактор кода значительно упрощает написание кода HTML, CSS и JavaScript. Редакторы кода воспринимают синтаксис

создаваемого кода, поэтому они могут выполнять такие операции, как цветовое кодирование, обнаружение ошибок и автоматическое завершение простых задач — типа закрытия тегов HTML. Некоторые редакторы дают возможность предварительного просмотра страницы, чтобы вы могли увидеть результаты выполнения своего кода во время работы.

На рис. ЦВ-1.6 показано, как выглядит HTML-документ в окне редактора Sublime Text (напомним, что иллюстрации с префиксом ЦВ также продублированы на цветной вклейке). Вот лишь некоторые из наиболее известных редакторов кода, применяемые для создания веб-страниц, которые стоит изучить:

- Sublime Text — sublimetext.com;
- Atom — atom.io (бесплатно загружается из GitHub);
- Brackets — brackets.io (бесплатно загружается из Adobe);
- CodeKit — codekitapp.com (только для Macintosh);
- Adobe Dreamweaver — www.adobe.com/products/dreamweaver.html;
- Coda — panic.com/coda/;
- Microsoft Visual Studio — visualstudio.com.

Инструменты, предназначенные для создания интерфейса пользователя и макетирования

В настоящее время появилось множество инструментов разработки интерфейса веб-сайтов и других приложений. Поскольку они изначально создавались для разработки интерфейса, они, похоже, предвосхищают все потребности веб-дизайнера.

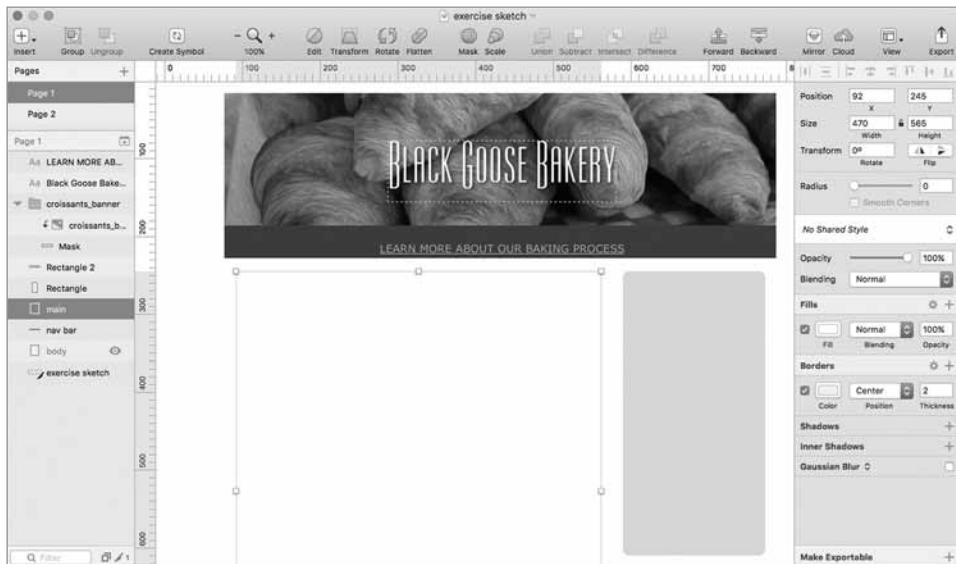


Рис. 1.7. Приложение Sketch (только для Macintosh) — пример инструмента разработки интерфейса

Инструменты, предназначенные для разработки интерфейса, позволяют легко проектировать несколько макетов (например, макетов для отображения на экранах с разными размерами), а также экспортировать изображения и код для использования в производстве. Некоторые инструменты допускают базовую интерактивность — такую, как щелчки и прокрутку, поэтому ваши макеты могут быть размещены в Интернете и использоваться для базового тестирования интерфейса.

Пригодное для использования только на Macintosh приложение Sketch (sketchapp.com), окно которого показано на рис. 1.7, было чрезвычайно популярным на момент подготовки книги. Наравне с этим приложением доступны также следующие:

- Affinity Designer — affinity.serif.com/en-us/designer/;
- Adobe XD — www.adobe.com/products/xd.html;
- Figma — figma.com;
- UXPin — uxpin.com.

Инструменты, предназначенные для создания веб-графики

Безусловно, все изображения, необходимые для сайта, можно создать с помощью одного из упомянутых ранее инструментов разработки интерфейса. Существуют также программы, ориентированные исключительно на создание изображений, которые могут экспортить файлы в веб-форматах. Профессиональным дизайнерам стоит воспользоваться набором инструментов Adobe Creative Cloud (adobe.com), который включает Photoshop (рис. 1.8), Illustrator и другие высококлассные инструменты создания изображений.

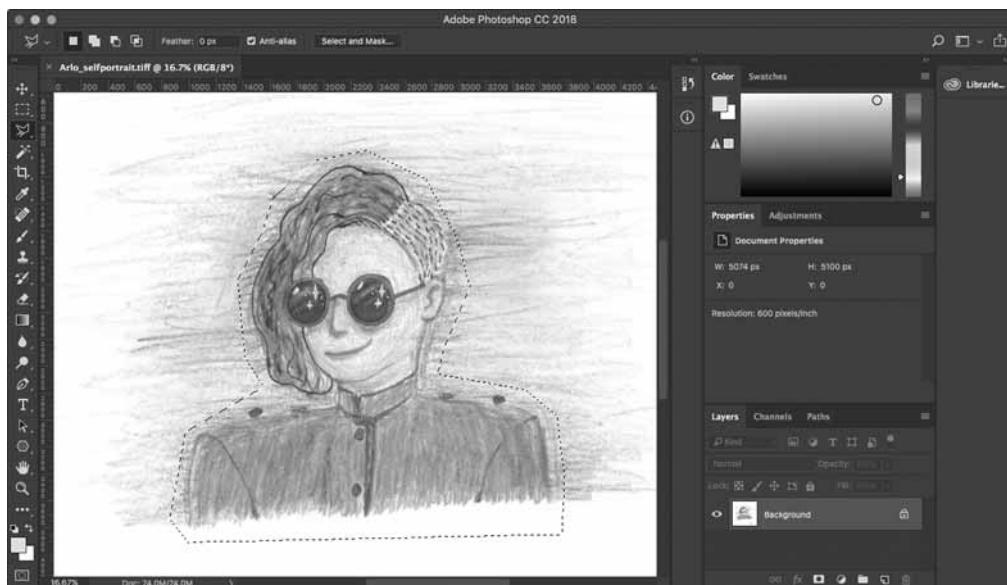


Рис. 1.8. Adobe Photoshop — профессиональный стандарт приложения для редактирования изображений

Если ежемесячная абонентская плата за доступ к продуктам Adobe для вас слишком велика, вы можете попробовать недорогие альтернативы, которые предоставляют многие аналогичные функции. Количество графических инструментов просто огромно, поэтому здесь представлены лишь некоторые из них:

- GIMP — gimp.org (бесплатное, с открытым исходным кодом);
- Corel PaintShop Pro — paintshoppro.com (предназначен для редактирования фотографий, только для Windows);
- Corel DRAW — coreldraw.com (для работы с векторной графикой, только для Windows);
- Pixelmator — pixelmator.com (только для Macintosh).

Следующие графические редакторы работают прямо в вашем браузере без необходимости загружать программу, хотя вам придется платить за поддержку учетной записи:

- SumoPaint — sumopaint.com;
- Pixlr — pixlr.com.

Разнообразие браузеров

Одна из самых больших проблем для веб-дизайнеров заключается в том, что создаваемые сайты могут выглядеть и вести себя по-разному при отображении в разных браузерах. По этой причине очень важно, чтобы создаваемые проекты тестировались на ранних этапах и зачастую в самых разных браузерах. Далее приведен список браузеров, используемых дизайнерами и разработчиками для тестирования:

- Chrome — google.com/chrome;
- Firefox — www.mozilla.org;
- MS Edge — www.microsoft.com/en-us/windows/microsoft-edge (только для Windows);
- Internet Explorer 9–11 — www.microsoft.com (доступен по поиску Internet Explorer, только для Windows);
- Safari — support.apple.com/downloads/#safari (только для Macintosh);
- Opera — opera.com.

Также нужно будет протестировать различные браузеры для смартфонов, включая iOS Safari, Android-браузеры и сторонние мобильные браузеры (тестирование браузеров для мобильных устройств будет рассмотрено в главе 17).

Инструменты для управления файлами и передачи их между устройствами

В процессе веб-дизайна и разработки сайтов происходит перемещение множества файлов — особенно часто с рабочего компьютера на сервер, где размещается сайт. Для перемещения файлов через Интернет используется программа FTP (File Transfer Protocol, Протокол передачи файлов). Помимо этого, многие хостинговые службы предлагают свои собственные инструменты FTP, предназначенные для загрузки

ваших файлов на свои серверы. Многие из упомянутых ранее редакторов кода также включают встроенные функции FTP. Или же можно использовать отдельную программу FTP — например, одну из следующих:

- Filezilla — filezilla-project.org (бесплатная, для всех платформ);
- Cyberduck — cyberduck.io (существуют версии для Macintosh и Windows);
- WinSCP — winscp.net/eng/index.php (бесплатная, только для Windows);
- Transmit — panic.com/transmit (только для Macintosh).

Может также оказаться полезным иметь *терминальное приложение* (*инструмент командной строки*), которое позволяет вводить команды UNIX для установки прав доступа к файлам, перемещения или копирования файлов и каталогов, или управления программным обеспечением сервера.

Инструменты командной строки, которые используются в процессе веб-дизайна и разработки, более подробно обсуждаются в главе 20:

- Terminal — устанавливается вместе с macOS (показан на рис 1.9);
- Cygwin — cygwin.com (эмулятор Linux для Windows, который включает инструмент командной строки).

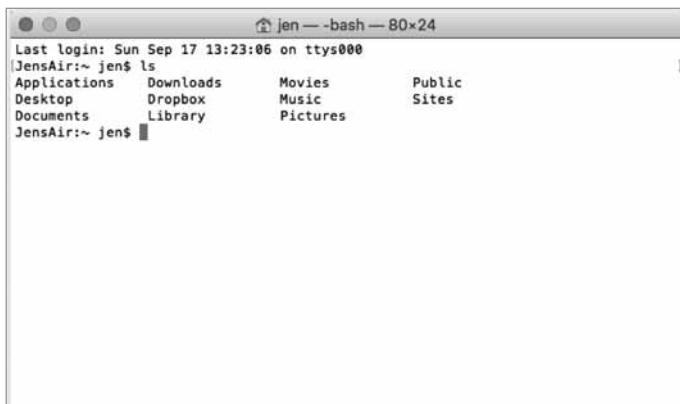


Рис. 1.9. Инструмент командной строки Terminal для macOS

ПОДВЕДЕМ ИТОГИ

Я надеюсь, что эта глава дала вам обзор многих ролей и обязанностей, которые входят в сферу «веб-дизайна». Я также надеюсь, что после прочтения этой главы вы осознаете, что вовсе необязательно изучать кучу всего, чтобы стать веб-дизайнером. И даже если вы хотите узнать абсолютно все о веб-дизайне, совсем не обязательно изучать все сразу. Так что расслабьтесь и не волнуйтесь. Другая хорошая новость заключается в том, что, хотя существует много профессиональных инструментов, можно создать базовый веб-сайт и запустить его, не тратя много денег, используя свободно доступные или недорогие инструменты и имеющийся у вас компьютер.

Как вы вскоре увидите, начать создавать веб-страницы легко. Более того, вы сможете начать создание простых страниц сразу после прочтения этой книги. Затем вы сможете продолжить самообразование и найти свою конкретную нишу в веб-дизайне. А пока попробуйте ответить на вопросы из *упражнения 1.1*.

УПРАЖНЕНИЕ 1.1. ПОДВЕДЕНИЕ ИТОГОВ

Теперь, когда вы сделали первый шаг в изучении веб-дизайна, самое время подвести первые итоги. Используя материал этой главы в качестве общего руководства, попробуйте найти ответы на следующие вопросы.

- Каковы ваши цели при занятии веб-дизайном? Стать профессиональным веб-дизайнером? Создавать лишь персональные веб-сайты?
- Какие аспекты веб-дизайна интересуют вас в первую очередь?
- Какие ваши текущие навыки будут полезными при создании веб-страниц?
- Какие навыки вам нужно приобрести или освежить?
- Какое аппаратное и программное обеспечение, применяемое для веб-дизайна, у вас имеется?
- Какие инструменты вам нужно купить? Какие инструменты вы хотели бы купить в конце концов?

КОНТРОЛЬНЫЕ ВОПРОСЫ

Каждая глава в этой книге заканчивается несколькими вопросами, на которые вы можете ответить, чтобы убедиться в том, что хорошо усвоили материал. Ответы на эти вопросы приведены в *приложении 1*.

1. Сопоставьте этих веб-профессионалов с конечным продуктом, за который они могут отвечать:
 - Графический дизайнер _____ документы HTML и CSS;
 - Отдел производства _____ сценарии PHP;
 - Дизайнер пользовательского опыта _____ продукты «смотри и ощущай»;
 - Бэкенд-программист _____ раскладовки.
2. Чем занимается группа W3C?
3. Сопоставьте веб-технологию с соответствующей ей задачей:
 - HTML _____ проверка корректности ввода в поле формы;
 - CSS _____ создание пользовательского серверного веб-приложения;
 - JavaScript _____ идентификация текста как заголовка второго уровня;
 - Ruby _____ выделение всех заголовков второго уровня синим цветом.
4. Чем отличается фронтенд-разработка от бэкенд-разработки?
5. Для чего нужен инструмент FTP и как его получить?

ГЛАВА 2

КАК ВСЕ ЭТО РАБОТАЕТ?

В этой главе...

- ▶ Связь между Интернетом и Всемирной паутиной
- ▶ Роль сервера
- ▶ Роль браузера
- ▶ Структура ссылок URL
- ▶ Структура веб-страницы

Я начала заниматься веб-дизайном в начале 1993-го года, в те времена, когда Всемирная сеть была на этапе зарождения. Это было четверть века назад, но я до сих пор отчетливо помню, как впервые увидела веб-страницу. Тогда мне трудно было сказать, откуда берется информация и как все это работает.

В этой главе будут рассмотрены базовые понятия и вводятся некоторые основные термины. Мы начнем с общей картины и перейдем к деталям.

СВЯЗЬ МЕЖДУ ИНТЕРНЕТОМ И ВСЕМИРНОЙ ПАУТИНОЙ

В этом разделе мы рассмотрим термины, которые все чаще взаимозаменяются.

- ▶ *Интернет* — это международная сеть подключенных компьютеров. Ни одна компания не владеет Интернетом — он представляет собой результат совместных усилий, регулируемых системой стандартов и правил. Целью соединения между компьютерами, конечно же, является обмен информацией. Существует множество способов передачи информации между компьютерами, включая электронную почту (POP3, IMAP, SMTP), передачу файлов (FTP), безопасную оболочку (SSH) и многие другие специализированные режимы, на которых построен Интернет. Эти стандартизованные методы передачи данных или документов по сети называются *протоколами*.
- ▶ Всемирная паутина (World Wide Web, то есть те самые **www** в адресах сайтов) — это лишь один из способов обмена информацией через Интернет. Он уникален тем, что позволяет связывать документы друг с другом посредством гипертекстовых ссылок, формируя таким образом огромную «сеть» связанной информации. Во Всемирной паутине используется протокол под

ВСЕМИРНАЯ ПАУТИНА

Всемирная паутина является подмножеством Интернета. Это всего лишь один из многих способов передачи информации через сетевые компьютеры.

названием HTTP (HyperText Transfer Protocol, Протокол передачи гипертекста). Эта аббревиатура должна быть вам знакома, поскольку это первые четыре буквы входят практически во все адреса веб-сайтов, о чём мы поговорим в следующем разделе.

КРАТКАЯ ИСТОРИЯ ВСЕМИРНОЙ ПАУТИНЫ

Рождение Всемирной паутины произошло в 1989-м году, в стенах Европейского центра ядерных исследований (ЦЕРН) в Женеве, Швейцария. Компьютерный специалист Тим Бернерс-Ли (Tim Berners-Lee) предложил сформировать систему управлением информацией, в которой путем обработки «гипертекста» через сеть поддерживается связь между документами. Тим и его партнер, Роберт Каю (Robert Cailliau) сформировали прототип и представили его для просмотра широкой общественности. На протяжении ряда лет веб-страницы включали только текст. Трудно представить, что еще в 1992-м году во всем мире существовало лишь 50 веб-серверов.

Взрывообразный прорыв в популярности Всемирной паутины произошел в 1992-м году, когда появился первый графический браузер (NCSA Mosaic) и Всемирная паутина «перекочевала» из области научных исследований в разряд масс-медиа. Дальнейшее совершенствование веб-технологий связано с консорциумом World Wide Web (W3C).

Если вас заинтересует история становления Всемирной паутины обратитесь к историческим архивам W3C W3C, доступным по следующему адресу: www.w3.org/History.html.

Забавный факт: если вы обратитесь к историческим архивам, то увидите запись о первом семинаре «WWW Wizards Workshop», датированную июлем 1993-го года. И, хотя я не присутствовала на этом семинаре, мне выпала честь создать дизайн памятной футболки!

ПОРЯДОК ОБРАБОТКИ ИНФОРМАЦИИ

Давайте поговорим о больших компьютерах, которые составляют Интернет. Поскольку именно они по запросу «обрабатывают» документы, их называют *серверами*. Точнее, сервер — это программное обеспечение, позволяющее компьютеру взаимодействовать с другими компьютерами (а не только компьютер сам по себе). Тем не менее именно термин «сервер» чаще всего используется для такого компьютера. Роль серверного программного обеспечения заключается в ожидании запроса данных, находящийся нужной информации и ее скорейшей отсылке инициатору запроса.

ОТКРЫТЫЙ ИСХОДНЫЙ КОД

Программное обеспечение с открытым исходным кодом является результатом совместных усилий разработчиков, направленных на то, что сделать исходный код этого приложения доступным для использования и изменения со стороны других программистов. Программы с открытым исходным кодом, как правило, свободны для использования.

Обычные компьютеры могут быть самыми разными: от универсальных UNIX-машин до скромного персонального компьютера. Именно серверное программное обеспечение делает возможным использование компьютера в роли сервера. Компьютер становится частью сети, если на нем установлено специальное программное обеспечение веб-сервера, позволяющее обрабатывать транзакции протокола передачи гипертекста. Веб-серверы также называются *HTTP-серверами*.

Существует много разновидностей серверного программного обеспечения, но наиболее популярны среди них: Apache (программное обеспечение с *открытым исходным кодом*) и Microsoft Internet Information Services (IIS). Программное обеспечение Apache распространяется на бесплатной основе для компьютеров на базе UNIX и устанавливается на компьютерах Macintosh с операционной системой macOS. Существует также версия и для Windows. Программное обеспечение Microsoft IIS входит в семейство серверных решений, предлагаемых Microsoft.

Каждому компьютеру и устройству (маршрутизатору, смартфону, автомобилю и т. д.) при подключению к Интернету присваивается уникальный числовой *IP-адрес* (аббревиатура «IP» означает «Internet Protocol», протокол Интернета). Например, компьютер, поддерживающий сайт oreilly.com, имеет IP-адрес 199.27.145.64. На первый взгляд, подобный набор цифр непонятен, но в нам сейчас следует знать, что он существует и хорошо выполняет свои функции. *Система доменных имен* (DNS, Domain Name System) позволяет ссылаться на этот сервер и с помощью *доменного имени* «oreilly.com». Цифровой IP-адрес используется компьютерным программным обеспечением, а доменное имя более привычно для пользователей. Сопоставление же текстовых имен доменов с соответствующими числовыми IP-адресами отнесено к задачам выделенного DNS-сервера. Если представлять IP-адрес как телефонный номер, то DNS-сервер выполняет роль телефонной книги.

Можно настроить веб-сервер так, чтобы несколько имен доменов относились к одному IP-адресу, — тогда несколько сайтов будут совместно пользоваться одним сервером.

ПАРА СЛОВ О БРАУЗЕРАХ

Узнав о том, что «сервер выполняет обработку данных», перейдем к рассмотрению другой части уравнения. Программное обеспечение, которое осуществляет запросы, называется *клиентом*. В роли клиентов для доступа к документам, находящимся в Интернете, выступают настольные браузеры, мобильные браузеры и другие вспомогательные средства (например, программы чтения с экрана). Сервер возвращает документы для дальнейшего просмотра браузеру, который в технических кругах также называется *пользовательским агентом*.

Запросы и ответы обрабатываются в соответствии с упомянутым ранее протоколом HTTP. Хотя мы называем все это «документами», HTTP успешно используется для передачи изображений, фильмов, аудиофайлов, данных, сценариев и всех других веб-ресурсов, которые обычно входят в состав веб-сайтов и приложений.

Обычно браузер представлен на мониторе компьютера окном, в котором отображается веб-страница. Ранее известные как графические или настольные браузеры, эти приложения длительное время обеспечивали монопольный доступ к Интернету. На момент подготовки книги наиболее популярными являются следующие браузеры для настольных компьютеров: Edge и Internet Explorer — для Windows, Chrome, Firefox и Safari, а вот Opera и Vivaldi пока отстают по популярности.

В наши дни более половины интернет-трафика приходится на мобильные браузеры, установленные на смартфонах и планшетах, — таких, как Safari для iOS, Android и Chrome для устройств Android, Opera Mini, а также на многие другие стандартные и устанавливаемые мобильные браузеры (см. полный список по адресу: ru.wikipedia.org/wiki/Мобильный_браузер). Серфинг в Интернете с помощью мобильных браузеров становится все более популярным.

Также следует упомянуть об альтернативных возможностях для получения доступа ко Всемирной паутине. Пользователи с ослабленным зрением могут услышать содержание веб-страницы, которая читается с экрана специальной программой (или же просто увеличить шрифт). Пользователи с ограниченной подвижностью могут прибегать к вспомогательным устройствам — таким, как джойстики, или же использовать голосовые команды для доступа к ссылкам и вводу контента. Создаваемые сайты должны быть доступными и служить как можно более широкому кругу пользователей независимо от их возможностей.

Доступ ко Всемирной паутине можно получить с помощью смарт-телевизоров и игровых систем, причем пользователи могут управлять доступом к своим любимым веб-страницам с помощью пультов телевизоров или контроллеров Xbox. Экспансия Всемирной паутины продолжается, и недалек тот час, когда доступ к ней можно будет получить самым экзотическим образом!

ИНTRANЕТ И ЭКСТРАНЕТ

Когда идет речь о веб-сайте, обычно предполагается, что доступ к нему можно получить только через Интернет. Тем не менее многие компании обращаются к преимуществам этого способа обмена информацией и создают веб-сайты для работы лишь в пределах собственной корпоративной сети. Подобные специальные веб-сети именуются *интранетом*. Они созданы и функционируют как обычные веб-сайты, но при помощи специальных устройств обеспечения безопасности (называемых *брандмауэрами*) остаются невидимыми и недоступными для внешних пользователей. Возможности интранета широко используются при обменах различного рода конфиденциальной информацией (в том числе и личными данными сотрудников организации) или для доступа к информации, находящейся во внутренних базах данных.

Экстранет во многом схож с интранетом, различие заключается в том, что с его помощью предоставляются ограниченные возможности для части пользователей, находящихся вне пределов организации. Например, компания-производитель может предоставлять клиентам пароли, с помощью которых можно проверять состояние своих заказов в базе данных заказов компании. Пароли и определяют, какая именно часть информации компании доступна клиентам.

СТОРОНА СЕРВЕРА И СТОРОНА КЛИЕНТА

Весьма часто при обсуждении веб-дизайна можно услышать про «клиентские» или «серверные» приложения. Подобные термины применяются, чтобы подчеркнуть, какая машина выполняет обработку: клиентские приложения выполняются на компьютере пользователя (также называемом *фронтиеном*), а серверные приложения и функции используют вычислительную мощность серверного компьютера (*бэкенд*).

МЕХАНИЗМЫ ВИЗУАЛИЗАЦИИ БРАУЗЕРА

Программа, отвечающая за преобразование HTML и CSS в изображение, выводимое на экран, называется *механизмом визуализации* (а также *движком браузера* или *инструментом разметки*). Браузеры, устанавливаемые на настольные компьютеры и мобильные устройства, включают механизмы визуализации и прочий код, используемый для собственных пользовательских интерфейсов и функций. Хотя в этой книге много говорится о том, какие именно браузеры поддерживают те или иные функции, технически речь идет именно о механизме визуализации браузера. Различные браузеры часто используют один и тот же механизм визуализации — например, движок Blink работает в браузерах Chrome, Opera и различных браузерах для платформы Android. В табл. 2.1 приведены механизмы визуализации, используемые наиболее популярными в наше время веб-браузерами.

Для получения дополнительной информации выполните поиск в Википедии — например, по запросу: сравнение механизмов визуализации браузеров или: сравнение браузеров.

Таблица 2.1. Современные браузеры и задействованные в них механизмы визуализации

Браузер	Механизм визуализации
Chrome 28+	Blink (ветвь WebKit)
Firefox (все версии)	Gecko (помимо Firefox для iOS, который использует WebKit)
Safari и Safari iOS (все версии)	WebKit
Internet Explorer 4–11	Trident
MS Edge (все версии)	EdgeHTML (ветвь Trident)
Opera 15+	Blink (ветвь WebKit)

Как показывает практика, веб-страницы — в зависимости от используемого браузера — могут выглядеть и вести себя различным образом. Это связано с различиями в поддержке веб-технологий, с возможностями устройств, а также с собственными предпочтениями пользователей. Кстати, именно этот аспект представляет наибольшую сложность в процессе проектирования и разработки веб-сайтов для разных сред.

АДРЕСА ВЕБ-СТРАНИЦ: URL

Каждая веб-страница либо ресурс в Интернете имеют собственный специальный адрес — URL (от слов Uniform Resource Locator, унифицированный локатор ресурса). Практически дня невозможно прожить, не наткнувшись на очередной URL, которые в качестве рекламы фигурируют на транспорте, вносятся на визитные карточки или демонстрируются по телевидению. Адреса веб-страниц уже повсеместно интегрировались в современный стиль общения.

Некоторые URL короткие и запоминающиеся. Другие же могут выглядеть как хаотичные строки символов, разделенных точками и слэшами, но каждая отдельная их часть имеет свое предназначение. Давайте рассмотрим их повнимательнее.

Структура интернет-адреса

Полный интернет-адрес (URL) обычно состоит из трех компонентов: названия протокола, имени сайта и абсолютного пути к документу или ресурсу (рис. 2.1).

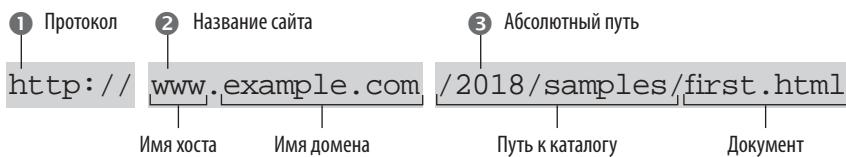


Рис. 2.1. Части интернет-адреса (URL)

- ➊ **http://** — сначала интернет-адрес определяет протокол, который будет использоваться для выполнения конкретной транзакции. Префикс **HTTP** сообщает серверу об использовании протокола передачи гипертекста (HyperText Transfer Protocol) или о переходе в «веб-режим». Вы также можете увидеть интернет-адрес, начинающийся с **https://**. Значение этих букв объясняется во врезке «HTTPS — защищенный протокол Интернета».
- ➋ **www.example.com** — следующая часть интернет-адреса идентифицирует сайт по его доменному имени. В этом примере доменное имя: **example.com**. Фрагмент **www.** в начале имени сайта представляет конкретное имя хоста в этом домене. Имя хоста **www** выбрано по соглашению, но не является правилом. Иногда имя хоста опускается. На одном домене может находиться несколько веб-сайтов (тогда их называют *поддоменами*). Например, может указываться: **development.example.com**, **clients.example.com** и т. д.
- ➌ **/2018/samples/first.html** — таким образом задается абсолютный путь к запрашиваемому HTML-документу **first.html**, проходящий через каталоги на сервере. Разделенные слэшем (косой чертой) слова представляют имена каталогов, начиная от корневого каталога хоста (что указано с помощью начального слэша: `/`). Поскольку Интернет изначально состоял из компьютеров, функционирующих под управлением операционной системы UNIX, способ их работы по-прежнему следует правилам и соглашениям UNIX, поэтому для разделения имен каталогов и применяется символ слэша: `/`.

В итоге, если вникнуть в смысл представленной на рис. 2.1 записи, мы узнаем, что протокол **HTTP** используется для подключения в Интернете к веб-серверу под названием **www.example.com**, а затем запрашивается документ **first.html**, расположенный в каталоге **samples**, который, в свою очередь, находится в каталоге **2018**.

Упрощенные интернет-адреса

Конечно же, далеко не каждый указатель ресурса URL такой длинный. Например, чтобы попасть на сайт O'Reilly, можно просто ввести: `oreilly.com` вместо: `http://www.oreilly.com/index.html`. И все это прекрасно работает.

URL ИЛИ URI?

В наши дни консорциум W3C и сообщество разработчиков отходят от термина URL (Uniform Resource Locator, унифицированный локатор ресурса) и переходят к более универсальному и технически точному термину URI (Uniform Resource Identifier, унифицированный идентификатор ресурса). Но среди обычных пользователей все равно чаще используется термин URL.

В чем же различия между URL и URI? URL — это один из типов URI, который идентифицирует ресурс по его расположению в сети: от L, что означает «Location» (Расположение). Еще один тип URI называется URN. Этот тип идентифицирует ресурс по имени или пространству имен: от N, что означает «Name» (Имя).

Поскольку аббревиатура URL более распространена, именно она используется в книге. Просто надо учесть, что URL служит лишь подмножеством URI, но эти термины часто взаимозаменямы.

Если вы хотите поближе познакомиться с термином URI, обратитесь к следующей статье из Википедии: ru.wikipedia.org/wiki/URI.

Пропуск названия протокола

Поскольку практически все веб-страницы используют протокол передачи гипертекста HTTP (HyperText Transfer Protocol), использование названия `http://` зачастую само собой подразумевается, и это название опускается. Обычно пропуск этого названия имеет место при рекламе названий сайтов — как в печатном виде, так и на телевидении, что облегчает запоминание собственно интернет-адреса (URL).

Кроме того, браузеры запрограммированы на автоматическое добавление к вводимому имени сайта префикса `http://`, чтобы пользователю не приходилось лишний раз нажимать клавиши. Вам может показаться, что вы пропускаете название протокола, но оно все равно будет передано на сервер.

При рассмотрении методики использования интернет-адресов для создания гиперссылок в документах HTML (см. главу 6) вы заметите, что при формировании ссылки на веб-страницу на другом сервере включать название протокола в интернет-адрес необходимо.

HTTPS — ЗАЩИЩЕННЫЙ ПРОТОКОЛ ИНТЕРНЕТА

Если обратить внимание на адресную строку браузера при совершении покупок в Интернете или на банковском сайте, можно заметить, что всюду используется протокол HTTPS. Протокол HTTPS, где «S» означает «Security» (Безопасность), является такой модификацией HTTP, которая шифрует информацию формы при передаче данных между клиентом пользователя и сервером. Если панель, отображаемая на веб-странице, содержит текстовые поля (например, поле поиска или логин), то следует использовать HTTPS.

На момент подготовки книги уже около 60% информации (и этот объем возрастает!) передается с помощью HTTPS, и на это есть веские причины. С одной стороны, применение HTTPS позволяет гарантировать безопасность передаваемых данных

► пользователя, да и к тому же Google серьезно стимулирует переход к более широкому применению HTTPS. Если ваш сайт принимает текстовые данные, но при этом HTTPS не используется, ваш сайт вряд ли получит высокий рейтинг по результатам поиска в Google. Кроме того, в браузере Chrome подобные сайты отмечаются значком «Незащищенный» (Not Secure), отображаемым в верхней панели браузера.

Протокол HTTPS работает в тандеме с другим протоколом — SSL (Secure Socket Layer, уровень защищенных сокетов), который должен выполняться на сервере для обработки защищенных транзакций. Хостинговые компании располагают возможностями подключения SSL, причем часто на бесплатной основе.

Помните, что протокол HTTPS защищает данные форм при их отправке на сервер, но не предпринимает никаких действий для гарантирования защиты вашего сайта и его охраны от хакеров.

Указание файлов, заданных по умолчанию

Многие интернет-адреса не включают имени файла, а просто указывают на каталог, например:

<http://www.oreilly.com>

<http://www.jendesign.com/resume/>

Как только сервер получает запрос имени каталога, а не конкретного файла, система по умолчанию разыскивает в этом каталоге документ, обычно имеющий имя: **index.html**. Поэтому при вводе в адресной строке браузера указанных интернет-адресов на самом деле там отображаются следующие записи:

<http://www.oreilly.com/index.html>

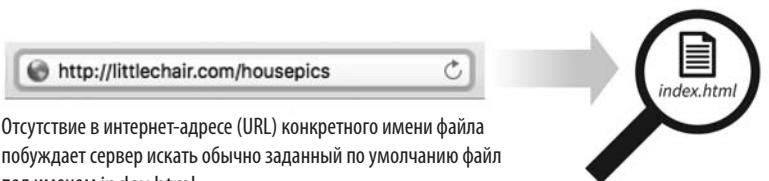
<http://www.jendesign.com/resume/index.html>

Имя файла, который используется по умолчанию (его также называют *индексным файлом*), может быть иным и зависит от конфигурации сервера. В приведенных примерах он называется **index.html**, но некоторые серверы используют в качестве имени файла **default.htm**. Если на вашем сайте для генерирования веб-страниц применяется серверное программирование, индексный файл может называться **index.php** или **Default.aspx**. Обратитесь к администратору сервера или в отдел технической поддержки хостинга, чтобы убедиться, что для файла по умолчанию у вас указано правильное имя.

Следует также отметить, что в первом примере исходный интернет-адрес не имел косой черты (слэша), указывающей на то, что это каталог. Если косая черта не указана, сервер проверяет, является ли запрос файлом или каталогом. Если это каталог, сервер просит браузер снова отправить запрос с косой чертой. Поэтому косая черта в конце имени каталога так или иначе появится, даже если она не была введена при исходном запросе.

НЕ ПРЕНЕБРЕГАЙТЕ СЛЭШАМИ!

Если нужно минимизировать количество обращений к серверу, добавьте в конце имен каталогов в интернет-адресах символы слешей.



Некоторые серверы настроены таким образом, что если в каталоге заданный по умолчанию файл не найден, возвращают список содержимого этого каталога.



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

Индексный файл также полезно иметь в целях обеспечения безопасности. Некоторые серверы (в зависимости от их конфигурации) отображают содержимое каталога, если указанный по умолчанию файл не найден. На рис. 2.2 показано, что в случае отсутствия файла по умолчанию документы, находящиеся в каталоге **housepics**, отображаются в выводе веб-страницы. Один из способов предотвращения неприятностей при поиске ваших файлов в том и состоит, чтобы в каждом каталоге находился индексный файл. Администратор вашего сервера также может добавить другие средства защиты, что предотвратит отображение в окне браузера содержимого ваших каталогов.

АНАТОМИЯ ВЕБ-СТРАНИЦЫ

Все мы знакомы с тем, как веб-страницы выглядят в окне браузера, — теперь пришло время разобраться в том, как они устроены.

На рис. 2.3, *a* показана небольшая веб-страница, отображаемая в окне графического браузера. Несмотря на то что эта веб-страница отображается как одно целое, на самом деле она состоит из четырех отдельных файлов:

- текстового документа HTML — index.html;
- таблицы стилей — kitchen.css;
- двух изображений: foods.png и spoon.png



a



б



в

Рис. 2.3, а — простая веб-страница, состоящая из файла исходного HTML-кода (листинг 2.1), таблицы стилей (листинг 2.2) и изображений: foods.png (б) и spoon.png (в)

А запускает все это шоу HTML-документ — теги, находящиеся в исходном HTML-документе, дают браузерам инструкции о том, как структурирован текст и где должны размещаться изображения.

Документы HTML

Многих изначально удивляет тот факт, что графически насыщенные и интерактивные страницы, которые можно видеть во Всемирной паутине, создаются с помощью простых текстовых документов. Среди специалистов текстовый файл называется *исходным документом*.

Давайте рассмотрим файл index.html (листинг 2.1), который является исходным документом для веб-страницы Jen's Kitchen (Кухня Джен). Этот документ включает текстовый контент (содержимое) страницы и специальные теги (обозначены угловыми скобками: < >), которые описывают каждый элемент веб-страницы.

Листинг 2.1. Файл index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
```

```
<title>Jen's Kitchen</title>
<link rel="stylesheet" href="kitchen.css" type="text/css">
</head>
<body>
<h1> Jen's Kitchen</h1>
<p>If you love to read about <strong>cooking and eating</strong>, would like to learn about some of the best restaurants in the world, or just want a few choice recipes to add to your collection, <em>this is the site for you!</em></p>
<p> Your pal, Jen at Jen's Kitchen</p>
<hr>
<small>Copyright 2018, Jennifer Robbins</small>
</body>
</html>
```

В результате добавления к текстовому документу описательных тегов как раз и появляется «разметка» документа. Веб-страницы используют язык разметки, называемый *гипертекстовым языком разметки* (HyperText Markup Language), или сокращенно HTML, который специально создан для документов, содержащих гипертекстовые ссылки. Язык HTML определяет десятки текстовых элементов, которые составляют такие документы: заголовки, абзацы, выделенный текст и, конечно же, ссылки. Имеются также элементы, вносящие информацию о документе (например, его заголовок), мультимедийные элементы (например, изображения и видео) и виджеты для ввода данных в формы, — и это лишь краткий перечень возможностей этого языка.

При желании можно просматривать исходный код любой веб-страницы. Обратитесь к *упражнению 2.1*, в котором приведено несколько подсказок и указаний по просмотру исходного кода.

В настоящее время используется версия HTML5. Начиная с момента появления HTML в 1989-м году увидело свет несколько версий этого языка разметки, причем некоторые из них используются до сих пор.¹

УПРАЖНЕНИЕ 2.1. ПРОСМОТР ИСХОДНОГО КОДА

Можно просмотреть файл HTML любой веб-страницы с помощью браузера настольного компьютера (ноутбука). Большинство современных браузеров поддерживают функцию просмотра исходного кода (в составе инструментов разработчика), обычно открывая исходный код в отдельном окне или на панели разработчика в нижней части текущего окна.

¹ Вполне исчерпывающая история развития HTML, всех его версий, а также обзор понятий, придающих уникальность HTML5, представлена в приложении 4.

Далее представлены способы получения доступа к функции просмотра исходного кода в большинстве настольных браузеров:

- Safari: **Рабочий стол | Показать | Исходный код страницы** (Develop | Show | Page Source).
- Chrome: **Вид | Разработчик | Просмотр исходного кода** (View | Developer | View Source).
- Firefox: **Инструменты | Веб-разработка | Исходный код страницы** (Tools | Web Developer | Page Source).
- MS Edge: щелкните правой кнопкой мыши на странице и выберите параметр **Просмотр исходного кода** (View Source). Если этот параметр не отображается в контекстном меню, возможно понадобится активизировать его в разделе **Настройки разработчика** (Developer Settings). Откройте новое окно браузера и в строке адреса введите: about :flags. В разделе **Настройки разработчика** выберите контекстное меню и установите флажки **Просмотр исходного кода** (Show View source) и **Инспектировать элемент** (Inspect element). Теперь, когда вы находитесь на веб-странице, можете щелкнуть правой кнопкой на странице, и вы получите доступ к функции просмотра исходного кода. Для получения доступа к этой функции можно также воспользоваться комбинацией клавиш <Ctrl>+<U> или клавишей <F12>.

1. Выбрав браузер, в его адресной строке введите следующий интернет-адрес (URL):

www.learningwebdesign.com/5e/kitchen.html

На экране появится веб-страница Jen's Kitchen, показанная на рис. 2.3, а.

2. Следуйте инструкциям для вашего браузера, приведенным ранее, чтобы просмотреть исходный HTML-код веб-страницы Jen's Kitchen. Он должен быть таким же, как показано в листинге 2.1.
3. Чтобы просмотреть страницу, которая устроена немного сложнее, взгляните на исходный код домашней страницы по адресу: learningwebdesign.com.
4. Исходный код для большинства сайтов значительно сложнее. Например, обратите внимание на исходный код сайта oreilly.com. Здесь есть таблицы стилей, сценарии, встроенная SVG-графика ... и все это работает! Не беспокойтесь, если вы не понимаете, что происходит. Многое из этого будет выглядеть более знакомым к тому времени, когда вы закончите читать эту книгу.

НЕ ЗАИМСТВУЙТЕ ЧУЖОЙ КОД!

Необходимо учитывать следующее обстоятельство: учеба на примерах чужого кода только приветствуется, но заимствование чужого кода выходит за рамки порядочности (и даже является незаконным). Если вам необходим найденный где-то код, попросите разрешения воспользоваться им у автора. Относитесь с уважением к труду разработчиков кодов.

Краткое введение в HTML-разметку

Создание HTML-разметки подробно рассматривается в *части II* книги, поэтому, не фиксируясь на подробностях, отмечу лишь несколько важных моментов. В этом разделе пойдет речь лишь о том, как работает HTML и каким образом браузеры интерпретируют HTML-разметку.

Ознакомьтесь с HTML-документом, который представлен в листинге 2.1, и сравните его с веб-страницей, отображенной в окне вашего браузера. Нетрудно заметить, что в окне браузера отображаются элементы, размеченные тегами HTML в исходном документе.

Во-первых, можно сразу отметить, что находящийся в угловых скобках текст (например: `<body>` и ``) не отображается в результирующей странице. Браузер отображает лишь то, что находится между тегами, — а именно, контент элемента, а разметка остается скрытой от глаз пользователя. Тег определяет имя HTML-элемента, обычно в виде аббревиатуры: например, `h1` — для «заголовка уровня 1», или `em` — для «выделенного текста».

Во-вторых, большинство тегов HTML отображаются парами, окружающими контент элемента. В нашем HTML-документе тег `<h1>` указывает на то, что следующий текст должен быть представлен как заголовок первого уровня, а тег `</h1>` указывает на завершение заголовка. Некоторые элементы, называемые *пустыми*, не имеют контента. В нашем примере тег `<hr>` указывает на пустой элемент, который лишь предписывает браузеру необходимость в качестве тематического разделителя «вставить сюда горизонтальное правило» (*insert a horizontal rule here*).

Когда я начинала использовать HTML, я не знала азов компьютерного программирования, поэтому теги и текст представляла в виде «бусинок на веревочке», которые браузер последовательно интерпретирует одну за другой. Например, если браузер обнаруживает открытую угловую скобку: `<`, предполагается, что последующие символы являются частью разметки, пока не найдется закрывающая скобка: `>`. Аналогично, весь следующий за открывающим тегом `<h1>` контент служит заголовком, пока не встретится закрывающий тег `</h1>`. Именно таким образом браузер и анализирует HTML-документ. Понимание методики работы браузера полезно при устранении неточностей в некорректно функционирующем HTML-документе.

Где же изображения?

Очевидно, что в HTML-файле нет изображений, а как же они тогда попадают на веб-страницу? Как сказано в начале *разд. «Анатомия веб-страницы*», каждое изображение представлено отдельным файлом: `foods.png` (см. рис. 2.3, б) и `spoon.png` (см. рис. 2.3, в). Изображения помещаются в потоке текста с указанием элемента изображения HTML `img`, который и сообщает браузеру о том, где найти изображение (а именно, его интернет-адрес). Когда браузер встречает элемент `img`, он направляет серверу другой запрос — уже о файле изображения и помещает его в поток контента.

Запросы, направляемые браузером серверу, относятся также к таблицам стилей (например, `kitchen.css`), файлов JavaScript (`*.js`) и другим встроенным мультимедийным

элементам — таким, как аудио и видео. Программное обеспечение браузера (или, более конкретно, его механизм визуализации) и объединяет все составляющие части в результирующую веб-страницу.

Сборка страницы обычно выполняется очень быстро, поэтому и создается впечатление, что сразу загружается вся страница. При замедленной сборке страницы или в том случае, когда страница содержит большие графические или мультимедийные файлы, процесс сборки идет медленнее, и тогда появление изображений отстает от текстовой информации. Страница может даже заново создаваться при использовании новых изображений, шрифтов и таблиц стилей (хотя в ваших силах сформировать страницы так, чтобы избежать этого).

Добавление стилей

Обратите внимание на ключевой компонент минимальной веб-страницы. В верхней части HTML-документа (см. листинг 2.1) располагается элемент ссылки, указывающий на документ таблицы стилей: `kitchen.css`. Эта таблица стилей включает несколько строк инструкций, определяющих внешний вид страницы в браузере (листинг 2.2). Стилевые инструкции составлены с учетом правил формирования каскадных таблиц стилей (CSS, Cascading Style Sheets). Таблицы стилей CSS позволяют дизайнерам вносить в размеченный текст (структуру документа — по терминологии веб-дизайна) визуальные стилевые инструкции (или *представления* документа).

В третьей части книги вы более подробно познакомитесь с универсальными возможностями каскадных таблиц стилей.

Листинг 2.2. Файл `kitchen.css`

```
body { font: normal 1em Verdana; width: 80%; margin: 1em auto; }
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; }
```

На рис. 2.4 показана веб-страница Jen's Kitchen: без использования стилевых инструкций (рис. 2.4, а) и включающая эти инструкции (рис. 2.4, б). Браузеры располагают



а



б

Рис. 2.4. Веб-страница Jen's Kitchen: без применения (а) и с применением пользовательских стилевых правил (б)

стилями, заданными по умолчанию для каждого элемента HTML, который они поддерживают, поэтому, если в документе HTML отсутствуют инструкции по пользовательскому стилю, браузер применяет собственный стиль. Обратите внимание на верхнюю часть экранного снимка — применение лишь нескольких стилевых правил значительно улучшает внешний вид веб-страницы.

Добавление поведений с помощью JavaScript

Для активизации элементов на странице применяется язык сценариев — JavaScript. На веб-странице Jen's Kitchen сценарии отсутствуют, поскольку вы только приступаете к веб-дизайну, но следует иметь в виду, что JavaScript является важным компонентом современных веб-сайтов.

Язык разметки HTML формирует структуру страницы, таблица стилей CSS — внешний вид, а JavaScript вносит компонент *поведения*, управляющий всеми аспектами управления веб-страницами.

Сценарии можно как записывать в документ, так и включать в автономные файлы на сервере (с расширением js). Они могут запускаться сразу после загрузки страницы или же после щелчка пользователя на элементе, при наведении указателя мыши на элемент, а также при вводе данных в поле формы. Базовое введение в JavaScript приводится в части IV книги.

ЯЗЫК JAVASCRIPT ТРЕБУЕТСЯ НЕ ВСЕГДА
Язык JavaScript не требуется для организации взаимодействия со ссылками и веб-формами, которые функционируют только с использованием HTML.

СОБИРАЕМ ВСЕ ВМЕСТЕ

И в завершение введения в работу Всемирной паутины проследим за обычным потоком событий, которые позволяют веб-странице отобразиться на экране вашего браузера (рис. 2.5).

- ➊ Для запроса веб-страницы непосредственно в адресной строке браузера вводится интернет-адрес (URL) — например: `http://jenskitchensite.com`, или щелчком мыши (касанием) выбирается имеющаяся на странице ссылка. Интернет-адрес содержит информацию для определения целевого документа, находящегося на некотором веб-сервере в Интернете. В нашем случае адрес указывает на файл, заданный по умолчанию (`index.html`), который находится в верхнем каталоге.
- ➋ Ваш браузер отправляет HTTP-запрос указанному в интернет-адресе (URL) серверу и запрашивает конкретный файл. Запрос также содержит информацию о предлагаемых пользователю страницы языках, а также о типах файлов, которые может принимать браузер. Если в интернет-адресе указан каталог (а не файл), это равносильно запросу файла, заданного в этом каталоге по умолчанию.
- ➌ Сервер ищет запрошенный файл и выдает HTTP-отклик в виде HTTP-заголовка. Заголовок включает информацию о файле — например, дату последнего

изменения, величину файла, тип его контента (Content-Type). Так, файл **.html** имеет тип контента «text/html».

- Если страница не обнаружена, сервер возвращает сообщение об ошибке. Обычно это сообщение имеет вид **404 Not Found**, хотя они могут выглядеть и более привлекательно. Возможно появление и иных ошибок (см. врезку «Коды состояния HTTP»).
- Если документ обнаружен, сервер извлекает файл и возвращает его браузеру. Если сайт динамический, сервер сначала собирает страницу из сохраненных данных, а уже затем возвращает ее браузеру.

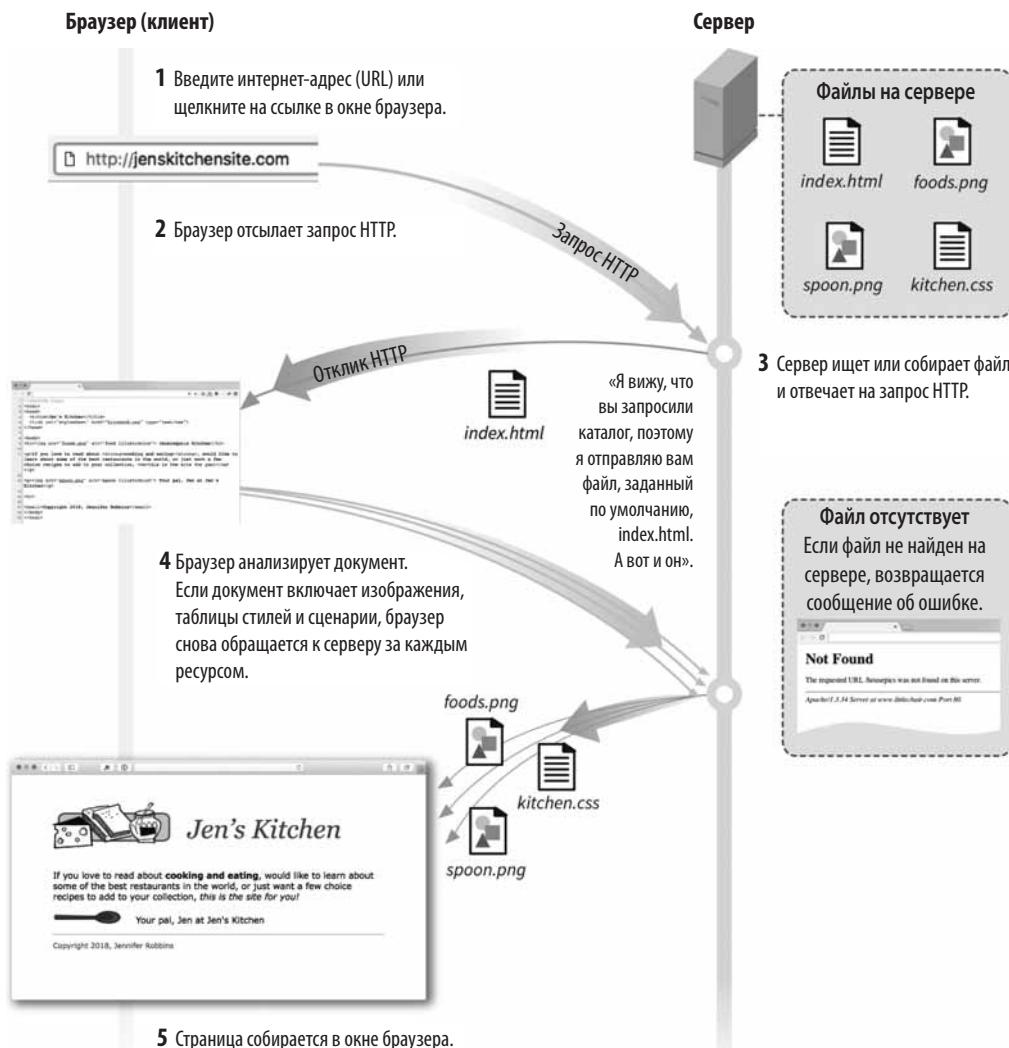


Рис. 2.5. Порядок отображения веб-браузерами веб-страниц

- ④ Браузер анализирует HTML-документ. Если страница содержит изображения (обозначенные HTTP-элементом `img`) или другие внешние ресурсы — такие, как сценарии или таблицы стилей, браузер снова обращается к серверу для запроса каждого указанного в разметке ресурса.
- ⑤ Браузер вставляет изображение в поток документов, обозначенный элементом `img`, применяет стили и запускает сценарии. И на этом все! Собранная веб-страница отображается для просмотра.

СТАТИЧЕСКИЕ И ДИНАМИЧЕСКИЕ САЙТЫ

Статические веб-сайты формируются из HTML-файлов с фиксированным контентом, которые отображают одну и ту же информацию для каждого посетителя. То есть каждая отображенная в браузере страница представлена на сервере одним HTML-файлом. В книге рассматриваются процедуры, приводящие к формированию статических веб-страниц — создавать их проще, и они служат удобным отправным пунктом для начинающих.

В отличие от статических сайтов, динамические веб-сайты создаются с применением внутреннего программирования — такого, как PHP или ASP. Каждая страница генерируется приложением, что называется, «с ходу». Динамические сайты получают доступ к контенту и данным из базы данных, а полученные в результате страницы можно подстраивать с учетом запросов конкретного пользователя. Для очень больших сайтов, включающих сотни и даже тысячи страниц, процедуры настройки и поддержки, в случае обращения к динамическому сайту, требуют значительно меньше трудозатрат, чем формирование каждой отдельной страницы и ее сохранение в виде статического HTML-документа.

Следует отметить, что здесь приведен обычный несложный сценарий сборки веб-страниц. Современные веб-страницы часто собираются с помощью систем управления контентом (CMS, content management systems), которые сохраняют контент в базах данных и для ускоренного сбора данных применяют шаблоны. Тогда на этапе 3, b реализуется более сложный процесс сборки веб-страницы из различных частей, а не передача уже имеющегося файла.

КОДЫ СОСТОЯНИЯ HTTP

Серверы сообщают коды состояния в ответ на запросы браузера. Полный список кодов состояния весьма обширен — исчерпывающая информация о кодах состояния HTTP представлена в Википедии: ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP, а здесь приведен небольшой перечень общих откликов:

- 200 OK
- 301 Moved Permanently (Перемещено на постоянной основе)
- 302 Moved Temporarily (Перемещено временно)
- 404 Not Found (Не найдено)
- 410 Gone (no longer available) (Удалено (больше не доступно))
- 500 Internal Server Error (Внутренняя ошибка сервера)

РАЗМЕЩЕНИЕ ВАШЕЙ ВЕБ-СТРАНИЦЫ В ИНТЕРНЕТЕ

Если нужно получить дополнительную информацию о регистрации доменных имен и поиске сервера для размещения сайта, загрузите статью под названием «Getting Your Pages on the Web» (PDF), доступную по адресу: learningwebdesign.com/articles/.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Давайте поиграем в игру «Узнайте эту аббревиатуру!». Далее приведены несколько основных интернет-терминов, упомянутых в этой главе. Укажите правильный вариант ответа, выбрав его в нижнем списке. Ответы на эти вопросы приведены в *приложении 1*.

1. HTML _____.
2. W3C _____.
3. ЦЕРН _____.
4. CSS _____.
5. HTTP _____.
6. IP _____.
7. URL _____.
8. NCSA _____.
9. DNS _____.
10. FTP _____.

- a. Домашняя страница Mosaic, первого графического браузера.
- b. Местоположение веб-документа или ресурса.
- c. Язык разметки, применяемый для описания веб-контента.
- d. Сопоставление доменных имен с числовыми IP-адресами.
- e. Протокол, применяемый для передачи файлов.
- f. Протокол, применяемый для передачи веб-документов в Интернете.
- g. Язык, применяемый для инструктирования браузеров о формировании внешнего вида веб-контента.
- h. Европейский центр ядерных исследований, в стенах которого родилась Всемирная паутинка.
- i. Протокол Интернета.
- j. Организация, которая отслеживает интернет-технологии.

ГЛАВА 3

ВАЖНЫЕ КОНЦЕПЦИИ

В этой главе...

- ▶ Доступ к Интернету с мобильных устройств
- ▶ Соблюдение интернет-стандартов
- ▶ Постепенное улучшение
- ▶ Адаптивный (отзывчивый) веб-дизайн
- ▶ Специальные возможности
- ▶ Быстродействие сайта

По мере развития Всемирной паутины и безудержного роста количества устройств, с помощью которых мы подключаемся к ней, наша работа в качестве веб-дизайнеров и веб-разработчиков значительно усложняется. Все, что связано с Всемирной паутиной, настолько многообразно, что для описания этого понадобится не одна книга. Поэтому важно сосредоточиться на главном, без чего трудно обойтись в работе. В следующих главах я сосредоточусь на основных строительных блоках веб-дизайна: элементах HTML, стилях CSS, JavaScript и создании изображений для веб-сайтов. Все это даст вам прочную основу для дальнейшего развития навыков веб-дизайнера.

Прежде чем перейти к сути, я хочу представить вам несколько важных концепций, которые должен знать каждый веб-дизайнер. Мы рассмотрим идеи и проблемы, которые определяют наши решения и вносят вклад в современную интернет-среду. На протяжении всей книги я буду возвращаться к терминологии, представленной в этой главе.

Суть проблемы заключается в том, что мы, как веб-дизайнеры, никогда не знаем точно, как будут просматриваться создаваемые нами веб-страницы. Мы не знаем, какой из десятков браузеров будет использоваться, какое устройство будет для этого применяться (компьютер или более портативное), насколько большим будет окно браузера, какие шрифты установлены, активизированы ли такие функциональные свойства, как JavaScript, какова скорость подключения к Интернету, прослушиваются ли веб-страницы с помощью программы чтения с экрана и так далее. Важные концепции, рассматриваемые в этой главе, — прежде всего, являются реакцией на методы преодоления непредсказуемого элемента Неизвестности во Всемирной паутине. Эти концепции включают следующие компоненты:

- множественность устройств;
- интернет-стандарты;
- постепенное улучшение;
- адаптивный (отзывчивый) веб-дизайн;
- специальные возможности;
- быстродействие сайта.

Поскольку мы только начинаем учиться веб-дизайну, описания будут краткими и нетехническими. Моя цель состоит в том, чтобы у вас появилось базовое понимание термина «постепенное улучшение» (progressive enhancement). По каждой из этих тем и связанным с ними технологиям написано много прекрасных статей и книг, и вы получите ссылки на углубленные ресурсы для дальнейшего чтения.

МНОЖЕСТВЕННОСТЬ УСТРОЙСТВ...

Еще до 2007-го года веб-дизайнеры могли быть практически уверены в том, что пользователи посещают сайты, сидя за рабочими столами, просматривая сайты на больших мониторах и используя быстрое подключение к Интернету. Ширина веб-страницы в те времена преимущественно составляла 960 пикселов — эта величина была основана на наиболее распространенном размере монитора. В те времена наибольшей проблемой была поддержка дюжины или около того браузеров для настольных компьютеров и выполнение нескольких дополнительных действий для поддержки причудливых старых версий Internet Explorer. Как видите, в те времена заниматься веб-дизайном было не так уж и сложно.

Хотя доступ к веб-страницам и интернет-контенту на мобильных телефонах можно было получить и до 2007-го года, появление смартфонов iPhone и Android, а также более быстрых сетей привело к революции в интернет-серфинге (особенно масштабные сдвиги произошли в США, которые отставали от Азии и стран Евросоюза в мобильных технологиях). С тех пор постоянно появляются телефоны и планшеты разных размеров, а также внедряются веб-браузеры на телевизорах, игровых приставках и прочих устройствах. И это разнообразие будет только расти. Обратите внимание на то, каким образом эксперт в области мобильного веб-дизайна Брэд Фрост (Brad Frost) иллюстрирует эти факты (рис. 3.1).

Задачи, связанные с выполнением веб-дизайна для всех этих устройств, выходят за рамки решения проблем, связанных с различными размерами экрана. Существует большая разница между использованием сайта, доступного по широкополосному соединению и по медленной сотовой сети. В процессе разработки дизайнеры не должны связывать быстродействие и контент сети с размером экрана. Владельцы устройств с маленьким экраном далеко не всегда пользуются медленным подключением к Интернету или торопливо просматривают веб-страницы в транспорте. Нередко они часами могут просматривать сайты на смартфоне со сравнительно не-



Рис. 3.1. Брэд Фрост иллюстрирует разнообразие мобильных устройств (bradfrostweb.com)

большим экраном, усевшись на диване и пользуясь высокоскоростным подключением к Wi-Fi. К тому же планшеты типа iPad с большими экранами, обладающими высоким разрешением, могут подключаться к Интернету через 3G/4G-соединения. Другими словами, весьма сложно предвидеть все возможные варианты использования мобильных устройств для просмотра веб-страниц.

В настоящее время доступ к Всемирной паутине получают преимущественно с мобильных устройств, а не с настольных компьютеров. Например, значительная часть населения США подключается к Интернету исключительно с помощью своих смартфонов. Это означает, что очень важно правильно выбрать дизайн и функциональность сайта. Мы добились огромных успехов в предоставлении приятного опыта пользователям мобильных устройств, причем технология, предназначенная для удовлетворения их потребностей, продолжает развиваться в правильном направлении.

Я хочу, чтобы вы были в курсе того, что восприятие дизайна сайта на экране настольного компьютера отличается от восприятия дизайна сайта на экране смартфона. Пользователи некоторых мобильных устройств будут видеть элементы сайта намного меньшими, чем пользователи настольных компьютеров. Некоторым из пользователей будет казаться, что сайт загружается мучительно долго. Другие же будут просматривать сайты на экране телевизора, установленном в противоположном углу комнаты. Всем профессиональным веб-дизайнерам следует знать и помнить об этом.

РАЗМЕР НЕ ИМЕЕТ ЗНАЧЕНИЯ!

Не судите о быстродействии и контенте сети на основе размера экрана.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Найдите книгу «Mobile First», написанную Люком Вроблевски (Luke Wroblewski). Люк одним из первых стал утверждать, что сайты хорошо отображаются на экранах мобильных устройств, и своими взглядами он делится в этой маленькой книге, которая просто «нафарширована» идеями.

МОБИЛЬНЫЙ ИНТЕРНЕТ?

Вы, наверное, неоднократно слышали, как многие люди используют термин «мобильный Интернет», хотя на самом деле — как писал Стивен Хэй (Stephen Hay) в своем твите, опубликованном в 2011-м году, — не существует мобильного Интернета, настольного Интернета, планшетного Интернета или чего-то другого в этом роде (рис. 3.2). Существует единый Интернет, к которому можно получить доступ с разных устройств. Поэтому термин «мобильный Интернет» используется нами для описания усилий, направленных на переориентацию наших навыков построения веб-дизайна для настоль-



Stephen Hay
@stephenhay

Following



There is no Mobile Web. There is only The Web,
which we view in different ways. There is also no
Desktop Web. Or Tablet Web. Thank you.



Reply



Retweet



Favorite

ных компьютеров, на более широкий спектр вариантов применения. И, как мы выяснили, существует несколько способов осуществления подобной переориентации.

Рис. 3.2. Твит Стивена Хэя, написанный в январе 2011-го года. Этот твит доступен по следующей ссылке: www.the-haystack.com/2011/01/07/there-is-no-mobile-web

СОБЛЮДЕНИЕ ИНТЕРНЕТ-СТАНДАРТОВ

Как же все-таки справиться с разнообразием мобильных устройств и интернет-приложений? Для начала нужно придерживаться стандартов, принятых Консорциумом World Wide Web (W3C). Соблюдение стандартов — это основное средство для обеспечения совместимости сайта со всеми браузерами, соответствующими этим стандартам (это примерно 99% браузеров, используемых в настоящее время). Это также помогает сделать ваш контент более совместимым с развивающимися интернет-технологиями и возможностями браузеров. Еще одно преимущество заключается в том, что в этом случае вы можете сказать своим клиентам, что вы создаете «соответствующие стандартам» сайты, и вы им понравитесь еще больше.

На первый взгляд, понятие «соответствие стандартам» может показаться простым, но тут следует учитывать, что раньше все, включая производителей браузеров, легко и свободно обращались с HTML и со сценариями. Использование несовместимых реализаций браузеров и необходимость создавать несколько копий сайтов, ориентированных на разные стандарты, обходится недешево. Поскольку к интернет-стандартам мы будем обращаться на протяжении всей книги, в этом разделе подробно мы их рассматривать не станем.

СОБЛЮДАЙТЕ ИНТЕРНЕТ-СТАНДАРТЫ

Соблюдение интернет-стандартов — это главное средство, позволяющее обеспечить максимально возможную согласованность разрабатываемого сайта.

Достаточно сказать, что нужно придерживаться интернет-стандартов, и все будет хорошо. Так что все, что вы узнаете из этой книги, поможет вам сделать свой первый шаг с правильной ноги.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

- На сайте W3C (w3.org/standards) сосредоточена основная информация, относящаяся ко всем интернет-стандартам.
- Все, что связано с проблемами соответствия стандартам и методами достижения такого соответствия, изложено в книге «Designing with Web Standards», 3-е издание. Эта книга написана Джейфри Зельдманом (Jeffrey Zeldman) и увидела свет в издательстве New Riders. Со времени выхода книги прошло много лет, но изложенная в этой книге информация не стареет с годами.

ПОСТЕПЕННОЕ УЛУЧШЕНИЕ

С появлением множества браузеров возникает множество уровней поддержки интернет-стандартов. На самом деле, ни в одном из браузеров не реализованы все стандарты на 100%, и всегда есть новые технологии, которые начинают применяться на практике. Кроме того, пользователи могут задавать браузерам собственные настройки — например, браузер поддерживает JavaScript, но эта поддержка может быть отключена пользователем. К тому же, разные браузеры обладают самыми разными возможностями: от базового HTML до поддержки всех модных «примочек».

Постепенное улучшение — это одна из стратегий, предназначенных для работы с неизвестными возможностями браузера (см. врезку «Постепенное улучшение»).

Используя постепенное улучшение, вы начинаете с базового опыта, который делает доступным контент или основные функциональные возможности даже для самых элементарных браузеров или вспомогательных устройств. Затем вы добавляете более сложные функции, предназначенные для браузеров, которые могут их реализовать. Ну и напоследок можно добавить некоторые эффекты, которые хотелось бы иметь, — такие, как анимация или обтекание текста вокруг изображений интересными фигурами, которые облегчают работу пользователей с самыми продвинутыми браузерами, но не слишком нужны для работы с браузерами на начальном уровне.

Итак, постепенное улучшение — это подход, который информирует обо всех аспектах дизайна и создания веб-страниц, включая HTML, CSS и JavaScript:

- ▶ *стратегия авторинга* — документ HTML, созданный с соблюдением логики вывода всех его соответственно размеченных элементов, будет правильно открываться в самых разных средах просмотра, включая устаревшие браузеры, современные браузеры, а также мобильные и вспомогательные устройства. Может, в вашем случае ситуация немного отличается, но тут важно то, что ваш контент будет всегда доступен. Это также гарантирует, что поисковые системы, такие, как Google, будут корректно каталогизировать контент. Чистый HTML-документ с его точными и подробно описанными элементами является основой для формирования специальных возможностей;
- ▶ *стратегия стайлинга* — вы можете создавать несколько уровней опыта, просто воспользовавшись методикой анализа браузерами правил таблиц стилей. Не вдаваясь в технические подробности, можно создать правило стиля, которое окрашивает фон элемента в красный цвет, а также включить стиль, который придает ему красивый градиент (переход между цветами) для браузеров, которые могут отображать градиенты. Или же можно использовать современный CSS-селектор, чтобы передавать определенные стили только в самые современные браузеры. Знание того, что браузеры просто игнорируют свойства и правила, которые они не понимают, дает вам лицензию на применение инноваций, не отказываясь от устаревших браузеров. Вам просто нужно помнить сначала о стиле базового интерфейса, а затем добавлять улучшения по мере того, как будут выполнены минимальные требования.
- ▶ *стратегия написания сценариев* — как и в случае с другими интернет-технологиями, имеет место нестыковка между тем, как браузеры обрабатывают JavaScript (особенно на

ПОСТЕПЕННОЕ УЛУЧШЕНИЕ

Постепенное улучшение — это оборотная сторона подхода к разнообразию браузеров, называемого постепенной деградацией, который предполагает изначальную разработку полностью улучшенного интерфейса, с последующим созданием набора запасных вариантов для браузеров, которые не поддерживают этот интерфейс. Оба метода используются в современном веб-дизайне. В этой книге излагаются множество альтернативных методов, позволяющих обеспечивать работоспособность самых разных браузеров.

ИСПОЛЬЗУЙТЕ ПОСТЕПЕННОЕ УЛУЧШЕНИЕ

Постепенное улучшение — это стратегия, которая приходит на помощь при наличии браузеров с неизвестными возможностями.

мобильных устройствах), и ситуацией, когда некоторые пользователи предпочитают полностью отключать поддержку JavaScript. Первое правило постепенного улучшения: убедиться в том, что базовая функциональность — такая, как ссылки, ведущие со страницы на страницу, или выполнение важных задач — таких, как отправка данных через формы, не нарушаются, даже если поддержка JavaScript отключена. Таким образом, вы обеспечиваете базовый опыт и улучшаете его, когда поддержка JavaScript включена.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

- Не существует лучшего введения в методику постепенного улучшения, чем книга Аарона Густафсона (Aaron Gustafson) «Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement», 2-е издание, издательство New Riders.
- Книга Роба Ларсона (Rob Larson) «The Uncertain Web: Web Development in a Changing Landscape», издательство O'Reilly.
- Как только вы приобретете небольшой опыт в области веб-дизайна, обратитесь к книге Тодда Паркера (Todd Parker), Пэтти Толанд (Patty Toland), Скотта Джела (Scott Jehl) и Мэгги Костелло Вакс (Maggie Costello Wachs) «Designing with Progressive Enhancement», издательство New Riders. Эта книга обеспечивает превосходное глубокое погружение в методики и лучшие практики по затронутым темам. Дополнительные сведения об этой книге можно найти на сайте: filamentgroup.com/dwpe/.

АДАПТИВНЫЙ ВЕБ-ДИЗАЙН

По умолчанию большинство браузеров, установленных на небольших устройствах, — таких, как смартфоны и планшеты, уменьшают размер веб-страницы до размеров экрана и предоставляют механизмы для масштабирования и перемещения по веб-странице. Хотя эта технология вполне рабочая, это не очень хороший опыт. Текст слишком мал для чтения, ссылки слишком малы, чтобы попасть на них пальцем или стилусом, и все эти изменения масштаба и прокрутки отвлекают пользователя от работы.

Адаптивный (отзывчивый) веб-дизайн (RWD, Responsive Web Design) — это стратегия предоставления устройствам соответствующих макетов в зависимости от размера области просмотра (окна браузера). Ключом к адаптивному веб-дизайну является предоставление одного HTML-документа с одним интернет-адресом (URL) всем устройствам, но применение разных таблиц стилей в зависимости от размера экрана, чтобы обеспечить наиболее оптимальный макет для каждого устройства. Например, когда веб-страница просматривается на смартфоне, она отображается в одном столбце с большими ссылками, обеспечивающими удобное касание. Но когда эту же страницу просматривают в браузере большого настольного компьютера, содержимое перегруппируется в несколько столбцов с традиционными элементами навигации. Со стороны это выглядит как волшебство! (На самом деле это всего лишь CSS.)

Сейчас сообщество веб-дизайнеров просто гудит по поводу отзывающего дизайна. Впервые же об этой технологии написал Итан Маркотт (Ethan Marcotte) в статье

«Responsive Web Design», опубликованной на сайте A List Apart в 2010-м году (alistapart.com/articles/responsive-web-design/). Эта технология обычно используется в тех случаях, когда изначально неизвестны размеры области просмотра.

Строго говоря, английский термин «responsive» ближе к русскому «отзывчивый», и в сообществе веб-дизайнеров не утихают споры, какой же перевод ближе отражает смысл технологии, и не две ли разные технологии скрываются под терминами «Adaptive Web Design» (адаптивный веб-дизайн) и «Responsive Web Design» (отзывчивый веб-дизайн). Рекомендуем познакомиться с интересной статьей на эту тему по адресу: <https://habr.com/ru/post/148224/>, в которой анализируются тонкости технологий, описанных в уже упомянутых книгах Аарона Густафсона и Итана Маркотта. Мы же здесь и далее будем в основном называть рассматриваемую технологию «адаптивный веб-дизайн»¹.

На рис. 3.3 показано несколько примеров адаптивных сайтов, имеющих типичные размеры для настольного монитора, планшета и смартфона. Множество вдохновляющих примеров можно также найти на сайте галереи Media Queries (mediaqueri.es). Попробуйте открыть один из адаптивных сайтов в окне вашего браузера, а затем изменить размер окна по ширине. Обратите внимание, как изменяется макет в зависимости от размера окна. Круто, не правда ли?!

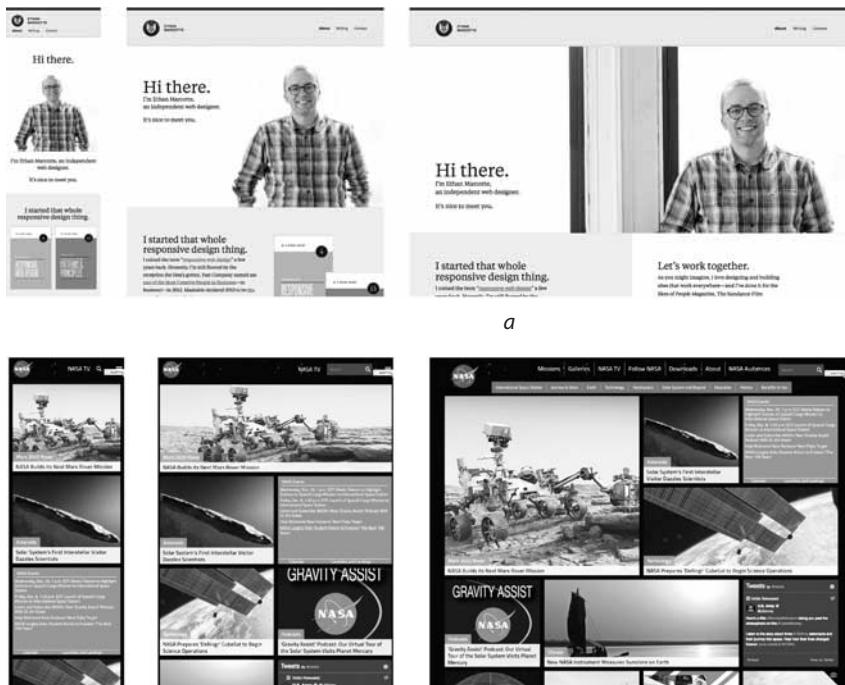


Рис. 3.3. Макет адаптивного сайта изменяется в зависимости от размеров окна браузера: *а* — персональный сайт Итана Маркотта (ethanmarkotte.com); *б* — сайт NASA (nasa.gov)

¹ Адаптивный веб-дизайн или Отзывчивый веб-дизайн — в чем разница? (Прим. ред.).

ИСПОЛЬЗУЙТЕ АДАПТИВНЫЙ ВЕБ-ДИЗАЙН

Благодаря использованию адаптивного веб-дизайна можно создавать сайты, рассчитанные на устройства с разным размером экрана.

Адаптивный веб-дизайн помогает в вопросах верстки, но он не решает всех проблем, связанных с мобильным веб-дизайном. Дело в том, что для обеспечения наилучших впечатлений для ваших пользователей и выбранных ими устройств может потребоваться оптимизация, выходящая за рамки настройки внешнего вида. Вы можете лучше решить некоторые проблемы, используя сервер для определения устройства и его возможностей, а затем принимая решение о том, какой контент отправлять обратно.

Для некоторых сайтов и услуг предпочтительнее создать отдельный мобильный сайт (см. врезку «Сайты M-dot») с настраиваемым интерфейсом и набором функций, который использует преимущества таких функций смартфона, как, например, геолокация. Тем не менее, несмотря на то что адаптивный дизайн не решает все проблемы, он является важной частью решения, обеспечивающего удовлетворительный опыт работы с широким спектром браузеров.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Адаптивный веб-дизайн мы подробно рассмотрим в главе 17, когда вы уже будете иметь опыт проектирования веб-сайтов. Там же вы найдете множество ресурсов, которые помогут вам продолжить образование в области веб-дизайна.

САЙТЫ M-DOT

Некоторые компании и службы предпочитают создавать отдельные сайты с уникальным интернет-адресом (URL), предназначенным только для мобильных устройств. Сайты M-dot (названные так потому, что их URL-адреса обычно начинаются с `m` или `mobile`) предлагают ограниченный набор возможностей и могут также включать специфические для мобильных устройств функции — такие, как геолокация. Многие «лишние» функции (например, рекламные баннеры), которые присутствуют на сайтах, адаптированных для настольных компьютеров, просто устраняются. (Это заставляет задуматься о том, насколько ценная информация добавляется на настольном компьютере.) Отдельный мобильный сайт может быть лучшим решением, если вы знаете, что ваши мобильные пользователи используют шаблоны, отличающиеся от шаблонов, применяемых пользователями настольных компьютеров.

На рис. 3.4 сравниваются основной сайт компании CVS и ее сайт M-dot в том виде, в котором они появились в начале 2018-го года. Вы можете видеть, что пользователям смартфонов предлагается более упрощенный набор параметров. Другие известные сайты, имеющие отдельные мобильные версии: Твиттер и Фейсбук.

Все дело в том, что адаптивный веб-дизайн не является универсальным решением. Для сайтов, которые содержат в основном текстовый контент, достаточно выполнить небольшую настройку макета, чтобы обеспечить хорошее восприятие на всех устройствах. Для сложных сайтов и веб-приложений предпочтительнее пойти другим путем.

Недостаток выделенного мобильного сайта заключается в том, что при его создании понадобится выполнить в два раза больше работы. Понадобится дополнительное



► планирование контента, шаблоны дизайна, время для производства и текущего обслуживания. Но если это означает предоставление вашим посетителям необходимой им функциональности, то тогда «игра стоит свеч».

Возможно, у вас есть бизнес, для которого применение мобильных устройств отличается от использования настольных компьютеров. В этом случае создание отдельного мобильного сайта имеет смысл, но в целом сайты M-dot дрейфуют в сторону адаптивного веб-дизайна. Благодаря Google этот процесс ускоряется, поскольку эта компания в 2018-м году уже побуждала все сайты M-dot перейти на адаптивный веб-дизайн перед запуском их «индекса mobilefirst» (webmasters.googleblog.com/2016/11/mobile-first-indexing.html). Если ранжирование результатов поиска вашего сайта затруднено, переход на адаптивный веб-дизайн может эти результаты улучшить.



Рис. 3.4. Сравнение сайта для настольного компьютера (а) и выделенного мобильного сайта (б), имеющих одинаковый контент

ОДИН ИНТЕРНЕТ ДЛЯ ВСЕХ (ДОСТУПНОСТЬ САЙТОВ)

В этой книге уже упоминалось об огромном количестве браузеров, используемых в настоящее время, но пока еще рассматривались лишь визуальные браузеры, управляемые указателем мыши или кончиками пальцев. Однако при этом крайне важно помнить о том, что люди получают доступ к Интернету различными способами: с помощью клавиатуры, мыши, голосовых команд, программ чтения с экрана, шрифтов Брайля, экранных луп, джойстиков, педалей и т. п. Веб-дизайнерам следует разрабатывать веб-страницы таким образом, чтобы создавать как можно меньше

барьеров для доступа к информации, независимо от возможностей пользователя и устройства, используемого им для обеспечения доступа в Интернет. Другими словами, веб-дизайн должен поддерживать специальные возможности, требуемые для пользователей с особыми потребностями.

Несмотря на то что рассматриваемые в этом разделе способы и стратегии предназначены для пользователей с ограниченными возможностями — например, для людей с плохим зрением или ограниченной подвижностью, они могут быть полезными для других пользователей, имеющих неоптимальный опыт просмотра. Сайты, поддерживающие специальные возможности, также более эффективно индексируются поисковыми системами — такими, как Google. Однако веб-дизайн с учетом использования специальных возможностей также требует дополнительных усилий.

Существует четыре широких категории нарушений здоровья, которые влияют на то, как люди взаимодействуют со своими компьютерами и информацией о них:

- ▶ *нарушения зрения* — люди, плохо видящие или в случае полного отсутствия зрения могут для взаимодействия с экраном задействовать вспомогательные средства: программу экранного чтения, дисплей Брайля или экранную лупу. Они также могут просто использовать функцию масштабирования текста в браузере, чтобы сделать текст достаточно большим для чтения;
- ▶ *ограниченная подвижность* — пользователи, страдающие ограничением подвижности рук либо лишенные возможности пользоваться руками, могут использовать для навигации в сети и ввода информации специальные устройства: модифицированные мыши и клавиатуры, ножные педали, голосовые команды или джойстики;
- ▶ *нарушения слуха* — пользователи, имеющие нарушения слуха либо вовсе лишенные слуха, не могут использовать мультимедийные возможности сайта, поэтому нужно предоставить им альтернативы — такие, как стенограммы звуковых дорожек или титры для видео;
- ▶ *когнитивные нарушения* — пользователи, испытывающие проблемы с памятью, с пониманием прочитанного, решением логических задач и неустойчивым вниманием, только выигрывают, если сайты выглядят просто и четко. Эти характеристики сайтов полезны для всех категорий пользователей, просматривающих сайт.

Консорциум W3C начал разрабатывать Инициативу доступности веб-сайтов (WAI, Web Accessibility Initiative), чтобы решить проблему доступности Интернета для всех. Сайт WAI (www.w3.org/WAI) является отличной отправной точкой для получения дополнительной информации о доступности веб-сайтов. Одним из документов, разработанных WAI в помощь разработчикам для создания доступных сайтов, являются Web Content Accessibility Guidelines (Рекомендации по обеспечению доступности веб-контента), которые вкратце называются WCAG и WCAG 2.0. Все эти документы можно найти на сайте www.w3.org/WAI/intro/wcag.php. Правительство США основывало свои руководящие указания по доступности в Разделе 508 на приоритете пункта 1 WCAG (см. врезку «Требования правительства к доступности»).

Раздел 508»). Соблюдение этих рекомендаций обеспечит преимущества для всех сайтов, но если вы разрабатываете правительственный сайт, то соблюдение их является обязательным.

Еще один документ от W3C, регулирующий доступность сайтов, — это спецификация WAI-ARIA (Accessible Rich Internet Applications, Доступные многофункциональные интернет-приложения), определяющая доступность веб-приложений, включающих динамически генерируемый контент, сценарии и расширенные элементы интерфейса, которые особенно мешают вспомогательным устройствам. В рекомендации ARIA определен ряд элементов для контента и виджетов, которые авторы могут применять явно с помощью атрибута элементов. Элементы эти включают в себя меню, строку процесса, слайдер, таймер и всплывающую подсказку. Полный список элементов можно найти на сайте: www.w3.org/TR/wai-aria/#role_definitions.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Следующие ресурсы являются хорошей отправной точкой для дальнейшего изучения доступности веб-сайтов:

- «Инициатива доступности веб-сайтов» (WAI, Web Accessibility Initiative), www.w3.org/WAI.
- «Доступность веб-сайтов в мыслях» (WebAIM: Web Accessibility in Mind), www.webaim.org.
- Книга Кати Каннингэм (Katie Cunningham) «Accessibility Handbook: Making 508 Compliant Websites», издательство O'Reilly.
- Книга Венди Чизем (Wendy Chisholm) и Мэтта Мэя (Matt May) «Universal Design for Web Applications: Web Applications That Reach Everyone», издательство O'Reilly.

ТРЕБОВАНИЯ К ДОСТУПНОСТИ ВЕБ-САЙТОВ ОТ ПРАВИТЕЛЬСТВА США: РАЗДЕЛ 508

Если вы разрабатываете сайт, получающий федеральное финансирование от правительства США, вы обязаны по закону соблюдать положения Раздела 508, которые обеспечивают доступность электронной информации и технологий для людей с ограниченными возможностями. Государственные и другие финансируемые государством сайты могут также требовать соблюдения этих требований.

Следующие рекомендации взяты из Раздела 508 «Стандарты» (Standarts), доступном на сайте www.section508.gov, и предоставляют хороший контрольный список базовой доступности для всех веб-сайтов.

1. Нужно предоставить текстовый эквивалент для нетекстовых элементов (например, с помощью атрибута «alt» или в содержимом элемента).
2. Эквивалентные альтернативы для любой мультимедийной презентации должны быть синхронизированы с презентацией.
3. Веб-страницы должны быть спроектированы таким образом, чтобы вся информация, передаваемая с помощью цвета, была также доступна без цвета — например, из контекста или разметки.
4. Документы должны быть организованы таким образом, чтобы их можно было читать, не используя связанную таблицу стилей.
5. Заголовки строк и столбцов должны быть идентифицированы для таблиц данных.

- ▶ 6. Разметку нужно применять для связывания ячеек данных и заголовка для таблиц с двумя или более уровнями заголовков строк или столбцов.
- 7. Веб-страницы нужно спроектировать таким образом, чтобы экран не мерцал с частотой более 2 Гц и меньше, чем 55 Гц.
- 8. Если на веб-страницах используются языки сценариев для отображения содержимого или для создания элементов интерфейса, информация, предоставляемая сценарием, должна быть идентифицирована в виде функционального текста, который может быть прочитан с помощью вспомогательной технологии.
- 9. Если веб-страница требует наличия апплета, плагина или другого приложения в клиентской системе для интерпретации содержимого веб-страницы, эта страница должна содержать ссылку на плагин или апплет, который соответствует §1194.21 (a) — (l).
- 10. Если электронные формы предназначены для заполнения в режиме онлайн, форма должна позволять людям, использующим вспомогательные технологии, получать доступ к информации, элементам поля и функциям, необходимым для заполнения и отправки данных формы, включая все указания и подсказки.
- 11. Должен быть предусмотрен метод, который позволяет пользователям пропускать повторяющиеся навигационные ссылки.
- 12. Если требуется ответ по времени, пользователь должен быть предупрежден, и ему должно быть предоставлено достаточно времени, чтобы указать на то, что ему требуется больше времени.

ПОТРЕБНОСТЬ В СКОРОСТИ (БЫСТРОДЕЙСТВИЕ САЙТА)

Несмотря на то что количество пользователей, подключающихся к Интернету по медленным коммутируемым соединениям, сокращается (на момент подготовки книги в США такие каналы использовали 3–5% пользователей), процент людей, использующих для доступа к Интернету не столь и быстрые мобильные сети, резко вырос. В некоторых же областях — таких, как социальные сети и поиск информации — доля использования мобильных устройств уже превысила долю использования настольных компьютеров. Если у вас есть смартфон, то вы знаете, столь тягостно ожидание момента полного отображения веб-страницы на экране при использовании для передачи данных мобильного соединения.

Быстродействие сайта имеет решающее значение независимо от того, каким образом пользователи получают доступ к вашему сайту. Согласно исследованиям, проведенным Google² в 2009-м году, затягивание всего лишь на 100–400 миллисекунд отображения страницы результатов поиска ведет к уменьшению количества поисковых запросов (на 0,2–0,6%). Согласно же исследованиям Amazon.com³, сокращение времени загрузки веб-страницы всего на 100 мс привело к увеличению дохода на 1%. Согласно другим исследованиям, пользователи ожидают, что сайт загрузится

² «Speed Matters», googleresearch.blogspot.com/2009/06/speed-matters.html.

³ Статистика из «Make Data Matter», презентация PowerPoint, подготовленная Грегом Линденом (Greg Linden) из Стэнфордского университета (2006).

менее чем за 2 секунды, и почти третья вашей аудитории уйдет с вашего сайта на другой, если этого не произойдет. Кроме того, эти люди вряд ли вернутся. Учтите, что Google добавил параметр быстродействия сайта в алгоритм поиска, поэтому, если ваш сайт медленный, он вряд ли будет отображаться на этом желанном первом экране результатов поиска. Отсюда следует, что быстродействие сайта (вплоть до миллисекунды!) имеет большое значение.

Чтобы улучшить быстродействие сайта, можно выполнить много разных действий, и все они относятся к двум широким категориям: ограничение размера файлов и уменьшение количества запросов к серверу. В следующем далее списке приведены лишь некоторые аспекты оптимизации сайта, но он дает общее представление о том, что можно сделать в подобном случае.

- Оптимизируйте изображения, чтобы они имели наименьший размер файла без ущерба для качества (методы оптимизации изображений будут рассмотрены в главе 24).
- Оптимизируйте разметку HTML, избегая ненужных уровней вложенных элементов.
- Минимизируйте документы HTML и CSS, удаляя лишние символы пробелов и возвратов строк.
- Сведите к минимуму использование JavaScript.
- Добавляйте сценарии таким образом, чтобы они загружались параллельно с другими ресурсами веб-страницы и не блокировали визуализацию веб-страницы.
- Не загружайте ненужные ресурсы (например, изображения, сценарии или библиотеки JavaScript).
- Уменьшите количество запросов браузера к серверу (известные как *HTTP-запросы*).

Каждая передача данных серверу в виде HTTP-запроса занимает несколько миллисекунд, но эти миллисекунды могут складываться, что выливается в приличные задержки. Все эти маленькие виджеты Твиттера, кнопки Фейсбука **Нравится** и рекламные объявления могут отправлять десятки запросов к каждому серверу. Вы можете быть удивлены, увидев, сколько запросов к серверу делает даже простой сайт.

Если вы хотите убедиться в этом сами, вы можете использовать утилиту Сеть (Network), доступную в меню **Веб-разработка** в Firefox. Утилита Сеть отображает каждый запрос к серверу и количество миллисекунд, который он занял. Вот как ее можно использовать в Firefox (или в другом браузере):

1. Запустите браузер Firefox и перейдите к любой веб-странице.
2. В меню **Инструменты** (Tools) выберите команды **Веб-разработка | Сеть** (Web design | Network) — в окне браузера откроется соответствующая панель.
3. Теперь перезагрузите веб-страницу. На диаграмме (обычно называемой *диаграммой водопада*) показаны все выполненные запросы и загруженные ресурсы. Столбцы справа показывают количество времени, которое занимает каждый запрос (в миллисекундах). В нижней части таблицы показана сводка по количеству выполненных запросов и общему объему переданных данных.

На рис. 3.5 показана часть диаграммы водопада быстродействия для сайта oreilly.com — так можно просматривать в Интернете любой сайт. И это может быть очень познавательно. Я не стану в этой книге подробно останавливаться на вопросах, связанных с быстродействием сайта, но хочу, чтобы вы при разработке веб-дизайна помнили важность сохранения как можно меньших размеров файлов и исключения ненужных запросов к серверу.

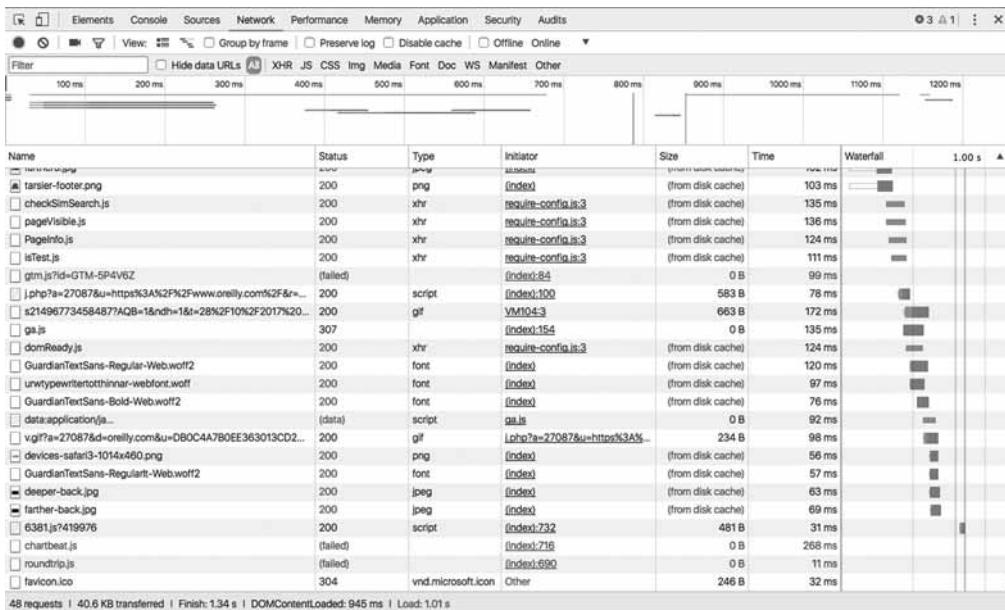


Рис. 3.5. Благодаря диаграммам водопада, созданным утилитой Сеть браузера Firefox, можно увидеть количество запросов сервера со стороны веб-страницы и время, которое занимает каждый запрос

ВСПОМОГАТЕЛЬНЫЕ ИНСТРУМЕНТЫ ПО ОЦЕНКЕ БЫСТРОДЕЙСТВИЯ САЙТА

Попробуйте некоторые из этих инструментов, позволяющих протестировать быстродействие сайта.

- Утилита WebPageTest (webpagetest.org). Этот инструмент изначально был разработан для AOL, но теперь доступен всем для свободного использования на условиях лицензии с открытым исходным кодом. Просто введите интернет-адрес, и WebPageTest вернет диаграмму водопада, снимок экрана и другую статистику.
- Утилита Google PageSpeed Insights (developers.google.com/speed/pagespeed/insights/). Это еще один сервис, который анализирует быстродействие любого сайта, на который вы указываете. Он также генерирует предложения по ускорению загрузки вашей веб-страницы.
- Утилита YSlow от Yahoo! (yslow.org), доступная на бесплатной основе, анализирует сайт в соответствии с 23 правилами быстродействия в Интернете, а затем дает сайту оценку и предложения по улучшению быстродействия.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Существуют и другие технические приемы оценки быстродействия сайта, которые являются слишком специфическими для этой книги (и, честно говоря, для меня), и я полагаю, что если вы читаете эту книгу, то вы, вероятно, не совсем готовы стать гуру в области оценки быстродействия сайта. Но, когда вы будете готовы к выполнению этих обязанностей, обратите внимание на несколько ресурсов, которые должны помочь вам в этом.

- На сайте larahogan.me/design Лара Хоган (Lara Hogan) собрала список исследований, инструментов и ресурсов, связанных с быстродействием сайтов. Вы можете также обратиться к ее книге «*Designing for Performance*» (O'Reilly), которая доступна на этом же сайте.
- В книге «*High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps*», написанной Максимилиано Фиртманом (Maximiliano Firtman) и выпущенной издательством O'Reilly, описываются методы оптимизации и инструменты, предназначенные для проверки прогресса в деле разработки веб-сайтов.
- Сайт [Google Make the Web Faster](http://code.google.com/speed/) (code.google.com/speed/) предоставляет вам отличную возможность узнать об оптимизации сайта. На этом сайте доступны множество отличных учебных пособий и статей, а также инструменты для измерения быстродействия сайта.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Вот несколько вопросов, которые помогут вам проверить, насколько хорошо вы усвоили материал этой главы. Ответы на эти вопросы приведены в *приложении 1*.

1. Приведите как минимум два неизвестных фактора, которые необходимо учитывать при проектировании и разработке сайта.
2. Сопоставьте технологию или практику (слева) с проблемой, которую она лучше всего решает (справа):

1. <input type="checkbox"/> Постепенное улучшение.	a. Вспомогательные средства чтения и ввода.
2. <input type="checkbox"/> Обнаружение на стороне сервера.	b. Низкая скорость подключений.
3. <input type="checkbox"/> Адаптивный дизайн.	c. Все уровни возможностей браузера.
4. <input type="checkbox"/> Документ WAI-ARIA.	d. Определение используемого устройства.
5. <input type="checkbox"/> Оптимизация быстродействия сайта.	e. Множество размеров экрана.
3. Стратегии доступности веб-сайтов учитывают четыре широкие категории проблем со здоровьем. Назовите не менее трех и укажите меры, которые можно предпринять, чтобы обеспечить доступность контента для пользователей, относящихся к этим категориям.
4. Когда используется диаграмма водопада?

ЧАСТЬ II

**РАЗМЕТКА HTML,
ПРИМЕНЯЕМАЯ
ДЛЯ СТРУКТУРИРОВАНИЯ
ВЕБ-СТРАНИЦ**

ГЛАВА 4

СОЗДАНИЕ ПРОСТОЙ ВЕБ-СТРАНИЦЫ

В этой главе...

- ▶ Элементы и атрибуты
- ▶ Разметка несложной веб-страницы
- ▶ Элементы, поддерживающие структуру документа
- ▶ Проблемы при создании веб-страниц

В части I книги был представлен общий обзор сферы веб-дизайна. Теперь, когда вы ознакомлены с основными его концепциями, пришло время создать настоящую веб-страницу. Это будет чрезвычайно простая страница, но даже самые сложные страницы основаны на тех же принципах.

В этой главе мы шаг за шагом создадим веб-страницу, чтобы вы могли почувствовать, что такое разметка документа с помощью HTML-тегов. Выполнение упражнений позволит вам работать вместе с нами.

По мере чтения главы вы сможете:

- получить представление о реализации разметки и функциях, которые выполняются элементами и атрибутами;
- ознакомиться с порядком интерпретации HTML-документов браузерами;
- получить представление о структуре HTML-документов;
- освоить применение таблиц стилей.

Мы пока не станем углубляться в описание конкретных элементов разметки текста или правил для таблиц стилей, поскольку эти вопросы будут подробно рассмотрены в следующих главах. В этой же главе вас ожидает знакомство с процессом веб-дизайна в целом, общей структурой HTML-документа и новой терминологией.

ПОШАГОВОЕ СОЗДАНИЕ ВЕБ-СТРАНИЦЫ

Первое знакомство с веб-страницей произошло у вас в главе 2, а сейчас вы сможете создать подобный документ самостоятельно и поэкспериментировать с ним в браузере. Процесс создания веб-страницы, рассмотренный в этой главе, состоит из пяти

шагов. И каждый из них базируется на принципах, лежащих в основе при создании любой веб-страницы.

- **Шаг 1. Начнем с ввода контента.** Для начала создадим текстовый контент (содержимое) и рассмотрим, как его обрабатывают браузеры.
- **Шаг 2. Добавление структуры в HTML-документ.** На этом этапе познакомимся с синтаксисом HTML-элементов и элементов, применяемых для создания контента и метаданных.
- **Шаг 3. Добавление тегов в HTML-документ.** А теперь опишем контент, используя соответствующие текстовые элементы, и познакомимся с правилами по использованию тегов HTML.
- **Шаг 4. Добавление изображений.** В процессе добавления изображений на веб-страницу познакомимся с соответствующими атрибутами и пустыми элементами.
- **Шаг 5. Изменение внешнего вида текста с помощью таблицы стилей.** На этом шаге мы получим представление о форматировании контента с помощью каскадных таблиц стилей (CSS, Cascading Style Sheets).

В результате выполнения этих шагов будет создана веб-страница, показанная на рис. 4.1. Конечно, выглядит она весьма просто, но — «лиха беда начало».



Рис. 4.1. Эта веб-страница была создана всего лишь за пять шагов

В процессе создания веб-страницы мы будем постоянно проверять внесенные изменения в окне браузера. Необходимость в столь частой проверке обусловлена тем, что мы только начинаем знакомиться с прекрасным миром веб-дизайна. В дальнейшем, когда вы уже приобретете необходимый опыт и навыки, вам не придется столь часто проверять плоды своих усилий.

ЗАПУСК ТЕКСТОВОГО РЕДАКТОРА

Как в этой главе, так и во всех последующих главах, мы будем создавать HTML-документы вручную, поэтому для начала нужно научиться запускать текстовый редактор. Вполне подойдет текстовый редактор, который входит в комплект поставки

вашей операционной системы, — например, Блокнот (Notepad) в Windows илиTextEdit в Macintosh. Можно воспользоваться и другими текстовыми редакторами, которые позволяют создавать и сохранять простые текстовые файлы с расширением **html**. Даже если у вас имеется визуальный инструмент, предназначенный для создания веб-страниц, такой, как Dreamweaver, не спешите прибегать к его использованию. Желательно начать с знакомства с разметкой HTML-документа, выполняемой вручную (см. врезку «Нелегкий путь освоения HTML»).

НЕЛЕГКИЙ ПУТЬ ОСВОЕНИЯ HTML

Я разработала собственный метод обучения HTML, который заключается в выполнении HTML-разметки вручную. На первый взгляд, это может показаться старомодным, но зато дает превосходные результаты. Нет лучшего способа понять, как устроена и работает разметка, чем добавлять в документ по одному тегу, открывая затем веб-страницу в браузере. Это не займет много времени, но позволит близко познакомиться с процессом разметки HTML-документов.

Конечно, можно начать с использования визуального инструмента разработки веб-страниц или инструмента, поддерживающего возможности перетаскивания, но в этом случае вы не получите представления о сущности процесса веб-дизайна. Эти инструменты лучше применять потом, когда вы овладеете основами веб-дизайна. Также, благодаря просмотру содержимого исходного файла, можно увидеть, что именно отображает веб-страница. Это важно для устранения возможных проблем с некорректно отображаемыми веб-страницами или при уточнениях настроек форматирования, заданного по умолчанию, которое предлагается веб-инструментами.

Следует отметить, что даже профессиональные веб-дизайнеры выполняют разметку контента вручную, что позволяет контролировать создание кода и приниматьзвешенные решения об использовании различных элементов.

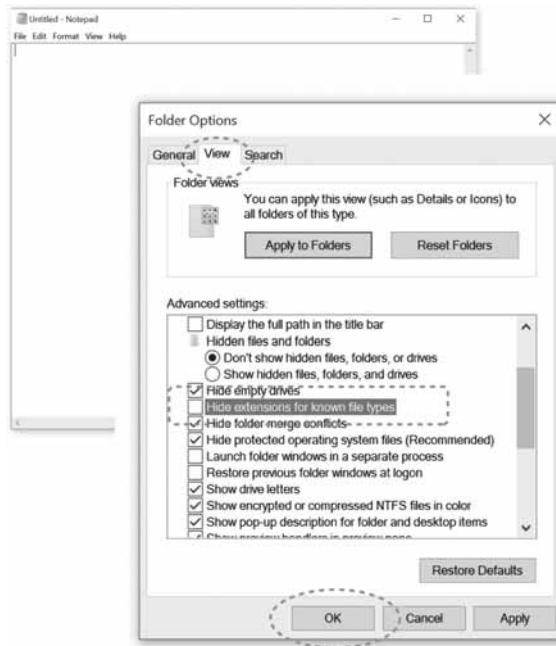
В следующем разделе мы узнаем, каким образом можно открывать новые документы в Блокноте (Notepad) и TextEdit. Даже если вы ранее использовали эти приложения, обратите внимание на специальные настройки, которые позволяют быстрее и качественнее выполнить предстоящую работу. И начнем мы с Блокнота, входящего в комплект поставки любой версии Windows (информация для пользователей Mac будет приведена в следующем разделе).

Создание нового документа в Блокноте (Windows)

Чтобы создать новый документ с помощью приложения Блокнот, входящего в комплект поставки Windows 10, выполните следующие действия (рис. 4.2):

1. Чтобы получить быстрый доступ к программе Блокнот, в строке поиска введите Блокнот. Щелкните на значке программы Блокнот, чтобы открыть окно нового документа, в котором можно начинать вводить данные. ①
2. Затем отобразите расширения имен файлов. Это действие не является обязательным, но лучше его выполнить, чтобы лучше ориентироваться в создаваемых файлах. Откройте окно Проводника (File Explorer), выберите вкладку Вид (View) и щелкните справа на кнопке Параметры (Options). На панели Параметры папок (Folder Options) снова выберите вкладку Вид (View). ②

3. Найдите флажок **Скрывать расширения для зарегистрированных типов файлов** (Hide extensions for known file types) и отмените его установку. ❸
4. Щелкните на кнопке **OK**, чтобы сохранить настройки. После этого расширения имен файлов начнут отображаться. ❹



❶ Щелкните на значке **Блокнот**, чтобы открыть новый документ.

❷ Откройте Проводник Windows, выберите вкладку **Вид**, а потом щелкните на находящейся справа кнопке **Параметры** (не показана). Выберите вкладку **Вид**.

❸ Отмените установку флажка **Скрывать расширения для зарегистрированных типов файлов**.

❹ Щелкните на кнопке **OK**, чтобы сохранить настройку, после чего отобразятся расширения имен файлов.

Рис. 4.2. Создание нового документа в Блокноте

Создание нового документа вTextEdit (macOS)

По умолчанию редактор TextEdit открывается в режиме форматированного текста (RTF), в котором создаются документы, которые уже включают необходимое форматирование: выделение текста полужирным шрифтом, настройки размера шрифта и т. п. При этом в верхней части окна отображается панель инструментов форматирования. В режиме работы с обычным текстом (plain-text) эта панель не отображается. Поскольку документы HTML представляют собой простой текст, следует перейти в режим работы с обычным текстом (рис. 4.3).

1. Чтобы открыть редактор TextEdit, запустите Finder и найдите папку **Программы** (Applications), в которой находится значок редактора TextEdit. После чего щелкните на его имени или значке, чтобы запустить его на выполнение.
2. В начальном диалоговом окне TextEdit щелкните на находящейся в нижнем левом углу кнопке **Новый документ** (New Document). Если в верхней части документа без названия (Untitled) отображается меню форматирования текста и линейка вкладок, это будет означать, что вы находитесь в режиме форматирования текста (rich-text mode) ❶. Если же эти элементы не отображаются, это первый признак

выбора режима работы с простым текстом (plain-text mode) ❷. Независимо от выбранного режима следует произвести некоторые настройки.

3. Закройте документ и откройте с помощью меню редактораTextEdit диалоговое окно **Настройки** (Preferences).
4. Измените следующие настройки:
 - на вкладке **Новый документ** (New Document) установите флажок **Простой текст** (Plain text) ❸. В разделе **Свойства** (Options) отмените установку всех флажков, задающих автоматическое форматирование ❹;
 - 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) ❺ и отмените установку флажка **Добавлять расширение .txt к именам файлов простого текста** (Add '.txt' extensions to plain text files) ❻. Значения остальных параметров, заданных по умолчанию, можно не изменять.
5. После завершения указанных действий щелкните на красной кнопке в верхнем левом углу диалогового окна настроек.
6. Создайте новый документ с помощью меню **Файл | Создать** (File | New). В нашем случае меню форматирования отсутствует, и сохранять текст можно в виде HTML-документа.

Если жеTextEdit не используется для создания HTML-документа, созданный документ можно преобразовать обратно в форматированный текст. Для этого воспользуйтесь командами **Формат | Конвертировать в форматированный текст** (Format | Make Rich Text).

❶ Наличие меню форматирования свидетельствует о выборе режима форматированного текста.

❷ В случае выбора режима простого текста меню форматирования отсутствует.



Рис. 4.3. Запуск редактораTextEdit и установка флажка **Простой текст** (Plain text) на вкладке **Настройки** (Preferences)

ШАГ 1. НАЧНЕМ С ВВОДА КОНТЕНТА

Завершив создание нового документа, можно приступать к вводу контента. Упражнение 4.1 проведет вас через ввод необработанного текстового содержимого будущей веб-страницы и сохранение документа в новой папке.

УПРАЖНЕНИЕ 4.1. ВВОД КОНТЕНТА

1. В новый документ, созданный в текстовом редакторе, введите приведенное далее содержимое начальной страницы. Вставьте содержимое именно в том виде, в котором оно отображается в нашем примере, сохраняя имеющиеся расстояния между строками. Исходный текст этого упражнения также доступен в Интернете по следующему адресу: learningwebdesign.com/5e/materials/.

Black Goose Bistro

The Restaurant

The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients.

Catering

You have fun. We'll handle the cooking. Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.

Location and Hours

Seekonk, Massachusetts;

Monday through Thursday 11am to 9pm; Friday and Saturday, 11am to midnight

2. В меню **Файл** (File) выберите команды **Сохранить** (Save) или **Сохранить как** (Save as), после чего откроется диалоговое окно **Сохранить как** (Save As), показанное на рис. 4.4. Сначала нужно создать новую папку. Для выполнения этой операции следует щелкнуть на кнопке **Новая папка** (New Folder), как при работе в Windows, так и в Mac. В созданной вами папке будут находиться все файлы для сайта. Техническое имя папки, где находятся все эти данные, — локальный корневой каталог.

Назовите новую папку *bistro* и сохраните в ней текстовый файл под именем **index.html**. Имя файла должно заканчиваться на **.html**, чтобы браузер распознал его как веб-документ (см. врезку «Соглашения об именах», содержащую советы по именованию файлов).

3. Давайте просто из любопытства отобразим файл **index.html** в окне браузера:

- пользователям *Windows*: щелкните двойным щелчком на имени файла в Проводнике (File Explorer) для запуска браузера, заданного по умолчанию, или щелкните на имени файла правой кнопкой мыши, чтобы открыть его в выбранном вами браузере;

- пользователям Mac: запустите ваш браузер (я использую Google Chrome) и выберите команды **Открыть** (Open) или **Открыть файл** (Open File) в меню **Файл** (File). Найдите файл **index.html** и выберите его для открытия в браузере.
4. В окне браузера должно отобразиться нечто, похожее на показанное на рис. 4.5. В следующем разделе мы обсудим полученный здесь результат.

Windows 10



MacOS 10

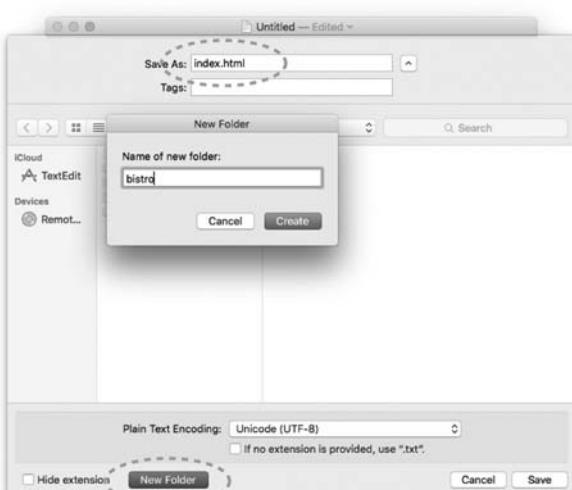
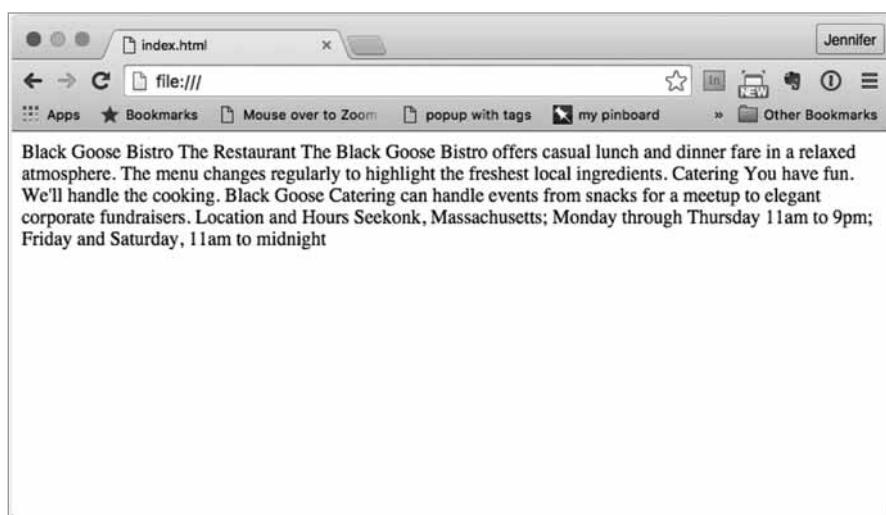
Рис. 4.4. Сохраните файл index.html в новой папке с названием **bistro**

Рис. 4.5. Первый просмотр созданного контента в окне браузера

СОГЛАШЕНИЯ ОБ ИМЕНАХ

При именовании файлов важно соблюсти определенные соглашения:

- **используйте корректные расширения имен файлов** — названия HTML-файлов должны заканчиваться на `.html` или `.htm`. Названия файлов веб-графики должны иметь расширения `gif`, `png`, `jpg` (также применяется `jpeg` или `svg`, но реже);
- **никогда не используйте пробелы в именах файлов** — для визуального разделения слов в именах файлов обычно применяется символ подчеркивания или дефис, например: `robbins_bio.html` или `robbinsbio.html`;
- **не применяйте специальные символы, такие, как: ?, %, #, /, :, ;, •, и т. п.** — включайте в имена файлов буквы, цифры, подчеркивания, дефисы и точки, но ограничьтесь только этими символами. Лучше не прибегать к международным символам — таким, например, как символ шведской буквы å;
- **имена файлов чувствительны к регистру символов** — это зависит от конфигурации сервера. При создании имен файлов используйте лишь строчные буквы. Это необязательно, хотя в значительной степени облегчит работу с именами файлов;
- **при именовании файлов соблюдайте краткость** — чем длиннее имя, тем больше вероятность ошибок, к тому же короткие имена уменьшают размер файла еще на несколько байтов. Если нужно присвоить файлу длинное имя, состоящее из нескольких слов, разделите слова дефисами — например: `long-document-title.html`. Это значительно улучшит его читаемость;
- **используйте мнемонические правила** — при разработке крупных сайтов удобно разработать согласованную схему именования файлов — например, всегда использовать строчные буквы с дефисами между словами. Тогда не придется мучительно долго вспоминать имя файла позднее.

Анализ результатов шага 1

Как видите, сейчас ваша веб-страница выглядит не слишком привлекательно (см. рис. 4.5). Контент отображается в виде одного большого блока — но вы, наверняка помните, что исходный документ выглядел иначе. А теперь подытожим увиденное. Во-первых, и это очевидно, браузер игнорирует разрывы строк в исходном документе. Обратитесь ко врезке «*Что игнорируют браузеры...*», где приведены символы исходного документа, игнорируемые браузерами.

Во-вторых, недостаточно просто ввести некоторый контент и присвоить файлу документа имя с расширением `html`. Хотя браузер и отображает текст, находящийся в файле, но этот текст будет неструктурированным. Здесь и пригодятся возможности HTML. Для добавления структуры в документ следует применить разметку: сначала к документу HTML (Шаг 2), а затем — к содержимому страницы (Шаг 3). Как только браузер получит представление о структуре контента, веб-страница будет отображаться значительно красивее.

ЧТО ИГНОРИРУЮТ БРАУЗЕРЫ...

Обратите внимание, что следующие символы будут проигнорированы браузером:

- **символы множественных пробелов (служебные пробелы)** — когда браузер встречается с несколькими пробелами, будет отображен лишь один пробел. Поэтому, если документ содержит фразу:
`long, long ago`
браузер отобразит:
`long, long ago`
- **разрывы строк (символы возврата каретки)** — браузеры преобразовывают символы возврата каретки в служебные пробелы, а из-за «игнорирования множественных пробелов» разрывы между строками браузером корректно не отображаются;
- **символы табуляции** — символы табуляции также преобразуются в символьные пробелы, угадайте, в какие именно? Они не нужны для выделения текста на веб-странице, хотя и делают код удобным для просмотра;
- **нераспознанная разметка** — браузеры игнорируют нераспознанные теги. В зависимости от элемента и возможностей браузера это приводит к разным результатам. Браузер может ничего не отобразить или отобразить содержимое тега так, как если бы речь шла об обычном тексте;
- **текст комментариев** — браузеры не отображают текст, расположенный между специальными тегами: `<!-- -->`, которые используются для обозначения комментария. Обратитесь ко врезке «Добавление скрытых комментариев» за более подробными сведениями по этому вопросу.

ШАГ 2. ДОБАВЛЕНИЕ СТРУКТУРЫ В HTML-ДОКУМЕНТ

После сохранения контента, добавленного в HTML-документ, можно приступить к его разметке.

Анатомия HTML-элемента

В главе 2 были рассмотрены примеры элементов с открывающим тегом (например, `<p>` для абзаца) и закрывающим тегом (`</p>`). Прежде чем добавлять теги в документ, рассмотрим, как устроен элемент HTML (его синтаксис) и уточним некоторые термины. Общая структура элемента показана на рис. 4.6.



Рис. 4.6. Составные части элемента HTML

Элементы в исходном тексте идентифицируются с помощью тегов. *Тег* состоит из имени элемента, которое представляет собой аббревиатуру от длинного описательного названия, заключенного в угловые скобки: < >. Браузеру известно, что текст в таких скобках скрыт, т. е. не отображается в окне браузера.

Имя элемента заключено между *открывающим тегом* (еще называется *начальным тегом*) и *закрывающим* (или *конечным*) тегом, которому предшествует косая черта (слэш): /. Закрывающий тег для элемента играет роль выключателя. Будьте осмотрительны и не применяйте в конечных тегах символ обратной косой черты (обратного слэша): \ (см. врезку «Слэш или обратный слэш?»).

Разметкой называются теги, находящиеся вокруг контента. Важно отметить, что в область действия элемента входит как сам контент, так и его разметка (начальный и конечный теги). Но далеко не все элементы включают контент. Некоторые из них являются пустыми по определению — например, элемент img, который служит для добавления на страницу изображения. О пустых элементах речь пойдет в этой главе позже.

ИСПОЛЬЗУЕМ СИМВОЛЫ НИЖНЕГО РЕГИСТРА

Имеется и версия HTML под названием XHTML, при использовании которой все имена элементов и атрибутов должны отображаться в нижнем регистре. После выхода HTML5 версия XHTML устарела, за исключением определенных случаев, когда ее применяют с другими языками XML, причем сохраняется предпочтение для нижнего регистра при написании имен элементов.

И последнее: регистр символов. В HTML не имеет значения регистр символов, используемых в именах элементов (то есть имена не чувствительны к регистру символов). Так что названия , или воспринимаются браузером одинаково. Большинство разработчиков указывают имена элементов с применением символов нижнего регистра — этого неписаного правила мы будем придерживаться и в этой книге.

СЛЭШ ИЛИ ОБРАТНЫЙ СЛЭШ?

В HTML-тегах и интернет-адресах (URL) используется слэш — символ косой черты: /. Этот символ на QWERTY-клавиатуре находится под вопросительным знаком: ? (учтите, что расположение клавиш на клавиатурах может различаться).

Слэш (символ косой черты) легко перепутать с обратным слэшем (символом обратной косой черты): \, который находится под символом канала: | (рис. 4.7). Символ обратной косой черты \ не будет воспринят в тегах и интернет-адресах, поэтому не путайте эти символы — они не взаимозаменяемы!

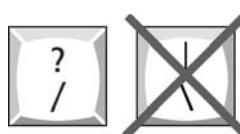


Рис. 4.7. Клавиши с обозначением слэша (косой черты): / (слева) и обратного слэша (обратной косой черты) \ (справа)

Базовая структура документа

На рис. ЦВ-4.8 показана рекомендуемая минимальная структура HTML-документа. Здесь идет речь о «рекомендации», поскольку единственным *обязательным* HTML-элементом является элемент `title`. Однако будет лучше, особенно для начинающих разработчиков, структурировать документ как в области метаданных (заголовка), так и контента (основной части). А теперь внимательно рассмотрим следующий пример, содержащий минимальную разметку:

- ❶ Обратите внимание, что первая строка примера не является элементом. Тут находится *объявление типа документа* (также именуемое *объявлением DOCTYPE*), позволяющее современным браузерам при интерпретации документа уточнять конкретную спецификацию HTML. Объявление DOCTYPE идентифицирует, что документ создан на HTML5.
- ❷ Документ полностью содержится в области действия элемента `html`. Элемент `html` называется *корневым* — он включает все элементы в документе и не может содержаться в каком-либо ином элементе.
- ❸ Внутри элемента `html` документ делится на элементы `head` и `body`. Элемент `head` включает относящиеся к документу элементы, которые не отображаются как часть контента, — например: заголовок, таблицы стилей, сценарии и метаданные.
- ❹ Элементы `meta` поддерживают *метаданные*, т. е. информацию о документе. В нашем случае здесь указано, что применяемая в документе *кодировка символов* (стандартизированный набор букв, цифр и символов) относится к Unicode (Юникод) версии UTF-8 (см. врезку «*Введение в Юникод*»). Сейчас нам пока нет необходимости подробно останавливаться на этом вопросе, но имеется ряд веских причин для указания кодировки в каждом документе, поэтому этот элемент включен в минимальную разметку документа. Другими типами метаданных, предоставляемых элементом `meta`, являются автор документа, перечень ключевых слов, статус публикации, а также описание документа, ориентированное на поисковые системы.
- ❺ Элемент `head` также содержит обязательный элемент `title`. Согласно HTML-спецификации, каждый документ должен иметь описательный заголовок.
- ❻ Наконец, элемент `body` содержит контент, который будет отображаться в окне браузера.

Вы готовы приступить к разметке начальной страницы «Black Goose Bistro»? Откройте в текстовом редакторе документ `index.html` и перейдите к выполнению *упражнения 4.2*.

ВВЕДЕНИЕ В ЮНИКОД

Все символы языка сохраняются в компьютерах в виде цифр. Стандартизованный набор символов с числами, соответствующими элементам кода, называется *набором кодированных символов*, способ же преобразования символов в байты для использования компьютерами называется *кодировкой символов*. На начальных этапах развития вычислительной

▶ техники в компьютерах задействовались ограниченные наборы символов — такие, как ASCII, которые включали всего 128 символов (буквы латинского языка, цифры и распространенные символы). В Интернете сначала применялась кодировка символов Latin-1 (ISO 8859-1), включающая 256 латинских символов, позаимствованных из языков западных регионов. Но, учитывая «всемирный» характер Интернета, оказалось, что этого недостаточно.

Набор символов *Unicode* (Юникод), также называемый *универсальным набором символов*, — это суперсимвольный набор, включающий более 136 тыс. символов (букв, цифр, символов, идеограмм, логограмм и т. п.), позаимствованных из всех современных языков. Дополнительную информацию об этом наборе можно получить на сайте: unicode.org. Набор символов Юникод включает три стандартных кодировки: UTF-8, UTF-16 и UTF-32, которые отличаются количеством байтов, используемых для представления символов (соответственно, 1, 2 или 3 байта).

В версии HTML5 по умолчанию применяется кодировка UTF-8, которая позволяет в одном документе использовать языки, имеющие весьма широкие наборы символов. Полезно объявлять кодировку символов для документа с помощью элемента `meta`, как показано в примере на рис. 4.8. Используемый вами сервер также должен в заголовке *HTTP* выполнять идентификацию HTML-документов, относя их к UTF-8 (в информации о документе, которую сервер направляет агенту пользователя). Можно попросить администратора вашего сервера о подтверждении кодировки HTML-документов.

УПРАЖНЕНИЕ 4.2. ДОБАВЛЕНИЕ МИНИМАЛЬНОЙ СТРУКТУРЫ В HTML-ДОКУМЕНТ

1. Откройте документ `index.html`, если он еще не открыт, и добавьте объявление DOCTYPE:
`<!DOCTYPE html>`
2. Поместите документ полностью в корневой HTML-элемент, добавляя после объявления DOCTYPE начальный тег `<html>` и конечный тег `</html>` в самом конце текста.
3. Затем создайте заголовок документа, куда войдет заглавие страницы. Перед контентом вставьте теги: `<head>` и `</head>`. В элемент `head` внесите информацию о кодировке символов `<meta charset="utf-8">`, а также — заголовок: `Black Goose Bistro`, заключенный в открывающие и закрывающие теги `<title>`.
4. И, наконец, выделите основную часть документа, заключая текстовый контент в теги `<body>` и `</body>`. По завершении этих действий исходный документ примет следующий вид (для удобства здесь и далее код, на который следует обратить внимание, выделен полужирным шрифтом):

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Black Goose Bistro</title>
</head>
```

```
<body>
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and dinner fare in
a relaxed atmosphere. The menu changes regularly to highlight
the freshest local ingredients.

Catering
You have fun. We'll handle the cooking. Black Goose
Catering can handle events from snacks for a meetup
to elegant corporate fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm; Friday and
Saturday, 11am to midnight
</body>
</html>
```

5. Сохраните документ в каталоге **bistro**, в результате будет переписана предыдущая версия документа. Откройте файл в браузере или щелкните на кнопке **Обновить** (Refresh) или **Перезагрузить** (Reload), если файл был открыт ранее. На рис. 4.9 показан полученный результат.

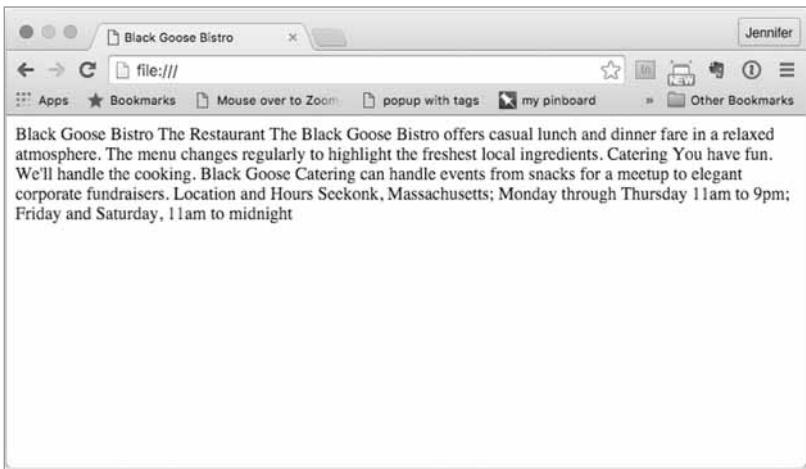


Рис. 4.9. Страница, отображаемая в браузере, после внесения элементов, относящихся к структуре документа

Как видите, после настройки HTML-документа страница **bistro** практически не изменилась, разве что браузер уже отображает заголовок документа на верхней панели (вкладке). Если добавить теперь эту страницу в закладки, заголовок также будет внесен в список закладок (Bookmarks) или в список избранного (Favorites) — как отмечено во врезке «Заголовок должен быть уникальным». Однако контент по-прежнему слит воедино, поскольку браузер не получил никаких указаний о его структуре. Добавление структуры в контент будет рассмотрено в следующем разделе.

ЗАГОЛОВОК ДОЛЖЕН БЫТЬ УНИКАЛЬНЫМ

Элемент `title` не только обязателен для каждого документа, но он также весьма полезен. Заголовок отображается в списке закладок (Bookmarks) или в списке избранного (Favorites) пользователя и на вкладках в браузерах. Описательные заголовки существенно улучшают доступность контента — это первое, что пользователь услышит при использовании программы чтения с экрана (вспомогательное средство, которое читает вслух содержимое страницы для пользователей с ослабленным зрением). Для корректной работы поисковых систем также важны названия документов.

По этим причинам следует научиться создавать содержательные и, вместе с тем, описательные заголовки для документов, избегая заголовков с неясным смыслом — например, таких, как «Добро пожаловать» или «Моя страница». Желательно контролировать длину заголовков, чтобы они целиком отображались в области заголовка браузера. Поскольку пользователи, как правило, держат открытыми несколько вкладок или большой перечень закладок, поэтому первые 20 символов заголовка и должны содержать уникальную информацию о вашем документе.

ШАГ 3. ДОБАВЛЕНИЕ ТЕГОВ В HTML-ДОКУМЕНТ

При наличии даже небольшого опыта разметки документов внесение в контент разметки для заголовков и подзаголовков: `h1` и `h2`, абзацев: `p` и выделенного текста: `em` не должно вызывать затруднений, и этим мы займемся в *упражнении 4.3*. Однако сначала разберемся, какие цели преследует разметка контента с помощью HTML.

Семантика как цель разметки

НАЗНАЧЕНИЕ HTML

Назначение HTML заключается в выделении смыслового содержания и структуры в контенте.

Применение HTML служит для выделения сути и структуры контента. Вопросы внешнего вида контента (его презентация) при этом не являются приоритетными.

При разметке контента следует выбрать HTML-элементы, которые служат для содержательного описания контента. Разметка, выполняемая с помощью таких элементов, называется *семантической*. Например, самый важный заголовок в начале документа помечается элементом `h1`, что свидетельствует о его значительности на веб-странице. Не беспокойтесь при этом, как будет выглядеть страница, поскольку дальнейшие изменения можно легко вносить с помощью таблицы стилей. Важно, чтобы при выборе элементов вы исходили из смысла самого контента.

Помимо добавления содержательных моментов разметка вносит в документ структуру. Порядок следования элементов или их вложения один в другой формируют между ними определенную систему отношений. Можно представлять структуру документа как своего рода скелет, который называется *объектной моделью документа* (Document Object Model, DOM). Базовая иерархия документов служит браузерам подсказкой при обработке контента. Также она является основой, куда при помощи JavaScript вносятся инструкции для презентаций, содержащие таблицы стилей и поведения.

Несмотря на то что с момента своего создания HTML предназначался исключительно для смысловой и структурной поддержки документов, эта функция HTML значительно расширилась в первые же годы существования Интернета. Тогда отсутствовала система таблиц стилей, поэтому возможности HTML были существенно расширены, что позволило авторам с помощью разметки изменять внешний вид шрифтов, цвет текста, а также способы его выравнивания. Эти дополнительные презентационные возможности так и остались в HTML. Вы можете их заметить, просматривая исходные тексты веб-страниц или же сайтов, созданных с помощью старых инструментов. Однако в книге речь идет о корректной методике применения HTML в соответствии с принятым сейчас, основанном на стандартах, семантическим подходом к веб-дизайну.

Итак, вернемся к нашей практике и приступим к работе с контентом, как показано в *упражнении 4.3*.

УПРАЖНЕНИЕ 4.3. ЗНАКОМСТВО С ЭЛЕМЕНТАМИ ПО РАБОТЕ С ТЕКСТОМ

1. Откройте документ `index.html` в текстовом редакторе, если он еще не открыт.

2. Первая строка текста: «Black Goose Bistro» — служит главным заголовком страницы. Для разметки этой строки применяется элемент заголовка 1-го уровня `h1`. В начало строки поместите открывающий тег `<h1>`, в конце строки укажите закрывающий тег `</h1>`:

```
<h1>Black Goose Bistro</h1>
```

3. Страница имеет также три подзаголовка. Аналогичным образом разметьте их с помощью элементов заголовка 2-го уровня `h2`. Далее показано, как это сделано на примере одного из них: «The Restaurant». Выполните аналогичные действия для заголовков «Catering» (Питание) и «Location and Hours» (Расположение и часы работы).

```
<h2>The Restaurant</h2>
```

4. После каждого из элементов `h2` находится небольшой текстовый абзац — разметьте его как элемент абзаца `p`. Далее показана разметка одного из них — остальную разметку выполните самостоятельно:

```
<p>The Black Goose Bistro offers casual lunch and dinner  
fare in a relaxed atmosphere. The menu changes regularly to  
highlight the freshest local ingredients.</p>
```

5. И наконец, в разделе «Catering» следует подчеркнуть то обстоятельство, что посетители могут возложить все заботы по приготовлению угощения, в том числе и на выезде, на нас. Чтобы выделить соответствующий текст, разметьте его с помощью элемента `em`, поддерживающего выделение:

```
<p>You have fun. <em> We'll handle the cooking.  
</em> Black Goose Catering can handle events from snacks for  
a meetup to elegant corporate fundraisers.</p>
```

6. После завершения разметки документа сохраните его, как мы делали раньше, и откройте (или перезагрузите) страницу в браузере — вы увидите веб-страницу, похожую на показанную на рис. 4.10. Если это не так, проверьте свою разметку, убедитесь, что не пропущены угловые скобки и символы косой черты (слэша) в закрывающих тегах.

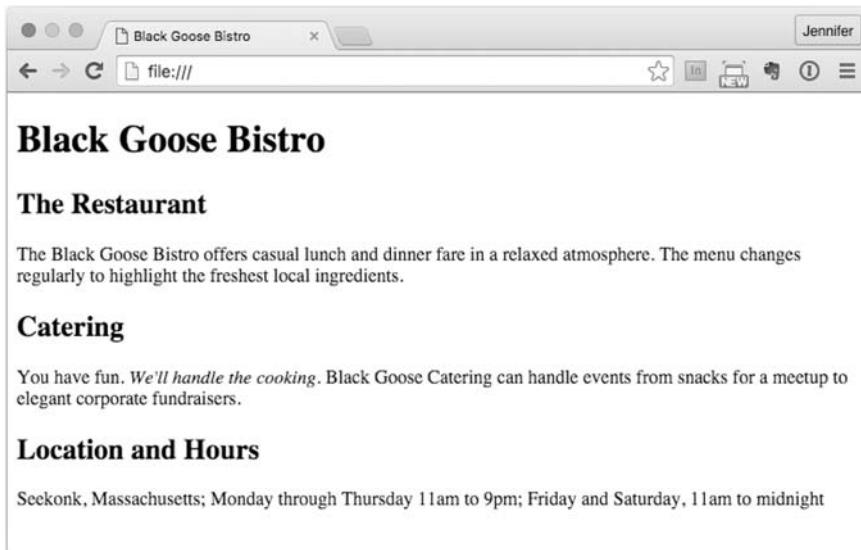


Рис. 4.10. Начальная страница после разметки с помощью HTML-элементов.

Теперь можно говорить о достижении определенных успехов. При условии верной идентификации элементов браузер теперь отображает более внятный текст. Подчеркнем далее некоторые важные моменты на изображении, показанном на рис. 4.10.

Блоchные и стрoчные элемеnты

Отметим очевидный факт: как элементы заголовка, так и элементы абзаца начинаются с новый строки и не слиты все вместе, как это было до сих пор. По умолчанию заголовки и абзацы отображаются как *блочные элементы*. Браузеры обрабатывают блочные элементы, как будто они размещены на странице в небольших прямоугольных ячейках. Каждый блочный элемент начинается с новой строки, и, по умолчанию, обычно над и под ним добавляется некоторое пространство. На рис. ЦВ-4.11 края блочных элементов выделены красной контурной линией.

Для сравнения обратите внимание на текст, размеченный с помощью элемента `em` (он на рис. ЦВ-4.11 выделен синим цветом). Этот текст не начинает новую строку, а остается в пределах абзаца. Причина в том, что элемент `em` является *строчным элементом* (также называется *семантическим элементом текстового уровня* или *элементом фразы*). Строчные элементы не начинают новых строк — они просто «плывут по течению».

ДОБАВЛЕНИЕ СКРЫТЫХ КОММЕНТАРИЕВ

В исходном документе можно делать заметки, размечая их как комментарии. Все, что находится между тегами комментариев: `<!-- -->`, не отображается в браузере и не влияет на оставшуюся часть текста:

```
<!-- Комментарий -->
<!-- Многострочный комментарий,
    который заканчивается здесь. -->
```

Комментарии полезно применять при маркировке или в процессе структурирования больших документов, особенно при работе над ними команды разработчиков. В следующем примере комментарии применяются для указания раздела исходного текста, который описывает навигацию:

```
<!-- начало глобальной навигации -->
<ul>
...
</ul>
<!-- завершение глобальной навигации -->
```

Учтите, что, хотя браузер и не отображает комментарии на веб-странице, читатели могут ознакомиться с ними при «просмотре исходного текста», поэтому оставленные вами комментарии должны быть корректными.

Стили, заданные по умолчанию

Просмотр размеченной веб-страницы (см. рис. 4.10 и ЦВ-4.11) показывает, что браузер можно использовать при размещении на странице некоторой визуальной иерархии, когда заголовок первого уровня представляется на странице самым большим и выделенным шрифтом, а заголовки второго уровня уже выделены более мелким шрифтом и так далее.

Каким образом браузер узнает о том, какой вид должен иметь элемент `h1`? Здесь применяется таблица стилей! Браузеры располагают собственными встроенными таблицами стилей (в спецификации их называют *таблицами стилей пользователяского агента*), где описано заданное по умолчанию отображение элементов. Отображение, заданное по умолчанию, свойственно для всех браузеров (например, все элементы `h1` больше и сильнее выделены), но есть некоторые отличия (элемент `blockquote` для длинных цитат может как иметь отступ от текста, так и не иметь его).

Если вы полагаете, что элемент `h1` слишком велик и неуклюж при отображении с помощью браузера, измените его вид при помощи собственного правила таблицы стилей. Но не следует для улучшения вида помечать главный заголовок другим элементом — например, применять элемент `h3` вместо элемента `h1`, чтобы уменьшить размеры заголовка. Прибегать к подобным ухищрениям иногда приходилось, когда еще не существовало повсеместной поддержки таблиц стилей. Так что всегда следует выбирать элементы, исходя из их функций при описании контента, и меньше беспокоиться о заданном по умолчанию отображении элементов браузером.

Далее мы рассмотрим презентацию страницы с помощью таблиц стилей, но сначала добавим изображение.

ШАГ 4. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ

А теперь подумайте о том, может ли представлять интерес веб-страница без изображений? В *упражнении 4.4* для добавления изображения на страницу мы воспользуемся элементом `img`. Более подробно добавление изображений на веб-страницу будет рассмотрено в *главе 7*, ну а пока давайте познакомимся с двумя основными концепциями разметки: применением пустых элементов и атрибутов.

Пустые элементы

До этого момента почти все элементы, примененные для разметки начальной веб-страницы «Black Goose Bistro», следовали синтаксису, представленному на рис. 4.6: текстовый контент заключался в начальный и конечный теги.

Однако часть элементов не включает контент, поскольку применяются как носители указаний. Эти элементы называются *пустыми*. примером пустого элемента служит элемент изображения `img`. Этот элемент указывает браузеру, что необходимо получить с сервера файл изображения и вставить его на указанное место в тексте. Иные пустые элементы указывают на разрыв строки: `br`, тематические разрывы: `hr` — что расшифровывается как «горизонтальные линейки», а также — на элементы, представляющие информацию о документе, но не влияющие на отображение контента, — например: `meta` (этот элемент уже рассматривался в этой главе).

На рис. 4.12 показан совсем несложный синтаксис пустого элемента (сравните с рис. 4.6).

<название элемента>

Пример: элемент `br` вставляет разрыв строки.

`<p>1005 Gravenstein Highway North
Sebastopol, CA 95472</p>`

Рис. 4.12. Пустой элемент структуры

Атрибуты

АТРИБУТЫ

Атрибуты — это инструкции, которые уточняют или модифицируют элемент.

какое изображение использовать. И тут вступают в дело атрибуты. *Атрибуты* — это инструкции, уточняющие элемент или изменяющие его. Для элемента `img` необходим атрибут `src` (сокращение от `source`, источник), который и указывает местоположение (а именно, URL) файла изображения.

Рассмотрим синтаксис для атрибута:

`название_атрибута="значение"`

А теперь рассмотрим добавление на веб-страницу изображения с помощью пустого элемента `img`. Очевидно, тег `` не слишком информативен — не указывает,

Атрибуты указываются после имени элемента и отделяются от него пробелом. Но при использовании атрибутов в непустых элементах они заключаются лишь в открывающие теги:

```
<элемент название_атрибута="значение">
<элемент название_атрибута="значение">Контент</элемент>
```

В одном элементе можно также размещать несколько атрибутов — не забывайте только разделять их пробелами:

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

На рис. 4.13 показан элемент `img`, включающий необходимые атрибуты.

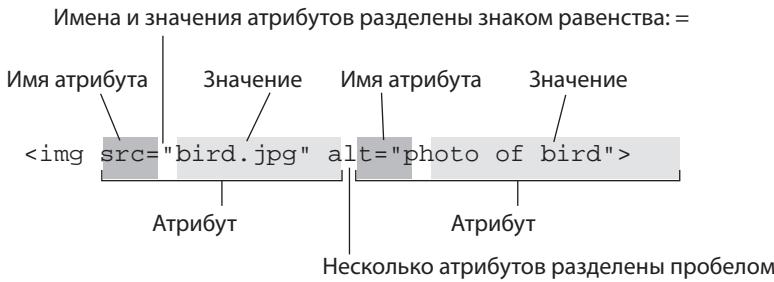


Рис. 4.13. Элемент `img` с двумя атрибутами

ЗАЧЕМ НЕОБХОДИМА ДОПОЛНИТЕЛЬНАЯ КОСАЯ ЧЕРТА?

Если вы внимательно просмотрите исходные коды реальных веб-страниц, наверняка обратите внимание на пустые элементы с дополнительными слэшами в конце — например: ``, `
`, `<meta />` и `<hr />`. Это означает, что документ составлен с учетом более строгих правил XHTML. При обращении к XHTML все элементы, включая и пустые, должны закрываться (или завершаться, если применять корректный термин). Пустые элементы завершаются путем добавления перед закрывающей скобкой косой черты. Пробел перед этим символом необязателен, но он добавляется для обратной совместимости с браузерами, которые не имеют синтаксических анализаторов XHTML, поэтому записи в виде как ``, так и `
` и т. п. вполне допустимы.

Подытожим, что нужно знать об атрибутах:

- атрибуты располагаются после имени элемента и только в открывающем теге, но никак не в закрывающем;
- к элементу можно применять несколько атрибутов, разделенных пробелами в открывающем теге. Порядок их расположения не столь важен;
- большинство атрибутов принимают те значения, которые указаны после знака равенства: =. В HTML некоторые значения атрибутов описываются одиночными

словами. Например, атрибут `checked`, который устанавливает флагок формы при загрузке формы, эквивалентен указанию `check="checked"`. Иногда говорят, что такой тип атрибута называется *логическим атрибутом*, поскольку описывает функцию, которая имеет два состояния (включение и отключение);

- значением атрибута может быть число, слово, текстовая строка, интернет-адрес (URL) или размер — в зависимости от назначения атрибута. С подобными примерами познакомимся в книге далее;
- если значение атрибута заключено в двойные кавычки, оно следует строгому соглашению, но учтите, что применение кавычек не является обязательным и они могут быть опущены. Также допускаются как одинарные, так и двойные кавычки, только соответствующие открывающие и закрывающие кавычки должны совпадать. Обратите внимание, что кавычки в HTML-файлах должны быть прямыми: " , а не наклонными: " ;
- доступные для каждого элемента имена и значения атрибутов определяются в HTML-спецификациях — иными словами, атрибут для элемента самим создать нельзя;
- некоторые атрибуты обязательны — например, атрибуты `src` и `alt` в элементе `img`. В HTML-спецификации также уточняется, какие именно атрибуты необходимы для корректности документа.

После подобного введения вы готовы попробовать свои силы и внести на страницу «Black Goose Bistro» элемент `img` с соответствующими атрибутами, как это сделано в *упражнении 4.4*. Добавим также несколько разрывов строк.

УПРАЖНЕНИЕ 4.4. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЯ

1. Если вы работаете синхронно с автором книги, создайте сначала на жестком диске своего компьютера копию файла изображения, который будет находиться в нужном месте при локальном открытии файла. Файл изображения можно загрузить с сайта книги, доступного по адресу: learningwebdesign.com/5e/materials. Также файл изображения можно получить, сохранив его непосредственно из примера веб-страницы в Интернете по адресу: learningwebdesign.com/5e/materials/ch04/bistro. Щелкните правой кнопкой мыши на изображении гуся (или выполните щелчок на этом изображении, удерживая клавишу **Control** при работе в Mac) и выберите из всплывающего меню пункт **Сохранить на диске** (Save to disk) или ему аналогичный, как показано на рис. 4.14. Назовите файл `blackgoose.png` и сохраните его в папке `bistro`, в которой находится файл `index.html`.
2. Получив изображение, вставьте его в начало заголовка первого уровня, для чего добавьте элемент `img` и его атрибуты:
`<h1>Black Goose
Bistro</h1>`



Windows: Щелкните правой кнопкой мыши на изображении для открытия всплывающего меню.

Mac: Удерживая клавишу **Control**, щелкните на изображении для открытия всплывающего меню. В зависимости от используемого браузера доступные опции могут различаться.

Рис. 4.14. Сохраните файл изображения из веб-страницы

Атрибут `src` поддерживает имя включаемого файла изображения, а атрибут `alt` поддерживает текст, который отобразится, если это изображение недоступно. Оба атрибута необходимо применять с каждым элементом `img`.

- Попробуем разместить изображение над заголовоком, для чего добавим элемент разрыва строки: `
` после элемента `img` — тогда текст заголовка начнется с новой строки.

```
<h1><br>Black Goose  
Bistro</h1>
```

- А теперь добавим разделители в последний абзац, состоящий из трех строк. Добавьте тег `
` туда, где должен располагаться разрыв строки. Попытайтесь, чтобы ваши действия привели к изображению, показанному на рис. 4.15.
- Сохраните файл `index.html`, откройте или обновите его в окне браузера. После этого веб-страница должна иметь вид, показанный на рис. 4.15. Если это не так, проверьте, что файл изображения: `blackgoose.png` — находится в том же каталоге, что и файл `index.html`. Если это так, убедитесь, что никакие символы не пропущены. Особенно обратите внимание на расположение в разметке элемента `img` закрывающих кавычек или скобок.



Рис. 4.15. Страница «Black Goose Bistro» с логотипом

ШАГ 5. ИЗМЕНЕНИЕ ВНЕШНЕГО ВИДА ТЕКСТА С ПОМОЩЬЮ ТАБЛИЦЫ СТИЛЕЙ

В зависимости от контента и целей создания веб-сайта для вас может оказаться вполне достаточным отображение документа в браузере с учетом заданных по умолчанию параметров. Тем не менее не исключено, что вы решите немного подправить начальную страницу сайта «Black Goose Bistro», с тем чтобы улучшить первое впечатление у потенциальных его посетителей. «Улучшить» — значит изменить представление, применив каскадные таблицы стилей (CSS).

В *упражнении 4.5* показано, каким образом можно, применяя некоторые простые правила таблицы стилей, изменить внешний вид текстовых элементов и фон страницы. Не озабочивайтесь, если не все будет понятно с первого раза. Более подробно мы познакомимся с функциями CSS в *третьей части* книги. Но желательно прямо сейчас получить представление о процедуре добавления в имеющуюся структуру «слоя» представления.

УПРАЖНЕНИЕ 4.5. ДОБАВЛЕНИЕ ТАБЛИЦЫ СТИЛЕЙ

1. Откройте файл `index.html`, если он еще не открыт. Применим к веб-странице элемент `style` для обращения к очень простой встроенной таблице стилей. Это — только одна из возможностей по внесению в страницу таблицы стилей, остальные возможности будут рассмотрены в главе 11.
2. Элемент `style` размещается в заголовке документа. Добавим к документу элемент `style`:

```
<head>
    <meta charset="utf-8">
    <title>Black Goose Bistro</title>
    <style>

        </style>
</head>
```

3. Затем добавьте в элемент `style` следующие далее стилевые правила, как это показано ниже. Не волнуйтесь, если вам непонятно, что в результате происходит (хотя это и не так сложно понять на уровне интуиции). Стилевые правила рассматриваются в *части III* книги.

```
<style>
body {
    background-color: #faf2e4;
    margin: 0 10%;
    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. А теперь пришло время сохранить файл и просмотреть его в окне браузера. Изображение должно иметь вид, похожий на показанное на рис. ЦВ-4.16. Если у вас получился иной результат, просмотрите таблицу стилей и убедитесь, что не пропущена точка с запятой или фигурная скобка. Сравните вид вашей веб-страницы с только что добавленными стилями со страницей, в которой использовались стили браузера, добавленные по умолчанию (см. рис. 4.15).

Итак, мы завершили работу над страницей «Black Goose Bistro». Вы не только сформировали свою первую веб-страницу с таблицей стилей, но и узнали об элементах, атрибутах, пустых элементах, блочных и строчных элементах, базовой структуре HTML-документа и правильном использовании разметки. Неплохо для начала!

УСТРАНЕНИЕ ПРОБЛЕМ ПРИ РАЗРАБОТКЕ ВЕБ-СТРАНИЦ

Проведенная демонстрация прошла достаточно гладко, но иногда при выводе созданной вручную HTML-разметки возникают проблемы. К сожалению, один пропущенный символ может привести в негодность всю страницу. А сейчас намеренно добавим ошибку, чтобы обратить ваше внимание на подобные проблемы.

Что получится, если в закрывающем теге выделения: `` игнорировать косую черту? Если лишь один символ окажется не на месте (рис. 4.17), текст оставшейся части документа отобразится выделенным курсивом. Дело в том, что без этой косой черты браузер не сможет отключить назначенное форматирование, которое будет продолжать применяться.

ПРОПУСК КОСОЙ ЧЕРТЫ

Пропуск косой черты в закрывающем теге (или даже пропуск закрывающего тега) для элементов блока, таких, как заголовки или абзацы, может и не иметь столь драматических последствий. Браузеры интерпретируют начало нового блочного элемента как сигнал, что предыдущий блочный элемент блока завершен.

Итак, мы только что увидели, что может произойти в случае отсутствия символа косой черты. Теперь посмотрим, что произойдет, если случайно пропустить угловую скобку в конце первого тега `<h2>` (рис. 4.18).

```
<h2>Catering</h2>
<p>You have fun. <em>We'll handle the cooking.</em> Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.</p>
```

Catering

You have fun. *We'll handle the cooking.* Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.

Location and Hours

Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm;
Friday and Saturday, 11am to midnight

Рис. 4.17. Если не указать косую черту, браузер не узнает, где завершается элемент

```
<h2>The Restaurant</h2>
<p>The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients.</p>
```

The Restaurant

Пропущенный подзаголовок

Без скобки все последующие символы интерпретируются как часть тега, и слово «The Restaurant» исчезает со страницы.

BLACK GOC

The Black Goose Bistro offers casual lunch and d changes regularly to highlight the freshest local in

Catering

You have fun. *We'll handle the cooking.* Black Go a meetup to elegant corporate fundraisers.

Рис. 4.18. Отсутствующая завершающая угловая скобка приводит к тому, что браузер воспринимает последующие символы как часть тега, поэтому текст заголовка не отображается

Вы обратили внимание на отсутствие заголовка? Дело в том, что при отсутствии в теге закрывающей угловой скобки браузер предполагает, что весь последующий текст — вплоть до следующей закрывающей угловой скобки: >, которую он обнаруживает, — является частью открывающего тега `<h2>`. Браузеры не отображают текст внутри тега, поэтому заголовок исчезает. Поэтому браузер игнорирует непонятное для него имя элемента и переходит к следующему элементу.

Преднамеренно вносить ошибки в первые самостоятельно созданные HTML-документы, а затем исправлять их — отличный способ обучения. Если ваши первые веб-страницы идеально написаны, рекомендую поэкспериментировать с кодом,

чтобы уточнить, каким образом браузер реагирует на вносимые изменения. Этот подход будет чрезвычайно полезен в последующем, когда придется обрабатывать страницы, содержащие ошибки. Некоторые наиболее распространенные проблемы упомянуты во *врезке «В чем проблема?»*. Учитите, подобные проблемы не являются уделом новичков. С мелкими ошибками в процессе веб-дизайна постоянно сталкиваются и профессионалы.

В ЧЕМ ПРОБЛЕМА?

Далее приведены некоторые типичные проблемы, возникающие при создании веб-страниц и их просмотре в браузере:

- Я изменил свой документ, но когда я перезагружаю веб-страницу в браузере, она выглядит точно так же.

Возможно, вы не сохранили документ перед перезагрузкой или же сохранили его в другом каталоге.

- Часть страницы не отображается.

Так происходит, если не хватает закрывающей угловой скобки: > или кавычки внутри тега. Это — весьма распространенная ошибка при написании кода HTML вручную.

- Я вставил рисунок с помощью элемента `img`, но отображается лишь значок с «разбитым» изображением.

Изображение может отображаться некорректно в силу двух причин. Во-первых, браузер может просто не найти изображение. Убедитесь в том, что правильно указан адрес (URL) файла изображения (адреса URL будут рассмотрены далее — в главе 6). Удостоверьтесь, что файл изображения действительно находится в указанном каталоге. Если файл имеется, обратите внимание, находится ли он в одном из форматов, которые могут отображаться веб-браузерами: PNG, JPEG, GIF или SVG, и что его имя имеет соответствующее расширение: `png`, `jpeg` или `jpg`, `gif` или `svg`, соответственно.

ВАЛИДАЦИЯ ДОКУМЕНТОВ

Одним из способов, к которому прибегают профессиональные веб-разработчики для выявления ошибок в разметке, — *валидация* (проверка) документов. Что это значит? Для *валидации* документа нужно проверить его разметку и убедиться, что соблюdenы все правила любой используемой версии HTML. Документы, не содержащие ошибок, считаются *валидными*. Настоятельно рекомендуется проверять ваши документы, особенно с учетом требований профессиональных сайтов. Валидные документы согласованы с требованиями различных браузеров, они отображаются быстрее и доступны для более широкой аудитории.

В настоящее время браузеры не требуют, чтобы документы были валидными (другими словами, они делают все возможное, чтобы отобразить их, выявить ошибки и тому подобное), но каждый раз при отклонении от стандарта валидности в способ обработки страницы браузерами или альтернативными устройствами вносится элемент непредсказуемости.

Как же удостовериться, что документ валидный? Можете проверить это самостоятельно или попросить специалиста, но людям свойственно ошибаться, тем более, что вы можете не заметить некоторые правила из спецификаций. Воспользуйтесь *валидатором* (средством проверки) — программным приложением, которое проверит соответствие исходного кода указанной вами HTML-версии. Посмотрите, на чем акцентируют внимание валидаторы:

- подключение объявления DOCTYPE. При отсутствии подобного объявления валидатору неизвестно, какую версию HTML следует проверять;
- указание кодировки символов для документа;
- включение необходимых правил и атрибутов;
- наличие нестандартных элементов;
- несоответствия в тегах;
- ошибки вложенности (некорректное размещение элементов внутри других элементов);
- опечатки и другие мелкие ошибки.

Разработчики используют ряд полезных инструментов для проверки HTML-документов и исправления в них ошибок. Удобный веб-валидатор доступен по адресу: html5.validator.nu. В нем предоставляется возможность загрузить файл или разместить ссылку на страницу, которая уже опубликована в Интернете. На рис. ЦВ-4.19 показан отчет, генерированный валидатором при загрузке версии файла `index.html` для страницы «Bistro», в котором отсутствовала разметка. В таком документе имеется ряд пропущенных элементов, которые не позволяют ему стать валидным. Также демонстрируется исходный проблемный код и предоставляется объяснение, какой вид он должен иметь. Весьма удобно! Встроенные инструменты разработчиков браузеров для Safari и Chrome также располагают валидаторами для быстрой проверки выполненной работы. Некоторые редакторы кода также имеют встроенные валидаторы.

ЭЛЕМЕНТЫ ДЛЯ НАСТРОЙКИ HTML-ДОКУМЕНТА

В этой главе представлены элементы, определяющие настройку метаданных и части контента для HTML-документа. Остальные представленные в упражнениях элементы будут подробно рассмотрены в следующих главах.

Элемент	Описание
Body	Идентифицирует основную часть документа, содержащего контент
Head	Идентифицирует заголовок документа, включающий информацию о документе
Html	Корневой элемент, включающий все иные элементы
Meta	Поддерживает информацию о документе
Title	Вносит заголовок на страницу

КОНТРОЛЬНЫЕ ВОПРОСЫ

Сейчас самое время убедиться, что вы разобрались с основами разметки. Вот несколько вопросов, которые помогут вам проверить, насколько хорошо вы усвоили материал этой главы. Ответы на эти вопросы приведены в *приложении 1*.

1. В чем разница между тегом и элементом?
2. Напишите рекомендованную минимальную разметку для документа HTML5.
3. Укажите, является ли каждое из этих имен файлов приемлемым именем для веб-документа, обведя «Да» или «Нет». Если имя неприемлемо, укажите причину:

a. Sunflower.html	Да	Нет
b. index.doc	Да	Нет
c. cooking home page.html	Да	Нет
d. Song_Lyrics.html	Да	Нет
e. games/rubix.html	Да	Нет
f. %whatever.html	Да	Нет
4. Все следующие примеры разметки неверны. Укажите причины ошибок для каждого примера, а затем перепишите их правильно:
 - a.
 - b. Congratulations!
 - c. linked text</a href="file.html">
 - d. <p>This is a new paragraph<\p>
5. Как бы вы разметили этот комментарий в HTML-документе, чтобы он не отобразился в окне браузера?
product list begins here

ГЛАВА 5

РАЗМЕТКА ТЕКСТА

В этой главе...

- ▶ *Подбор элементов для разметки контента*
- ▶ *Абзацы и заголовки*
- ▶ *Три разновидности списков*
- ▶ *Разбивка контента на разделы*
- ▶ *Строковые элементы текстового уровня*
- ▶ *Обобщающие элементы div и span*
- ▶ *Специальные символы*

Теперь, когда ваш контент готов к работе (вы вычитали его, не так ли) и в него внесена необходимая разметка (`<!DOCTYPE>, html, head, title, meta charset и body`), вы вполне готовы к тому, чтобы внимательно разобраться с элементами разметки. В этой главе рассмотрены элементы, которые применяются для выполнения разметки веб-страниц. Вероятно, их не так много, как можно ожидать, — всего лишь небольшая группа, но использоваться они будут с завидной регулярностью. Из-за того, что в этой главе рассматривается множество разных вопросов, она получилась достаточно велика по объему.

В самом начале нашего погружения в тему элементов разметки следует подчеркнуть, насколько важно подбирать элементы *семантически*, то есть отбирать те из них, которые наиболее точно описывают смысл контента. Если же вам не нравится полученное представление, вы сможете изменить его внешний вид с помощью таблицы стилей. Контент семантически размеченного документа становится доступным, причем эта доступность реализуется в самых разных средах для просмотра: от настольных компьютеров и мобильных устройств до вспомогательных программ чтения с экрана. Благодаря наличию такой разметки, программы чтения, а также программы индексации поисковых систем корректно анализируют предложенный им контент и принимают решения об относительной важности элементов, находящихся на странице.

Взяв на вооружение эти принципы, познакомимся с текстовыми элементами HTML, начиная с основных, — таких, как элементы абзаца.

АБЗАЦЫ

<п>...</п>

Элемент абзаца

Абзацы являются простейшими элементами текстового документа. Укажите абзац с помощью элемента `p`, вставляя в начале абзаца открывающий тег `<p>`, а после него — закрывающий тег `</p>`, как показано в этом примере:

<п>Serif typefaces have small slabs at the ends of letter strokes.
In general, serif fonts can make large amounts of text easier to read.
</п>
<п>Sans-serif fonts do not have serif slabs; their strokes are square
on the end. Helvetica and Arial are examples of sans-serif fonts.
In general, sans-serif fonts appear sleeker and more modern.</п>

СТАНДАРТ HTML5

Изложение элементов разметки в этой книге осуществляется на основе стандарта HTML5, поддерживаемого W3C (www.w3.org/TR/html5/). На момент подготовки книги последней версией HTML5 является HTML 5.2 Proposed Recommendation (www.w3.org/TR/html52/).

Списки, элементы разделения (секционирования) или любые элементы, которые обычно отображаются по умолчанию в виде блоков.

С технической точки зрения вполне допускается пропуск закрывающего тега `</п>`, поскольку это не влияет на корректность документа. Браузер предполагает, что абзац закрыт, если встречает следующий элемент блока. Тем не менее многие веб-разработчики, в том числе и я, предпочитают не пропускать закрытие абзацев и всех других элементов, чтобы сохранить последовательный и четкий характер разметки. Рекомендую пользователям, изучающим разметку, следовать этому правилу.

Визуальные браузеры практически всегда отображают абзацы на новых строках с заданными по умолчанию небольшими промежутками между ними (в терминах CSS абзацы называются блоками). Абзацы могут содержать текст, изображения и другие строчные элементы (так называемый *фразовый контент*), но они не могут включать заголовки, списки,

ИЗБЕГАЙТЕ «ГОЛОГО» ТЕКСТА БЕЗ РАЗМЕТКИ!

Применяйте элементы разметки ко всему тексту документа. Другими словами, текст полностью должен заключаться в какой-либо элемент. Текст, не содержащийся в тегах, является *открытым* или т. н. *анонимным* текстом, а документ, содержащий подобный текст, будет считаться некорректным.

ЗАГОЛОВКИ

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

Элементы заголовков

В предыдущей главе для выделения заголовков на странице «Bistro Black Goose» использовались элементы `h1` и `h2`. На самом деле имеются шесть уровней заголовков: от `h1` до `h6`. После добавления заголовков к контенту браузер применяет их для формирования на странице *структурьи документа*. Вспомогательные средства для чтения — такие, как программы чтения с экрана, ориентируются на схему документа для облегчения представления документа пользователям и перемещения по ним. Кроме того, поисковые системы воспринимают уровни заголовков как часть своих алгоритмов (информация, представленная с применением заголовков более высоких уровней, может иметь большее значение). Поэтому рекомендуется начинать с заголовка 1-го уровня: `h1` и обрабатывать их с учетом числового порядка, формируя логическую структуру и схему документа.

В следующем примере показана разметка для четырех уровней заголовка. Дополнительные уровни заголовков помечаются аналогичным образом.

```
<h1>Type Design</h1>

<h2>Serif Typefaces</h2>
<p>Serif typefaces have small slabs at the ends of letter
strokes. In general, serif fonts can make large amounts of text
easier to read.</p>

<h3>Baskerville</h3>

<h4>Description</h4>
<p>Description of the Baskerville typeface.</p>

<h4>History</h4>
<p>The history of the Baskerville typeface.</p>

<h3>Georgia</h3>
<p>Description and history of the Georgia typeface.</p>

<h2>Sans-serif Typefaces</h2>
<p>Sans-serif typefaces do not have slabs at the ends of
strokes.</p>
```

Разметка, примененная в этом примере, формирует следующую схему документа:

1. Type Design

1. Serif Typefaces

+ text paragraph

1. Baskerville

1. Description

+ text paragraph

2. History

+ text paragraph

2. Georgia

+ text paragraph

2. Sans-serif Typefaces

+ text paragraph

По умолчанию, заголовки в примере отображаются полужирным шрифтом, начиная с очень большого шрифта, применяемого для заголовков 1-го уровня. Ну а для заголовков меньших уровней будет применяться меньший по размеру шрифт, как показано на рис. 5.1. Для изменения внешнего вида контента можно применять таблицу стилей.

The screenshot shows a web page with the following structure and styling:

- h1** —— **Type Design** (Large bold font)
- h2** —— **Serif Typefaces** (Medium bold font)
- h3** —— **Baskerville** (Small bold font)
- h4** —— **Description** (Very small regular font)
- h4** —— **History** (Very small regular font)
- h3** —— **Georgia** (Small bold font)
- h2** —— **Sans-serif Typefaces** (Medium bold font)

Content under each heading:

- Serif typefaces**: Serif typefaces have small slabs at the ends of letter strokes. In general, serif fonts can make large amounts of text easier to read.
- Baskerville**: Description of the Baskerville typeface.
- History**: The history of the Baskerville typeface.
- Georgia**: Description and history of the Georgia typeface.
- Sans-serif typefaces**: Sans-serif typefaces do not have slabs at the ends of strokes.

Рис. 5.1. Заданное по умолчанию отображение заголовков четырех уровней

ТЕМАТИЧЕСКИЕ РАЗРЫВЫ (ГОРИЗОНТАЛЬНЫЕ ЛИНЕЙКИ)

<hr>

Горизонтальная линейка

Чтобы показать, что одна тема завершена и следующая начинается, воспользуйтесь элементом `hr` — для вставки того, что в спецификации называется «тематическим разрывом на уровне абзаца». Элемент `hr` представляет собой логический разделитель между разделами страницы или абзацами без введения нового уровня заголовка.

В устаревших версиях HTML элемент `hr` определялся как «горизонтальная линейка», поскольку вставлял на страницу горизонтальную линию. Браузеры по-прежнему отображают элемент `hr` как трехмерную линейку с тенью и помещают его в отдельную строку с некоторым количеством заданных по умолчанию пробелов сверху и снизу, но в спецификации HTML5 для этого элемента было добавлено новое семантическое имя и определение. Теперь для отображения декоративной линейки лучше сформировать линейку, указав с помощью CSS цветную рамку до или после элемента `hr`.

Элемент `hr` является пустым — его располагают там, где нужно указать тематический разрыв, как продемонстрировано в следующем примере кода и на рис. 5.2:

```
<h3>Times</h3>
<p>Description and history of the Times typeface.</p>
<hr>
<h3>Georgia</h3>
<p>Description and history of the Georgia typeface.</p>
```

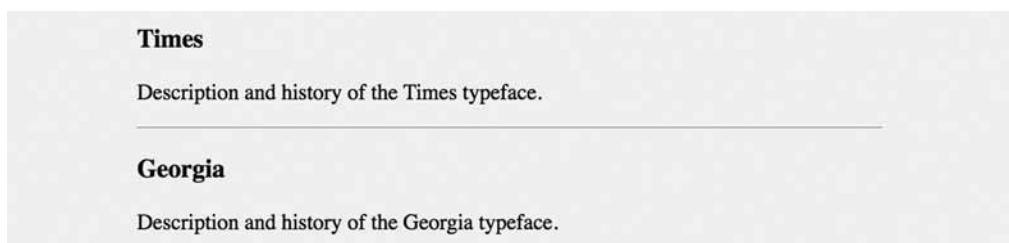


Рис. 5.2. Заданное по умолчанию отображение тематического разрыва (горизонтальной линейки)

СПИСКИ

Людям свойственно располагать информацию в виде списков, поэтому в HTML предлагается три типа элементов для разметки списков:

- *неупорядоченные списки* — наборы пунктов, отображаемые в произвольном порядке;

- *упорядоченные списки* — списки, для которых важна последовательность пунктов;
- *списки описаний* — списки, которые представлены парами «имя-значение». Подобные списки обычно применяются для ввода терминов и определений, но их можно использовать и для других целей.

Все элементы списка — и сами списки, и входящие в них элементы — по умолчанию отображаются как блочные элементы, то есть начинаются с новой строки и имеют некоторое свободное пространство выше и ниже них, изменяемое с помощью таблиц стилей CSS. А теперь подробнее рассмотрим каждую разновидность списка.

Неупорядоченные списки

`...`

Неупорядоченный список

`...`

Элемент списка в неупорядоченном списке

Любой список примеров, имен, компонентов, суждений или параметров можно трактовать как неупорядоченный список. На самом деле в эту категорию попадает большинство списков. По умолчанию неупорядоченные списки отображаются с маркером перед каждым элементом списка, но вид этих списков можно изменять с помощью таблицы стилей, как будет показано позже.

Для оформления неупорядоченного списка пометьте его как элемент `ul`. Открывающий тег `` располагается перед первым элементом списка, а закрывающий тег `` — после его последнего элемента. Затем, чтобы каждый элемент в списке пометить как элемент списка: `li`, заключите его в открывающий и закрывающий теги `li`, как показано в следующем примере кода. Заметьте, что в исходном документе маркеры отсутствуют и браузер добавляет их автоматически (рис. 5.3).

```
<ul>
  <li>Serif</li>
  <li>Sans-serif</li>
  <li>Script</li>
  <li>Display</li>
  <li>Dingbats</li>
</ul>
```

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

Рис. 5.3. Заданное по умолчанию отображение примера неупорядоченного списка: маркеры списка браузер автоматически добавляет сам

В неупорядоченном списке (то есть между начальным и конечным тегами `ul`) может располагаться один или несколько элементов списка — и только они! Туда нельзя

помещать другие элементы, и там не может находиться текст без тегов. Но зато в элемент списка `li` можно поместить любой тип элемента.

А вот еще один важный момент. С помощью различных таблиц стилей можно обратиться к разметке неупорядоченного списка и радикально изменить его внешний вид, как показано на рис. ЦВ-5.4. Здесь отключены маркеры браузера, добавлены собственные маркеры — причем в некоторых вариантах они даже стали похожими на графические кнопки, а элементы выровнены по горизонтали. Исходная же разметка остается прежней.

Упорядоченные списки

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

Упорядоченный список

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

Элемент списка в упорядоченном списке

Упорядоченные списки используются для представления элементов, следующих в определенном порядке, — например, пошаговых инструкций или направления перемещения. Они функционируют так же, как описанные ранее неупорядоченные списки, но определяются с помощью элемента `ol` (что естественно для «упорядоченного списка» — «ordered list» по-английски). Браузер автоматически вместо маркеров вставляет перед упорядоченными элементами списка порядковые номера, поэтому в исходном документе их нумеровать не нужно. При этом упорядочение элементов списка выполняется без изменения их нумерации.

Упорядоченный список должны включать один или несколько элементов списка, как показано в следующем примере и на рис. 5.5:

```
<ol>
  <li>Gutenberg develops moveable type (1450s)</li>
  <li>Linotype is introduced (1890s)</li>
  <li>Photocomposition catches on (1950s)</li>
  <li>Type goes digital (1980s)</li>
</ol>
```

ОТОБРАЖЕНИЕ ПОРЯДКОВЫХ НОМЕРОВ СПИСКА

Если вы формируете упорядоченный список, но не желаете, чтобы отображались цифры, не забывайте, что удалять нумерацию можно с помощью таблиц стилей. Отметьте список семантически как `ol` и настройте его отображение с помощью стилевого правила.

1. Gutenberg develops moveable type (1450s)
2. Linotype is introduced (1890s)
3. Photocomposition catches on (1950s)
4. Type goes digital (1980s)

Рис. 5.5. Заданное по умолчанию отображение упорядоченного списка: при выводе списка браузер автоматически добавляет нумерацию его пунктов

Если нужно, чтобы нумерованный список начинался с номера, отличного от 1, в элементе `ol` можно применить атрибут `start`, что позволит указать другой начальный номер:

```
<ol start="17">
  <li>Highlight the text with the text tool.</li>
  <li>Select the Character tab.</li>
  <li>Choose a typeface from the pop-up menu.</li>
</ol>
```

Полученные элементы списка будут пронумерованы соответственно номерами 17, 18 и 19.

Списки описаний

`<dl>...</dl>`

Список описаний

`<dt>...</dt>`

Название — например, термин или метка

`<dd>...</dd>`

Значение — например, описание или определение

Списки описаний используются для вывода любых пар типа «имя-значение» — например, терминов и их определений, вопросов и ответов на них или других типов слов и связанных с ними сведений. Структура списков описаний немного отличается от рассмотренных ранее двух других видов списков. Список описаний полностью помечается как элемент `dl`. Контент `dl` представляет некоторое количество элементов `dt`, обозначающих имена (удобно представлять их как «термины» — от буквы `t`, `terms` в элементе `dt`), и элементов `dd` — соответствующих им значений (удобно представлять их как «описания» — от буквы `d`, `description` — в элементе `dd`), хотя это лишь только один из вариантов применения таких списков.

Далее приведен пример списка, в котором сопоставляются наборы шрифтов с их описаниями (рис. 5.6):

```
<dl>
  <dt>Linotype</dt>
  <dd>Line-casting allowed type to be selected, used, then
recirculated into the machine automatically. This advance
increased the speed of typesetting and printing dramatically.</dd>

  <dt>Photocomposition</dt>
  <dd>Typefaces are stored on film then projected onto photo-
sensitive paper. Lenses adjust the size of the type.</dd>

  <dt>Digital type</dt>
  <dd><p>Digital typefaces store the outline of the font shape
in a format such as Postscript. The outline may be scaled to
any size for output.</p>
```

```

<p>Postscript emerged as a standard due to its support of
graphics and its early support on the Macintosh computer and
Apple laser printer.</p>
</dd>
</dl>
```

Linotype

Line-casting allowed type to be selected, used, then recirculated into the machine automatically. This advance increased the speed of typesetting and printing dramatically.

Photocomposition

Typefaces are stored on film then projected onto photo-sensitive paper. Lenses adjust the size of the type.

Digital type

Digital typefaces store the outline of the font shape in a format such as Postscript. The outline may be scaled to any size for output.

Postscript emerged as a standard due to its support of graphics and its early support on the Macintosh computer and Apple laser printer.

Рис. 5.6. Заданное по умолчанию отображение списка описаний: описания отделяются от терминов с помощью отступа

Элемент dl может включать только элементы dt и dd. В имена (элементы dt) нельзя помещать заголовки или элементы, которые служат для группировки контента (например, абзацы), но значение (элементы dd) может включать любой тип контента. Например, последний элемент dd в приведенном на рис. 5.6 примере включает два элемента абзаца (добавленный по умолчанию, но ненужный, интервал между абзацами можно устраниТЬ с помощью таблицы стилей).

Вполне возможно наличие нескольких определений для одного термина и наоборот. В таком случае каждая описывающая термин группа включает один термин и несколько определений:

```

<dl>
  <dt>Serif examples</dt>
  <dd>Baskerville</dd>
  <dd>Goudy</dd>

  <dt>Sans-serif examples</dt>
  <dd>Helvetica</dd>
  <dd>Futura</dd>
  <dd>Avenir</dd>
</dl>
```

ВЛОЖЕННЫЕ СПИСКИ

Любой список можно вкладывать в другой список — для выполнения этой операции нужно поместить его в пункт списка. В следующем примере показана структура неупорядоченного списка, вложенного во второй пункт упорядоченного списка:

```

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

Если один неупорядоченный список вкладывается в другой неупорядоченный список, браузер автоматически изменяет стиль маркера для списка второго уровня. К сожалению, при вложении упорядоченных списков стиль нумерации по умолчанию не изменяется. Для этого необходимо самостоятельно устанавливать стили нумерации с помощью правил CSS.

ИЗМЕНЕНИЕ ВИДА МАРКЕРОВ И НУМЕРАЦИИ

Для изменения вида маркеров и номеров списков можно воспользоваться свойствами таблицы стилей типа списка. Например, для неупорядоченных списков можно изменить форму маркера с заданной по умолчанию точки на квадратик или кружок, применить собственное изображение маркера или полностью удалить его. Для упорядоченных списков можно изменять номера на римские цифры (I, II, III или i, ii, iii), буквы (A, B, C или a, b, c) или использовать иные схемы нумерации. На практике же, если список размечен семантически, вообще нет надобности как-то задавать его маркеры или нумерацию. А процедуры, применяемые для изменения стиля списков с помощью CSS, описаны в главе 12.

ДОПОЛНИТЕЛЬНЫЕ ЭЛЕМЕНТЫ РАЗМЕТКИ КОНТЕНТА

Кроме элементов, ответственных за формирование абзацев, заголовков и списков, существуют и специальные элементы разметки текста. Эти элементы также следует использовать в наборе инструментов HTML, хотя они носят несколько специфический характер: длинные цитаты (`blockquote`), предварительно отформатированный текст (`pre`) и фигуры (`figure` и `figcaption`). Характерная особенность этих элементов заключается в том, что они в спецификации HTML5 «группируют контент» (наряду с элементами `p`, `hr`, элементами списка, элементом `main` и обобщающим элементом `div`, о которых пойдет речь в этой главе далее). Следует также отметить, что браузеры по умолчанию отображают такие элементы как блочные. Единственным исключением является недавно появившийся элемент `main`, который не распознается ни одной версией Internet Explorer (хотя и поддерживается браузером Edge), — обратите внимание на информацию во врезке «Поддержка HTML5 в Internet Explorer» далее в этой главе.

Длинные цитаты

```
<blockquote>...</blockquote>
```

Длинная цитата, цитирование на уровне блока

При наличии в контенте обширной цитаты, рекомендации или фрагмента копии из другого источника, выделите ее с помощью элемента `blockquote`. Желательно, чтобы контент, выделенный элементами `blockquote`, включался в другие элементы — такие, как элементы абзацев, заголовков или списков, как показано в следующем примере:

```
<p>Renowned type designer, Matthew Carter, has this to say  
about his profession:</p>
```

```
<blockquote>
```

```
  <p>Our alphabet hasn't changed in eons; there isn't much  
  latitude in what a designer can do with the individual letters.</p>
```

```
<p>Much like a piece of classical music, the score is  
written down. It's not something that is tampered with, and  
yet, each conductor interprets that score differently. There is  
tension in the interpretation.</p>  
</blockquote>
```

На рис. 5.7 приведено стандартное отображение в браузере элемента `blockquote`. Требуемые изменения можно добавить с помощью таблицы стилей CSS.

Renowned type designer, Matthew Carter, has this to say about his profession:

Our alphabet hasn't changed in eons; there isn't much latitude in what a designer can do with the individual letters.

Much like a piece of classical music, the score is written down. It's not something that is tampered with, and yet, each conductor interprets that score differently. There is tension in the interpretation.

РАЗМЕТКА КОРОТКИХ ЦИТАТ

Существует также строчный элемент `q`, применяемый для разметки имеющихся в тексте коротких цитат. Он будет рассмотрен в этой главе далее.

Рис. 5.7. Заданное по умолчанию отображение элемента `blockquote`

Предварительно отформатированный текст

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

Предварительно отформатированный текст

В предыдущей главе отмечалось, что браузеры игнорируют имеющиеся в исходном документе элементы оформления текста — такие, как символы пробелов и возврата каретки (разрыва строки). Но для некоторых типов информации: примеров кода, поэтических цитат и т. п. пробелы важны для передачи смысла. Так что для контента с семантически значимыми символьными пробелами вы можете использовать элемент предварительно отформатированного текста: `pre`. Уникальность этого элемента в том, что текст в нем отображается именно в том виде, как напечатан, включая все возвраты каретки и символьные пробелы. По умолчанию предварительно отформатированный текст также отображается в этом элементе с помощью шрифта постоянной ширины (шрифт, все символы которого имеют одинаковую ширину, называют *моноширинным*) — например, *Courier*. Впрочем, используемый шрифт можно легко изменить с помощью правила таблицы стилей.

СОХРАНЕНИЕ ПРОБЕЛОВ И СИМВОЛОВ РАЗРЫВА СТРОКИ

Для сохранения пробелов и символов разрыва строки также применяется свойство CSS `white-space:pre`.

Пример использования элемента `pre` показан в следующем коде и на рис. 5.8. Для сравнения — в нижней части примера кода и рис. 5.8 отображается тот же контент, но размеченный с помощью элемента абзаца `p`.

```
<pre>  
This is           an           example of  
      text with a           lot of
```

```

        curious
        whitespace.

</pre>

<p>
This is           an           example of
text with a       lot of
                   curious
                   whitespace.
</p>

```

This is an example of
text with a lot of
 curious
 whitespace.

This is an example of text with a lot of curious whitespace.

Рис. 5.8. Предварительно отформатированный текст (вверху) отличается тем, что браузер отображает пробелы и символы разрыва строк (возврата каретки) так, как они присутствуют в исходном документе. Сравните этот элемент с элементом абзаца (внизу), где фигурируют несколько символов пробелов и возврата каретки, а отображаются всего лишь по одному символьному пробелу между словами

Фигуры

<figure>...</figure>

Связанное изображение или ресурс

<figcaption>...</figcaption>

Текстовое описание фигуры

Элемент figure определяет контент, который иллюстрирует или поддерживает некоторый элемент в тексте. Фигура может включать изображение, видео, фрагмент кода, текст или даже таблицу — практически все, что может попасть в поток веб-контента. Контент в элементе figure должен рассматриваться как самостоятельный элемент, и ссылка на него формируется как самодостаточная единица. Это значит, что при перемещении фигуры из исходной позиции в основном потоке (например, во врезку или в примечание), и фигура, и основной поток не должны лишаться собственных смыслов.

Конечно, можно просто добавить на страницу изображение, но при заключении его в теги figure, оно становится более понятным *семантически*. Также этот элемент помогает применять специальные стили именно к фигурам, но не к другим представленным на странице изображениям:

```

<figure>
    
</figure>

```

Если для фигуры нужна текстовая подпись, поставьте элемент `figcaption` выше или ниже контента, находящегося внутри элемента `figure`. Подобный подход допускает большую семантическую вариативность при разметке заголовка, нежели применение простого элемента `p`:

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

Выполняя *упражнение 5.1*, вы можете самостоятельно размечать документ, экспериментируя с уже рассмотренными основными элементами текстовой разметки.

ПОДДЕРЖКА СО СТОРОНЫ БРАУЗЕРОВ

Элементы `figure` и `figcaption` не поддерживаются браузерами *Internet Explorer* версии 8 (и более ранними). Для получения дополнительных сведений по этой теме обратитесь ко врезке «Поддержка HTML5 в *Internet Explorer*» далее в этой главе.

УПРАЖНЕНИЕ 5.1. РАЗМЕТКА РЕЦЕПТУРЫ

Собственники кафе «Bistro Black Goose» решили на своем сайте поделиться с читателями рецептурой предлагаемых блюд и последними новостями. При выполнении упражнений этой главы мы произведем разметку контента.

В упражнении приведен исходный текст рецепта. Вам следует выбирать элементы, которые наилучшим образом семантически соответствуют каждой части контента. Используйте элементы *абзацев*, *заголовков*, *списков* и хотя бы один *специальный элемент контента*.

Вы можете написать теги прямо на этой странице. Или же создать текст в текстовом редакторе, а затем просматривать результаты в браузере. Этот текстовый файл, а также окончательная версия с разметкой доступны на сайте по следующему адресу: learningwebdesign.com/5e/materials.

Tapenade (Olive Spread)

This is a really simple dish to prepare and it's always a big hit at parties. My father recommends:

"Make this the night before so that the flavors have time to blend. Just bring it up to room temperature before you serve it. In the winter, try serving it warm."

Ingredients

1 8oz. jar sundried tomatoes
2 large garlic cloves
2/3 c. kalamata olives
1 t. capers

Instructions

Combine tomatoes and garlic in a food processor. Blend until as smooth as possible.

Add capers and olives. Pulse the motor a few times until they are incorporated, but still retain some texture.

Serve on thin toast rounds with goat cheese and fresh basil garnish (optional).

ОРГАНИЗАЦИЯ КОНТЕНТА НА СТРАНИЦЕ

Рассмотренные до сих пор элементы организуют довольно-таки специфические фрагменты контента: абзац, заголовок, фигуру и т. п. До появления HTML5 для группировки этих фрагментов в более крупные структуры приходилось использовать обобщающий элемент: `div`, который будет подробнее рассмотрен далее. В HTML5 появились новые элементы, которые придают разделам типичной веб-страницы или приложения определенное семантическое значение, включая основной контент: `main`, верхние колониттулы: `header`, нижние колониттулы: `footer`, разделы: `section`, статьи: `article`, навигацию: `nav` и связанный с контентом лишь относительно или дополнительный контент: `aside`. Интересно, что спецификация также содержит давно известный элемент: `address` — как элемент раздела, поэтому этот элемент мы также рассмотрим.

ПОДДЕРЖКА HTML5 В INTERNET EXPLORER

Практически все современные браузеры поддерживают семантические элементы HTML5. Если же ваш браузер не реализует их поддержку, используются правила таблицы стилей, предписывающие браузерам форматировать каждый элемент как элемент блочного уровня. Благодаря такому подходу браузеры будут отображать элементы требуемым образом:

```
section, article, nav, aside, header, footer, main {  
    display: block;  
}
```

К сожалению, подобный подход не смогут увидеть пользователи, которые используют Internet Explorer версий 8 и более ранних (по состоянию на 2017-й год на эти версии приходится менее 1,5% трафика всех браузеров). Версия IE8 не находится на пике популярности, поскольку она привязана к ущедшей в небытие операционной системе Windows Vista. Но если вы публикуете контент на крупном сайте, для которого 1% пользователей составляют тысячи людей, возможно, вам придется искать обходные пути, чтобы корректно донести свой контент до пользователей IE8. Конечно же, совсем необязательно поддерживать эту версию браузера. Однако, несмотря на ее непопулярность, в книге будут приводиться советы, относящиеся к поддержке IE8.

Например, далее описывается обходной путь, применяемый только при работе с IE8 и более ранними версиями этого браузера. Эти браузеры не только не распознают

► элементы HTML5, но игнорируют любые применяемые к ним стили. Чтобы преодолеть этот недостаток, для создания каждого элемента необходимо применять JavaScript — тогда IE узнает о наличии такого элемента, что и позволит реализовать его вложение и стайлинг. Команда JavaScript, создающая элемент `section`, имеет следующий вид:

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

Реми Шарп (Remy Sharp) написал сценарий, который и создает все элементы HTML5 для IE8 и более ранних версий. Он называется HTML5 Shiv (или Shim) и доступен на сервере, ссылку на который можно указать в ваших документах (см. также разд. ««Прокладки» (или «заточки») HTML5» главы 22). Просто скопируйте следующий код в заголовок документа и примените таблицу стилей для оформления новых элементов в виде блоков:

```
<!--[if lt IE 9]>
<script src="//cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.3/
html5shiv.min.js">
</script >
<!-[endif]-->
```

Сценарий HTML5 Shiv также является частью сценария Polyfill Modernizr, который добавляет функциональные возможности HTML5 и CSS3 в браузеры, не располагающие соответствующей поддержкой. Дополнительные сведения о сценарии Modernizr можно найти на сайте modernizr.com. Этим вопросам также посвящена глава 20.

Основной контент

```
<main>...</main>
```

Основная область контента страницы или приложения

Современные веб-страницы просто переполнены различными типами контента: заголовки, врезки, рекламные баннеры, нижние колонтитулы, дополнительная реклама, еще рекламные баннеры и так далее. Поэтому полезно указать на веб-странице основной контент. Так что для определения основного контента страницы или приложения используйте элемент `main`. В этом случае программы чтения с экрана и другие вспомогательные средства получат сигнал, где именно начинается основной контент страницы, что заменяет ссылки **Перейти к основному контенту**, которые применялись для этого ранее. Контент элемента `main` должен быть уникальным для страницы. Другими словами, заголовки, врезки и другие элементы, которые отображаются на нескольких страницах сайта, не должны включаться в раздел `main`:

```
<body>
<header>...</header>
<main>
  <h1>Humanist Sans Serif</h1>
  <!-- продолжение кода -->
</main>
</body>
```

Спецификация W3C HTML5 указывает, что страницы должны иметь только один основной раздел, который не может быть вложен в разделы `article`, `aside`, `header`, `footer`, или `nav`. Если не соблюдать это правило, документ не будет валидным.

Элемент `main` является последним дополнением к списку группирующих элементов HTML5. Можно применять и стилизовать его при работе с большинством браузеров, но для Internet Explorer (включая версию 11, наиболее актуальную на момент подготовки книги) необходимо создать с помощью JavaScript элемент и настроить его как `block` с помощью таблицы стилей, что и обсуждается во врезке «Поддержка HTML5 в Internet Explorer». Обратите внимание, что этот подход поддерживается и в браузере MS Edge.

Верхние и нижние колонтитулы

Многие годы веб-разработчикам приходилось размечать в своих документах разделы верхнего и нижнего колонтитула вручную. Теперь же для этих целей можно воспользоваться элементами `header` и `footer`.

Верхние колонтитулы

`<header>...</header>`

Верхний колонтитул страницы, раздела или статьи

Элемент `header` используется для разметки вводного материала, который обычно отображается в начале веб-страницы или в верхней части раздела либо статьи (соответствующие элементы рассматриваются далее). В раздел верхнего колонтитула можно включать произвольный материал — а именно, все, что имеет смысл. В следующем примере верхний колонтитул документа включает изображение логотипа, заголовок сайта и элементы навигации:

```
<body>
  <header>
    
    <h1>Nuts about Web Fonts</h1>
    <nav>
      <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/">Blog</a></li>
        <li><a href="/">Shop</a></li>
      </ul>
    </nav>
  </header>
  <!--page content-->
</body>
```

В отдельной статье элемент `header` может включать заголовок статьи, имя автора, дату публикации:

```
<article>
  <header>
    <h1>More about WOFF</h1>
```

КОД ``

Встречающийся в примерах код: `` — это разметка, служащая для добавления ссылок на другие веб-страницы. Ссылки будут подробно рассмотрены в главе 6. Обычно значением в этом коде служит интернет-адрес (URL) страницы, но для экономии места здесь вставлена просто косая черта (слэш).

```
<p>by Jennifer Robbins, <time datetime="2017-11-11">  
November 11, 2017</time></p>  
</header>  
<!-- article content here -->  
</article>
```

ЭЛЕМЕНТ TIME

Элемент `time` рассматривается в разд. «Даты и время» в этой главе далее.

Нижние колонтитулы

```
<footer>...</footer>
```

Нижний колонтитул страницы, раздела или статьи

Элемент `footer` используется для указания типа информации, которая отображается в конце страницы или статьи, — например, имя автора, информация об авторских правах, список связанных документов или элементы навигации. Элемент `footer` может применяться ко всему документу или же связываться с определенным разделом или статьей. Если нижний колонтитул включен непосредственно в элемент `body` — до контента основной части или после него, он применяется ко всей странице или приложению. Если же нижний колонтитул включается в элемент секционирования (`section`, `article`, `nav` или `aside`), он воспринимается как нижний колонтитул только для этого раздела. Обратите внимание, что хотя он и называется «нижний колонтитул», необязательно, чтобы он отображался последним в документе или элементе секционирования. Он также может отображаться в начале или ближе к началу, если это имеет смысл.

В следующем примере типичная информация, приводимая в нижней части статьи, помечается как `footer`:

```
<article>  
  <header>  
    <h1>More about WOFF</h1>  
    <p>by Jennifer Robbins, <time datetime="2017-11-11">  
      November 11, 2017</time></p>  
  </header>  
  <!-- article content here -->  
  <footer>  
    <p><small>Copyright  
      &copy; 2017 Jennifer Robbins.</small></p>  
    <nav>  
      <ul>  
        <li><a href="/">Previous</a></li>  
        <li><a href="/">Next</a></li>  
      </ul>  
    </nav>  
  </footer>  
</article>
```

ВЛОЖЕНИЕ ЭЛЕМЕНТОВ HEADER И FOOTER

Элементы `header` и `footer` не могут включать элементы `header` или `footer` в качестве вложенных элементов.

Разделы и статьи

```
<section>...</section>
```

Разделы: тематическое группирование контента

```
<article>...</article>
```

Статьи: Автономная многоразовая композиция

Объемные документы легче воспринять, если разделить их на части. Например, книги разделены на главы, в газетах есть разделы местных новостей, спортивных состязаний, комиксов и т. д. Для разделения больших веб-документов на тематические разделы, применяется элемент `section`. В раздел обычно входит заголовок (внутри элемента `section`), а также контент, который предварительно сгруппирован по смыслу.

ОБОБЩАЮЩИЙ ЭЛЕМЕНТ `div`

Спецификация HTML5 рекомендует для группирования элементов с целью их стилизации использовать обобщающий элемент `div`.

Элемент `section` используется весьма широко: от разделения страницы на основные разделы до выделения тематических разделов в отдельной статье. В следующем примере документ, содержащий информацию о типографских ресурсах, разделен на два раздела в зависимости от типа ресурса:

```
<section>
  <h2>Typography Books</h2>
  <ul>
    <li>...</li>
  </ul>
</section>

<section>
  <h2>Online Tutorials</h2>
  <p>These are the best tutorials on the web.</p>
  <ul>
    <li>...</li>
  </ul>
</section>
```

Элемент `article` используется для разметки отдельных подразделов, которые могут быть как автономными, так и применяться повторно в ином контексте (например, в целях рубрикации). Обычно так поступают при написании журнальных или газетных статей, сообщений в блоге, комментариев или других элементов, которые можно затем извлекать в виде цитат. Специализированный элемент `section` можно представлять как элемент, отвечающий согласием на вопрос: «Может ли этот контент отображаться на другом сайте, сохраняя смысловое содержание?».

Большой раздел `article` можно разбить на несколько разделов:

```
<article>
  <h1>Get to Know Helvetica</h1>
```

```
<section>
  <h2>History of Helvetica</h2>
  <p>...</p>
</section>

<section>
  <h2>Helvetica Today</h2>
  <p>...</p>
</section>
</article>
```

И наоборот, элемент `section` может включать ряд статей веб-документа:

```
<section id="essays">
  <article>
    <h1>A Fresh Look at Futura</h1>
    <p>...</p>
  </article>

  <article>
    <h1>Getting Personal with Humanist</h1>
    <p>...</p>
  </article>
</section>
```

Элементы `section` и `article` легко спутать, вследствие того что их можно вкладывать один в другой и наоборот. Имейте в виду: если контент является автономным и может отображаться вне текущего контекста, лучше разметить его в как `article`.

Размещение контента в виде как бы врезки

`<aside>...</aside>`

Частично связанный с контентом материал

Элемент `aside` идентифицирует контент, который отделяется от окружающего контента, но косвенно с ним связан. В печатных книгах эквивалентом такого контента является врезка, но здесь ее нельзя называть именно так, поскольку оформить врезку — это презентационное описание, а не семантическое. Тем не менее врезка служит хорошей иллюстрацией использования этого элемента. Этот элемент может применяться для выделения цитат, справочной информации, списков ссылок, выносок или какой-либо иной информации, связанной (но не существенно) с документом.

В следующем примере элемент `aside` используется для представления связанного с основной статьей списка ссылок:

```
<h1>Web Typography</h1>
<p>Back in 1997, there were competing font formats and tools
for making them...</p>
```

```
<p>We now have a number of methods for using beautiful fonts on
web pages...</p>
<aside>
  <h2>Web Font Resources</h2>
  <ul>
    <li><a href="http://typekit.com/">Typekit</a></li>
    <li><a href="http://fonts.google.com">Google Fonts</a></li>
  </ul>
</aside>
```

Для элемента `aside` отсутствует заданное по умолчанию отображение. Поэтому нужно сформировать его как блочный документ и настроить его внешний вид, макет с помощью правил таблицы стилей.

Навигация

```
<nav>...</nav>
```

Основные навигационные ссылки

Элемент `nav` поддерживает для разработчиков семантическую возможность навигации по сайту. В главе уже приводился неупорядоченный список, который можно было использовать для навигации верхнего уровня при обработке на сайте каталога шрифтов. При включении этого списка в элемент `nav` можно отследить предназначение этого элемента:

```
<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>
```

Но далеко не все списки связей можно окружить элементами `nav`. В спецификации указано, что элемент `nav` применяется для связей, которые поддерживают начальную навигацию по сайту или большому разделу, статье. Элемент `nav` чрезвычайно полезен в качестве указателя возможной доступности.

Адреса

```
<address>...</address>
```

Контактная информация

Элемент `address` используется для формирования контактной информации, относящейся к автору или владельцу документа. Обычно его размещают в конце документа или в конце раздела (статьи) документа. Элемент `address` весьма удобно использовать в элементе `footer`. Важно отметить, что элемент `address`

не следует применять для указания на странице обычного почтового адреса. Этот элемент предназначен специально для разметки контактной информации автора в Интернете — например, адреса его сайта или адреса его электронной почты. Далее приводится пример использования этого элемента:

```
<address>  
Contributed by <a href=". /authors/robbins/">Jennifer Robbins</a>,  
<a href="http://www.oreilly.com/">O'Reilly Media</a>  
</address>
```

СТРУКТУРА ДОКУМЕНТА

Следует учитывать, что браузеры рассматривают разметку документа как дорожную карту, и генерируют его иерархическую структуру на основе находящихся в контенте заголовков. То есть, если браузеру встречается новый уровень заголовка, в схему добавляется новый раздел.

При работе с устаревшими версиями HTML этот вариант представлял собой единственный способ формирования структуры. В HTML5 появился новый алгоритм формирования структуры, позволяющий разработчикам вносить новый раздел в схему явно, вставляя какой-либо из элементов секционирования: `article`, `section`, `aside` и `nav`. Кроме этих четырех элементов секционирования, спецификация определяет в качестве элементов секционирования и другие элементы (`blockquote`, `fieldset`, `figure`, `dialog`, `details` и `td`), а это означает, что заголовки в этих элементах не трактуются как часть общей структуры документа.

И это, несомненно, хорошая идея, поскольку позволяет перераспределить и объединять контент без нарушения структуры, но, к сожалению, пока еще браузеры не реализуют ее в полной мере, причем в ближайшее время вряд ли удастся этого добиться. Консорциум W3C сохраняет в спецификации элементы секционирования, а также предполагаемое их поведение (поэтому они и упоминаются), но теперь перед ними указывается баннер, рекомендующий придерживаться старого метода формирования иерархических заголовков.

ОБЗОР СТРОЧНЫХ ЭЛЕМЕНТОВ

После идентификации больших частей контента обратим внимание на семантическое содержание отдельных фраз внутри этих частей. Воспользуемся *семантическими элементами текстового уровня*, которые так называются согласно спецификации HTML5. Впрочем, обычно их называют просто *строчными элементами*, поскольку они по умолчанию отображаются в текстовом потоке и не приводят к формированию разрывов строк. Именно так их называли в ранних версиях HTML, еще до появления HTML5.

Строочные элементы текстового уровня

Для добавления в документ всех возможных видов информации в распоряжении разработчика имеется лишь пара десятков семантических элементов текстового уровня. Все они приведены в табл. 5.1.

И хотя знакомиться со всеми элементами текстового уровня удобно именно с помощью таблицы, безусловно, далее мы внимательно рассмотрим каждый из них по отдельности.

Таблица 5.1. Семантические элементы текстового уровня

Элемент	Описание
a	Якорь (привязка) или гипертекстовая связь (см. главу 6)
abbr	Сокращение
b	Визуальное выделение — например, ключевые слова, которые выделяются полужирным шрифтом
bdi	Текст, свойства которого зависят от направления
bdo	Двунаправленное переопределение — явное указание направления текста (слева направо: <code>ltr</code> или справа налево: <code>rtl</code>)
br	Разрыв строки
cite	Цитата, ссылка на заголовок работы — например, название книги
code	Пример компьютерного кода
data	Воспринимаемый компьютером эквивалент дат, времени, весов или других измеряемых величин
del	Удаленный текст — показывает внесенные в документ редакторские правки
dfn	Определяющий экземпляр или первое появление термина в тексте
em	Выделенный текст
i	Альтернативно выделенный текст (курсивный шрифт) или обращение к другому языку
ins	Вставленный текст — указывает на вставку в документ
kbd	Клавиатурный текст — введенный пользователем текст (для технической документации)
mark	Текст, подходящий по смыслу
q	Небольшая цитата в строке
ruby, rt, rp	Поддерживает аннотации или транскрипцию в соответствии с типографикой и идеограммами Восточной Азии
s	Некорректный текст (сплошное зачеркивание)
samp	Образец результата выполнения программы
span	Общий фразовый контент
strong	Контент, имеющий важное значение
sub	Нижний индекс
sup	Верхний индекс
time	Время и дата, записанные в формате, воспринимаемом компьютером
u	Указывает на формальное имя, слово с ошибкой или текст, который будет подчеркнут
var	Переменная или программный аргумент (для технических документов)
wbr	Разрыв слова

Выделенный текст

...

Выделение текста

Элемент `em` указывает, какая часть предложения должна быть выделена или подчеркнута. Размещение элементов `em` влияет на интерпретацию смысла в предложении. Обратите внимание на следующие идентичные предложения, смысл которых изменяется в зависимости от выделенных слов:

```
<p><em>Arlo</em> is very smart.</p>
<p>Arlo is <em>very</em> smart.</p>
```

Первое предложение указывает, *кто* именно умен. А во втором предложении речь идет о *степени разумности*. Заметьте, что именно элемент `em` изменяет значение предложения.

Выделенный с помощью элементов `em` текст по умолчанию всегда отображается курсивом (рис. 5.9, *вверху*), но с помощью таблицы стилей это отображение можно изменить. Программы чтения с экрана обычно используют иной тон голоса для передачи выделенного контента, поэтому элемент `em` следует применять, если имеется соответствующий семантический смысл, а не только в целях отображения текста курсивом.

Arlo is very smart.

Arlo is very smart.

When returning the car, drop the keys in the red box by the front desk.

Рис. 5.9. Заданное по умолчанию отображение выделенного текста (*вверху*) и текста, имеющего важное значение (*внизу*)

ИСТОРИЯ ПОЯВЛЕНИЯ СТРОЧНЫХ ЭЛЕМЕНТОВ

Многие строчные элементы появились вместе с Интернетом и служили для изменения визуального форматирования выделенного текста, поскольку тогда еще не существовала система таблиц стилей. При необходимости выделить текст «жирным» шрифтом его помечали с помощью элемента `b`. Нужен курсив? Тогда применялся элемент `i`. Фактически элемент `font` использовался исключительно для изменения шрифта, цвета и размеров текста (представляете?). Неудивительно, что с появлением HTML5 отказались от применения элементов `font` лишь для представления шрифтов. Тем не менее многие из устаревших элементов презентации (например, `u` для подчеркивания и `s` для зачеркивания) сохранились в HTML5 и получили новые семантические определения: `b` применяется теперь для выделения «ключевых слов», `s` — для интерпретации «неточного текста». Многие строчные элементы вполне ожидаемо отображают стиль (например, элемент `b` отображает полужирный стиль). Другие строчные элементы имеют исключительно семантическое значение (например, `abbr` или `time`) и по умолчанию не создают визуальные эффекты. В общем, если вы хотите изменить способ отображения любых строчных элементов — применяйте правила CSS.

УСТАРЕВШИЕ ТЕКСТОВЫЕ ЭЛЕМЕНТЫ ВЕРСИИ HTML 4.01

Часть давно применяемых текстовых элементов не вошла в HTML5: acronym, applet, basefont, big, center, dir (каталог), font, isindex (поле для поиска), menu, strike, tt (телетайп). Следует упомянуть о них здесь, поскольку можно встретить эти элементы при просмотре источника давнего документа или при обращении к старому инструментарию веб-авторизации. Приведенные здесь элементы не применяются в настоящее время.

Важный текст

...

Выделение важного значения

Элемент strong указывает, что отмеченное им слово или фраза чрезвычайно важны, имеют ключевое значение и должны привлечь первостепенное внимание. В следующем примере элемент strong указывает на часть инструкций, которая требует от читателя дополнительного внимания. Элемент strong не изменяет смысла предложения — он просто привлекает пристальное внимание к важным его составным частям:

```
<p>When returning the car, <strong>drop the keys in the red box by the front desk</strong>.</p>
```

Визуальные браузеры, как правило, по умолчанию отображают важные текстовые элементы с помощью полужирного шрифта (рис. 5.9, внизу). Программы чтения с экрана могут применять более четкий тон голоса для передачи контента, поэтому помечайте текст как strong только в том случае, если он имеет семантическую нагрузку, а не только для выделения его полужирным шрифтом.

Элементы, названные в соответствии с презентационными возможностями

...

Ключевые слова или визуально выделенный текст (полужирный)

<i>...</i>

Альтернативное суждение (курсив)

<s>...</s>

Некорректный текст (перечеркнутый)

<u>...</u>

Аннотированный текст (подчеркивание)

<small>...</small>

Юридический текст: мелкий шрифт (меньший размер шрифта)

Для выделения текста полужирным и курсивным стилем применяются элементы b и i. Эти элементы и подобные им элементы: s, u и small появились давно, на заре

развития Интернета. Эти элементы применялись для передачи инструкций по набору текста (подчеркивание, зачеркнутый текст и текст уменьшенного размера, соответственно). Несмотря на то что эти элементы уже устарели, они включены в HTML5 и получили основанные на шаблонах по их применению соответствующие семантические определения. Браузеры по-прежнему при отображении соответствующих стилей используют эти элементы по умолчанию (рис. 5.10). Но, если вам важно лишь определить стиль шрифта, желательно использовать правила таблицы стилей. Обращайтесь к ним, если они нужны исключительно с семантической точки зрения.

Рассмотрим эти элементы, способы их правильного их применения, а также альтернативные возможности, которые доступны при использовании таблиц стилей.

b

Этот элемент применяется для выделения ключевых слов, названий продуктов и других фраз, которые должны выделяться из окружающего текста, но не имеют дополнительной значимости и не несут смысловой нагрузки [*старое определение*: полуожирный].

Свойство CSS: Для выделения текста полуожирным шрифтом воспользуйтесь свойством `font-weight` — например, так: `font-weight: bold;`

Пример: `<p>The slabs at the ends of letter strokes are called serifs.</p>`

i

Этот элемент указывает на текст, имеющий иную тональность или передающий другое настроение по сравнению с находящимся рядом текстом — например, произносит фразу на другом языке, технический термин или суждение [*старое определение*: курсив].

Свойство CSS: Для выделения текста курсивом используется свойство `font-style` — например, так: `font-style: italic;`

Пример: `<p>Simply change the font and <i>Voila!</i>, a new personality!</p>`

s

Этот элемент указывает на некорректный текст [*старое определение*: зачеркнутый текст].

Свойство CSS: Для перечеркивания текста примените свойство `text-decoration` — например, так: `text-decoration: line-through`

Пример: `<p>Scala Sans was designed by <s>Eric Gill</s> Martin Majoor.</p>`

ЭЛЕМЕНТ ь

Можно представить, каким образом программа чтения с экрана считывает такой текст. Если не нужно, чтобы слово читалось громким, решительным тоном, но оно должно быть выделено, тогда элемент `ь` лучше подходит, чем элемент `strong`.

u

Иногда подчеркивание имеет семантическое значение — например, подчеркивание официального названия на китайском языке или указание слова с орфографической ошибкой после проверки орфографии. Так, в следующем примере при написании слова «Helvetica» была допущена орфографическая ошибка. Учтите, что подчеркнутый текст можно спутать со ссылкой, поэтому следует избегать подобного рода выделений, за исключением некоторых случаев [*старое определение*: подчеркивание].

Свойство CSS: Для подчеркивания текста примените свойство `text-decoration` — например, так: `text-decoration: underline`

Пример: `<p>New York subway signage is set in <u>Helviteca</u>.</p>`

small

Этот элемент указывает на дополнение или примечание к основному тексту — например, это может быть юридическая информация, представленная «мелким шрифтом» внизу документа [*старое определение*: отображение при помощи шрифта, который меньше по размеру, нежели шрифт окружающего текста].

Свойство CSS: Для уменьшения текста используйте свойство `font-size` — например, так: `font-size: 80%`

Пример: `<p><small>(This font is free for personal and commercial use.)</small></p>`

b	The slabs at the ends of letter strokes are called serifs .
i	Simply change the font and <i>Voila!</i> , a new personality!
s	Scala Sans was designed by <u>Erie Gill Martin Majoor</u> .
u	New York subway signage is set in <u>Helvetica</u> .
small	(This font is free for personal and commercial use.)

Рис. 5.10. Заданное по умолчанию отображение элементов b, i, s, u и small

Короткие цитаты

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

Краткая строковая цитата

Применяйте элемент q для обозначения кратких цитат в потоке текста — таких, например, как «To be or not to be». В следующем примере кода и на рис. 5.11 показано действие элемента q:

Matthew Carter says, `<q>Our alphabet hasn't changed in eons.</q>`

В соответствии со спецификацией HTML браузеры автоматически добавляют кавычки вокруг элементов `q`, поэтому в исходный документ их включать не нужно. Некоторые браузеры — например, Firefox, отображают фигурные кавычки, что предпочтительно. Другие браузеры (Safari и Chrome, которые использованы для примеров в книге) отображают прямые кавычки.

Matthew Carter says, "Our alphabet hasn't changed in eons."

Рис. 5.11. Браузеры автоматически добавляют кавычки вокруг элементов `q`

Сокращения и аббревиатуры

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

Аббревиатура или сокращение

Сокращения — это сокращенные варианты слова, которые завершаются точкой (например, «Конн.» для «Коннектикут»). Аббревиатуры — это сокращения, образованные первыми буквами слов из всей фразы (например, NASA или США). Разметка сокращений и аббревиатур с помощью элемента `abbr` содержит полезную информацию для поисковых систем, программ чтения с экрана и других подобных средств. Атрибут `title` поддерживает полную версию сокращенного термина, как показано в следующем примере:

```
<abbr title="Points">pts.</abbr>
<abbr title="American Type Founders">ATF</abbr>
```

ВЛОЖЕННЫЕ ЭЛЕМЕНТЫ

К текстовой строке можно применить два элемента (например, при передаче фразы, которая служит цитатой, но представлена на ином языке), но при этом убедитесь, что элементы вложены корректно. То есть внутренний элемент, включая его закрывающий тег, должен полностью содержаться внутри внешнего элемента и не перекрываться им:

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

Далее приводится пример некорректного вложения элементов. Заметьте, внутренний элемент `i` не закрыт в пределах элемента `q`:

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

Ошибку вложения можно легко обнаружить при рассмотрении небольшого примера, но, если большие части текста вкладывать с применением нескольких уровней, можно и не заметить перекрытий. Одним из преимуществ использования редактора HTML-кода является возможность автоматического корректного закрытия элементов или указания на допущенные ошибки.

ЭЛЕМЕНТ `acronym`

При работе с HTML 4.01 используется элемент `acronym`, специально выделенный для обработки сокращения и аббревиатур, но для HTML5 он не нужен, поскольку в обоих случаях можно применять элемент `abbr`.

Цитирование

<cite>...</cite>

Цитирование

Элемент `cite` применяется для идентификации ссылки на другой документ: книгу, журнал, статью (при указании ее заглавия) и т. п.:

```
<p>Passages of this article were inspired by <cite>The Complete  
Manual of Typography</cite> by James Felici.</p>
```

Цитаты обычно отображаются по умолчанию курсивом.

Определение терминов

<dfn>...</dfn>

Определение термина

Обычно в документе каким-либо образом выделяется первое упоминание или определение специфического слова (термина). В этой книге термины выделяются *курсивом*. В HTML вы можете идентифицировать термины с помощью элемента `dfn` и отформатировать визуально с помощью таблиц стилей:

```
<p><dfn>Script typefaces</dfn> are based on handwriting.</p>
```

Элементы программного кода

<code>...</code>

Код

<var>...</var>

Переменная

<samp>...</samp>

Пример программы

<kbd>...</kbd>

Вводимые пользователем коды клавиш

Часть строчных элементов служит для представления технической документации — например: кода (`code`), переменных (`var`), образцов программного текста (`samp`), вводимых пользователем кодов клавиш (`kbd`). Для меня это служит дополнительным напоминанием о причине появления самого HTML (как известно, Тим Бернерс-Ли разработал HTML в 1989-м году для обмена документами при выполнении работ в лаборатории физики элементарных частиц CERN).

Элементы кода, фрагменты программ и коды нажатия клавиатуры обычно отображаются с помощью шрифта постоянной ширины (его также называют *моноширинным*) — например, заданным по умолчанию шрифтом *Courier*. Переменные величины в коде, как правило, выделяются *курсивом*.

Нижний и верхний индексы

`_{...}`

Нижний индекс

`^{...}`

Верхний индекс

В результате применения элементов разметки нижнего: `sub` и верхнего: `sup` индексов текст отображается с использованием шрифта меньшего размера, расположенного немного ниже (`sub`) или выше (`sup`) базовой линии. Подобные выделения данных применяются в химических формулах либо математических уравнениях.

На следующем примере и на рис. 5.12 показано, каким образом нижний (*слева*) и верхний (*справа*) индексы обычно отображаются в браузере:

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



Рис. 5.12. Нижний (*слева*) и верхний (*справа*) индексы

Подсвеченный текст

`<mark>...</mark>`

Контекстно-релевантный текст

Элемент `mark` обозначает слово, которое для читателя должно нести определенную смысловую нагрузку. Можно применять этот элемент для динамического выделения поискового термина на странице результатов, для привлечения внимания к фрагменту текста в ручном режиме или для указания текущей страницы (в списке страниц). Некоторые дизайнеры (и браузеры) представляют выделенный текст на светлом фоне, как бы отмечая его маркером выделения, — это показано в следующем примере и на рис. 5.13:

```
<p> ... PART I. ADMINISTRATION OF THE GOVERNMENT. TITLE IX.  
TAXATION. CHAPTER 65C. MASS. <mark>ESTATE TAX</mark>. Chapter  
65C: Sect. 2. Computation of <mark>estate tax</mark>.</p>
```

A screenshot of a web browser window. The word 'ESTATE TAX' is highlighted with a yellow background, indicating it has been identified or selected. The surrounding text is in a standard black font on a white background.

Рис. 5.13. В примере разыскиваемые термины идентифицируются с помощью элементов пометки `mark` и с помощью таблицы стилей представляются на желтом фоне, что облегчает читателю их нахождение

Дата и время

`<time>...</time>`

Данные времени

Для человека очевидно, что фраза «4 ноября, полдень» указывает на дату и время. Но подобный контекст неочевиден для компьютерной программы. Элемент `time` позволяет отмечать даты и время таким образом, чтобы это было удобно не только для прочтения, но для кодировки стандартным образом, что применимо для компьютеров. Содержимое элемента информирует пользователей, а атрибут `datetime` представляет эту же информацию в виде, удобном для компьютера.

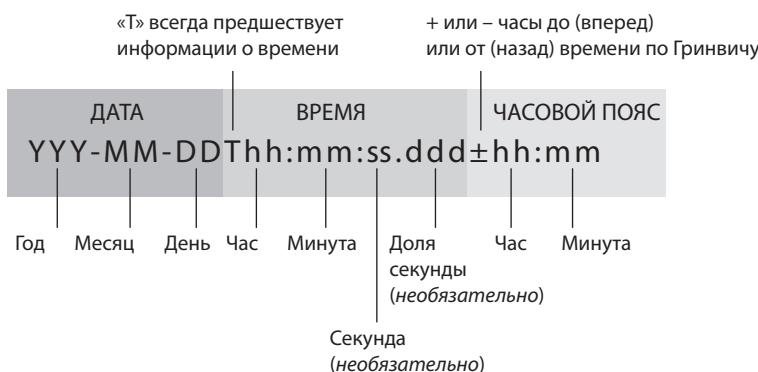
Элемент `time` указывает дату, время или комбинацию даты и времени и может применяться для передачи информации о дате и времени приложению — например,

ЭЛЕМЕНТ `time` «КОНЕЦ ПРОШЛОГО ГОДА» НЕ ВЫДЕЛИТ...

Элемент `time` не предназначен для такой разметки времени, когда невозможно установить точное время или дату, — например, с использованием выражений типа «конец прошлого года» или «вековой рубеж».

для сохранения события в личном календаре. Он также может использоваться поисковыми системами в процессе поиска последних из опубликованных статей. Или же для преобразования информации о времени в альтернативный формат (например, для перевода записи 18:00 в 6 p.m.).

Атрибут `datetime` указывает информацию о дате и/или времени в стандартизированном формате времени (рис. 5.14). Полный формат времени начинается с даты (год-месяц-день). Временной раздел начинается с буквы `T` и содержит список часов (с учетом 24-часовой разбивки), минут, секунд (необязательно) и миллисекунд (также необязательно). Наконец, часовой пояс обозначается количеством часов от (–) или до (+) часового пояса по Гринвичу (GMT, Greenwich Mean Time). Например, запись: –05: 00 указывает на часовой пояс восточного побережья США, который на пять часов отстает от часового пояса GMT.



Пример:

3pm PST on December 25, 2016 — 15:00 по тихоокеанскому времени, 25 декабря 2016 года

Рис. 5.14. Стандартный синтаксис дат и времени

Далее приводятся несколько примеров действительных значений для элемента `datetime`:

- *только время:* 9:30 p.m. (21 час 30 мин)

```
<time datetime="21:30">9:30p.m.</time>
```

- *только дата:* June 19, 2016 (19 июня 2016 года)

```
<time datetime="2016-06-19">June 19, 2016</time>
```

- *дата и время:* Sept. 5, 1970, 1:11a.m. (5 сентября 1970 года, 1.15)

```
<time datetime="1970-09-05T01:11:00">  
Sept. 5, 1970, 1:11a.m.</time>
```

- *дата и время с указанием информации о зоне:*

8:00am on July 19, 2015, in Providence, RI (8.00,
19 июля 2015 года, Провиденс, Род-Айленд)

```
<time datetime="2015-07-  
19T08:00:00-05:00">July 19, 2015,  
8am, Providence RI</time>
```

ЭЛЕМЕНТ `time` БЕЗ АТРИБУТА `datetime`

Элемент `time` может применяться и без атрибута `datetime`, но его контент должен представлять собой действительную строку даты/времени:

```
<time>2016-06-19</time>
```

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ ИНФОРМАЦИИ

Дополнительную информацию о входных и выходных параметрах, которые применяются при указании даты и времени, с иллюстрирующими ее суть примерами можно получить в спецификации HTML5 — в разделе, посвященном элементу `time`: www.w3.org/TR/2014/REC-html5-20141028/text-levelsemantics.html#the-time-element.

Информация, воспринимаемая компьютером

`<data>...</data>`

Данные, воспринимаемые компьютером

Элемент `data` является еще одним инструментом, который помогает компьютерам понять смысл контента. Этот элемент можно применять для всех видов данных, включая даты, время, результаты измерений, взвешиваний, данные о микромире и т. п. Обязательный атрибут `value` поддерживает информацию, воспринимаемую компьютером. Рассмотрим пару примеров:

```
<data value="12">Twelve</data>
```

```
<data value="978-1-449-39319-9">CSS: The Definitive Guide</data>
```

Я не останавливаюсь подробно на данных, воспринимаемых компьютером, поскольку на начальном этапе веб-дизайна вы вряд ли будете иметь с ними дело. Но все же интересно знать, каким образом применяется разметка для передачи полезной информации компьютерным программам и сценариям, а также другим пользователям.

Вставленный и удаленный текст

<ins>...</ins>

Вставленный текст

...

Удаленный текст

Элементы `ins` и `del` помечают правки с указанием частей документа, которые были вставлены или удалены (соответственно). Эти элементы опираются на правила стиля для презентаций, поскольку браузеры не поддерживают возможности этих элементов. Элементы `ins` и `del` могут включать как строчные, так и блочные элементы, в зависимости от типа включенного в них контента:

```
Chief Executive Officer: <del title="retired">Peter Pan</del>
<ins>Pippi Longstocking</ins>
```

Добавление разрывов

Разрывы строк

Разрыв строки

Иногда необходимо в поток текста добавить разрыв строки. Поскольку браузеры в исходном документе игнорируют разрывы строк, нужна специальная директива, чтобы указать браузеру «добавить тут разрыв строки».

Для выполнения этой задачи используется строчный элемент разрыва строки: `br`. Элемент `br` можно применять для разбиения строк адреса или представления стихов. Этот элемент пуст, а это означает, что у него отсутствует контент. Просто добавьте элемент `br` в поток текста, где это необходимо, чтобы создать разрыв строки, как показано на рис. 5.15.

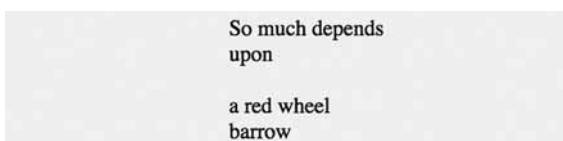


Рис. 5.15. Вставка разрывов строк с помощью элемента `br` (пример взят из стихотворения «Красная тачка» Уильяма Карлоса Уильямса)

К сожалению, элементом `br` зачастую пользуются легкомысленно. Не следует прибегать к нему при формировании разрывов в тексте, который должен представляться списком. Например, не поступайте следующим образом:

```
<p>Times<br>
Georgia<br>
Garamond
</p>
```

Если создается список, примените семантически корректный элемент для неупорядоченного списка и отключите маркеры с помощью таблиц стилей:

```
<ul>
    <li>Times</li>
    <li>Georgia</li>
    <li>Garamond</li>
</ul>
```

Разрывы слов

<**wbr**>

Разрыв слова

Элемент **wbr** поддерживает разрыв слова, позволяя отмечать место, где должен присутствовать такой разрыв («возможность по разрыву строки» согласно спецификации), если недостаточно места для расположения полного слова (рис. 5.16). Подобный подход не позволяет браузерам произвольно разрывать слова и предоставляет авторам возможность контролировать перенос части слова на следующую строку. То есть, если места достаточно, слово остается в целости. При отсутствии предварительно отмеченного разрыва написание слова будет слитным, а если места недостаточно, слово полностью переносится на следующую строку. Обратите внимание, что браузер не добавляет дефис (символ переноса), если часть слова переносится на вторую строку. Элемент **wbr** просто трактует разрыв как символное пространство в середине слова:

ПОДДЕРЖКА ЭЛЕМЕНТА **wbr** БРАУЗЕРАМИ

Элемент **wbr** не поддерживается ни одной из современных версий *Internet Explorer*, но поддерживается в браузере *MS Edge*.

```
<p>The biggest word you've ever heard and this is how it goes:
<em>supercali<b>fragilistic<b>expialidocious</em>!</p>
```

The biggest word you've ever heard and this is how it goes: *supercalifragilistic expialidocious!*

Рис. 5.16. Если слово не помещается в строке, оно разбивается в месте нахождения ближайшего к концу строки элемента **wbr**

ИСПОЛЬЗОВАНИЕ ЯЗЫКОВ, ОТЛИЧАЮЩИХСЯ ОТ ЗАПАДНЫХ

Если целью Интернета является охват всемирной аудитории, следует поддерживать отображение всех мировых языков с их уникальными алфавитами, символами, направлением чтения и специальной пунктуацией. Благодаря усилиям комитета W3C по интернационализации (результат этих усилий часто обозначается как «i18n» — i, затем 18 букв, а затем n) обеспечивается возможность применения определенных в веб-технологиях форматов и протоколов по всему миру. Стремление к интернационализации предполагает применение следующих мер:

- использование кодировки Юникод (Unicode), которая включает буквенные символы, символические знаки, обозначения, иероглифы и пр., относящиеся ко всем активным современным языкам. Кодировка Юникод рассматривалась в главе 4;

- ▶ • объявление основного языка документа с применением двухбуквенного кода языка, соответствующего стандарту ISO 639-1 (доступно на сайте по адресу: www.loc.gov/standards/iso639-2/php/code_list.php). Например, английский язык обозначается «EN», чешский — «CS», немецкий — «DE» и т. д. Используйте атрибут `lang` HTML-элемента для объявления языка всего документа или отдельных нуждающихся в пояснениях элементов;
- адаптация направлений чтения, присущих разных языкам. В HTML атрибут `dir` явно устанавливает направление чтения для документа или элемента: `ltr` (слева направо) или `rtl` (справа налево). В строчных элементах также формируется двунаправленная изоляция, что предотвращает изменение текста внутри элемента в зависимости от расположения текста вне элемента (это свойство может иметь важное значение при вложении пользовательского текста).

Например, при включении в английский документ фрагмента, написанного на иврите, применяйте атрибут `dir`, что позволит указать направление чтения при отображении фраз:

```
<p> This is how you write Shalom:  
<span dir="rtl"> שalom </span></p>
```

- поддержка системы, допускающей *аннотации ruby*, — знаков, которые отображаются над иероглифами восточноазиатских языков и служат подсказками для их произношения или перевода (элементы `ruby`, `rt` и `rp`). Дополнительные сведения по этой теме можно найти в спецификации.

На сайте W3C Internationalization Activity опубликован полный набор методов и ресурсов для разработки HTML и CSS, которые помогут в процессе интернационализации ваших текстов: www.w3.org/International/techniques/authoring-html.

Выполнив предыдущее *упражнение 5.1*, вы познакомились с 32 новыми элементами. Пришло время опробовать некоторые из строчных элементов в *упражнении 5.2*.

УПРАЖНЕНИЕ 5.2. ИДЕНТИФИКАЦИЯ СТРОЧНЫХ ЭЛЕМЕНТОВ

Этот небольшой пост для страницы новостей «Black Goose Bistro» даст вам возможность идентифицировать и разметить различные строчные элементы. Посмотрите, сможете ли вы найти фразы для точной разметки с помощью следующих элементов:

b br cite dfn em
i q small time

Поскольку разметка всегда в чем-то субъективна, ваша итоговая разметка может отличаться от моей окончательной разметки, но в любом случае вы сможете использовать все строчные элементы. В этом упражнении задана фраза, к которой можно применить два элемента (подсказка: ищите термин на другом языке). Не забудьте правильно вставить элементы, закрыв внутренний элемент, перед тем как закрыть внешний. Также убедитесь в том, что все элементы текстового уровня содержатся в элементах блока.

Можете писать теги прямо на этой странице. Если же вы хотите воспользоваться текстовым редактором и просмотреть результат в браузере, загрузите

текстовый файл вместе с результирующим кодом, используя следующую ссылку: learningwebdesign.com/5e/materials.

```
<article>

<header>
<p>posted by BGB, November 15, 2016</p>
</header>

<h2>Low and Slow</h2>
<p>This week I am extremely excited about a new cooking technique called sous vide. In sous vide cooking, you submerge the food (usually vacuum-sealed in plastic) into a water bath that is precisely set to the target temperature you want the food to be cooked to. In his book, Cooking for Geeks, Jeff Potter describes it as ultra-low-temperature poaching.</p>
<p>Next month, we will be serving Sous Vide Salmon with Dill Hollandaise. To reserve a seat at the chef table, contact us before November 30.</p>

<p>blackgoose@example.com
555-336-1800</p>

<p>Warning: Sous vide cooked salmon is not pasteurized. Avoid it if you are pregnant or have immunity issues.</p>
</article>
```

ОБОБЩАЮЩИЕ ЭЛЕМЕНТЫ (**DIV И SPAN**)

<div>...</div>

Обобщающий элемент блочного уровня

...

Обобщающий строчный элемент

Что же предпринять, если ни один из рассмотренных до сих пор элементов не описывает точно необходимый контент? Информация имеет бесконечное множество разновидностей, а семантических элементов не так уж и много. К счастью, HTML предоставляет два обобщающих элемента, которые подстраиваются для описания контента. Элемент `div` указывает раздел контента, а `span` конкретизирует слово или фразу контента, для которой отсутствует элемент текстового уровня. Эти универсальные элементы приобретают смысл и контекстное значение при помощи атрибутов `id` и `class`, которые будут рассмотрены далее.

Элементы `div` и `span` не располагают собственными презентационными возможностями, поэтому для их форматирования можно применять таблицы стилей на свое усмотрение. Фактически эти универсальные элементы служат основным инструментом при реализации основанного на стандартах веб-дизайна, поскольку позволяют точно описывать контент и предлагают множество дополнительных

возможностей для добавления правил стиля. Также с их помощью JavaScript-элементы получают доступ к элементам страницы и могут даже ими манипулировать.

Рассмотрим возможности по использованию элементов `div` и `span`, а также атрибутов `id` и `class` при структурировании контента.

Группировка, выполняемая с помощью элементов `div`

Используйте элемент `div` для логической группировки на странице контента или элементов. С помощью этого элемента можно указать, что они составляют одну концептуальную единицу или должны рассматриваться в качестве такой единицы при использовании CSS или JavaScript. Отмечая связанный контент как `div` и присваивая ему уникальный `id` или указывая, что этот контент является частью класса, вы задаете единый контекст для элементов всей группы. Рассмотрим несколько примеров, иллюстрирующих применение элементов `div`.

В следующем примере элемент `div` применяется как контейнер при группировке изображения и двух абзацев в группу `listing` продукта:

```
<div class="listing">
  
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type
  design.</p>
</div>
```

Путем помещения этих элементов в раздел `div` мы концептуально их связываем. Поэтому элементы `p` в этом разделе можно стилизовать иначе, чем другие элементы `p` в документе.

СОВЕТ: ЗНАЙТЕ МЕРУ, ИСПОЛЬЗУЯ `div`

Элементы `div` можно вкладывать в другие элементы `div`, но во всем следует знать меру. Ваша задача — максимально упростить разметку, поэтому добавляйте элемент `div` только в том случае, если это необходимо для формирования логической структуры, в соответствии со стилевыми требованиями или нужно по сценарию.

Далее показан еще один весьма распространенный вариант применения `div`, когда этот элемент используется для разбики страницы на разделы для выполнения верстки. В следующем примере заголовок и несколько абзацев заключаются в элемент `div` и обозначаются как `news` (новости):

```
<div id="news">
  <h1>New This Week</h1>
  <p>We've been working on...</p>
  <p>And last but not least,... </p>
</div>
```

В результате мы получим пользовательский элемент, которому и дано название `news`.

Возможно, вы подумали: «Нельзя ли для этого применить элемент `section`?» Что ж, на практике авторы могут теперь отставить в сторону универсальные элементы `div`, поскольку в HTML5 имеются более удобные семантические элементы секционирования.

Определение фразы с помощью элемента *span*

Элемент *span* обладает теми же возможностями, что и элемент *div*, но применяется для разметки строк и не добавляет их разрывов. Поскольку элементы *span* являются строчными, они содержат только текст и другие строчные элементы (другими словами, в элемент *span* нельзя помещать заголовки, списки, элементы группирования контента и т. п.). Рассмотрим некоторые примеры.

Поскольку элемента *telephone* в природе не существует, для разметки телефонных номеров можно использовать элемент *span*. В следующем примере каждый телефонный номер размечен с помощью элемента *span* и классифицируется как *tel*:

```
<ul>
  <li>John: <span class="tel">999.8282</span></li>
  <li>Paul: <span class="tel">888.4889</span></li>
  <li>George: <span class="tel">888.1628</span></li>
  <li>Ringo: <span class="tel">999.3220</span></li>
</ul>
```

Обратите внимание, каким образом классифицированные элементы *span* добавляют значение к тому, что в противном случае выглядело бы строкой случайных цифр. В качестве бонуса элемент *span* позволяет применять один и тот же стиль к телефонным номерам по всему сайту (например, гарантирует, что в них не появятся разрывы строк, для чего применяется CSS объявление *white-space: nowrap*). Информация в этом виде привычна не только для пользователей, но и для компьютерных программ, которым понятно, что “tel” — это номер телефона. Фактически, некоторые значения, включая “tel”, стандартизированы в системе разметки, известной как Microformats, что повышает пользу веб-контента для программного обеспечения (см. врезку «Кратко о структурированных данных»).

Атрибуты *id* и *class*

В предыдущих примерах атрибуты *id* и *class* применялись для поддержки контента обобщающих элементов *div* и *span*. Атрибуты *id* и *class* используются с различными целями, и важно учитывать эту разницу.

Идентификация с помощью атрибута *id*

Атрибут *id* служит для придания элементу в документе уникального идентификатора. То есть значение *id* должно использоваться в документе только один раз. Удобно применять его для назначения имени определенному элементу, как если бы речь шла о фрагменте данных, — см. врезку «Значения *id* и *class*», где можно ознакомиться с процедурой предоставления значений для атрибута *id*.

В следующем примере для уникальной идентификации каждого элемента списка используются стандарты ISBN (международные стандартные номера книг). Две различные книги не могут иметь один *id*.

```

<div id="ISBN0321127307">
    
    <p><cite>The Complete Manual of Typography</cite>, James
    Felici</p>
    <p>A combination of type history and examples of good and
    bad type.
    </p>
</div>

<div id="ISBN0881792063">
    
    <p><cite>The Elements of Typographic Style</cite>, Robert
    Bringhurst
    </p>
    <p>This lovely, well-written book is concerned foremost with
    creating beautiful typography.</p>
</div>

```

Веб-разработчики также применяют атрибут `id` для идентификации различных разделов страницы. В следующем примере в документе может содержаться не более одного элемента с `id` «`links`» или «`news`»:

```

<section id="news">
    <!-- news items here -->
</section>

<aside id="links">
    <!-- list of links here -->
</aside>

```

Классификация с помощью атрибута `class`

Атрибут `class` классифицирует элементы, собирая их в концептуальные группы, следовательно, в отличие от атрибута `id`, имя класса может использоваться несколькими элементами совместно. Включая элементы в один и тот же класс, можно применять стили ко всем помеченным элементам одновременно с помощью одного правила стиля или же манипулировать всеми ими при помощи сценария. Рассмотрим процедуру классификации некоторых элементов из предыдущего примера с книгами. В следующем примере добавлены атрибуты класса, что позволит каждый элемент `div` классифицировать как «`listing`», а абзацы классифицировать как «`description`»:

```

<div id="ISBN0321127307" class="listing">
    <header>
        
        <p><cite>The Complete Manual of Typography</cite>, James
        Felici</p>
    </header>
    <p class="description">A combination of type history and
    examples of good and bad type.</p>
</div>

```

```
<div id="ISBN0881792063"
class="listing">
  <header>
    
    <p><cite>The Elements of
Typographic Style</cite>, Robert Bringhurst
    </p>
  </header>
  <p class="description">This lovely, well-written book is
concerned foremost with creating beautiful typography.</p>
</div>
```

Обратите внимание, что один элемент может иметь как атрибут class, так и атрибут id.

Элементы также могут относиться к нескольким классам. Если имеется список значений класса, просто разделите его пробелами. В следующем примере каждый элемент div классифицирован как «book» (книга), что позволит отделить его от возможных списков «cd» или «dvd» в других местах этого документа:

```
<div id="ISBN0321127307"
class="listing book">
  
  <p><cite>The Complete Manual of
Typography</cite>, James Felici</p>
  <p class="description">
A combination of type history and
examples of good and bad type.</p>
</div>

<div id="ISBN0881792063"
class="listing book">
  
  <p><cite>The Elements of Typographic Style</cite>, Robert
Bringhurst
  </p>
  <p class="description">This lovely, well-written book is
concerned foremost with creating beautiful typography.</p>
</div>
```

ЗНАЧЕНИЯ ID И CLASS

В HTML5 значения атрибутов id и class должны включать хотя бы один символ (т. е. не могут быть пусты) и не могут содержать пробелы. В качестве значения может служить практически любой символ. В более ранних версиях HTML имелись ограничения на значения идентификаторов (например, идентификаторы должны были начинаться с буквы), но эти ограничения сняты в HTML5.

Идентификация и классификация всех элементов

Атрибуты id и class используются не только в составе элементов div и span — они в HTML носят *глобальный характер* (см. врезку «Глобальные атрибуты»). Это значит, что их можно применять вместе со всеми элементами HTML. Например, следующий упорядоченный список можно идентифицировать как «directions» вместо заключения его в элемент div:

СОВЕТ ПО ИСПОЛЬЗОВАНИЮ АТРИБУТОВ ID И CLASS

В процессе идентификации пользуйтесь атрибутом id. Для классификации применяйте атрибут class.

```
<ol id="directions">
  <li>...</li>
  <li>...</li>
  <li>...</li>
</ol>
```

Итак, вы получили представление об использовании атрибутов `class` и `id` для идентификации элементов в документах и придания этим документам структурированности. Мы продолжим работать с ними в главах *части III* книги, посвященных таблицам стилей. Во *врезке* «Кратко о структурированных данных» рассматриваются более продвинутые способы добавления в документы данных, воспринимаемых компьютерами.

ГЛОБАЛЬНЫЕ АТРИБУТЫ

HTML5 определяет набор атрибутов, которые можно применять с каждым элементом HTML. Их называют *глобальными атрибутами*:

`accesskey`
`class`
`contenteditable`
`dir`
`draggable`
`hidden`
`id`
`lang`
`spellcheck`
`style`
`tabindex`
`title`
`translate`

В *приложении 2* приведены все глобальные атрибуты, их значения и определения.

КРАТКО О СТРУКТУРИРОВАННЫХ ДАННЫХ

Нам, людям, весьма несложно заметить разницу между кулинарным рецептом и обзором кинофильма. Однако поисковые системы и другие компьютерные программы этих различий не замечают. Если для разметки документа используется исключительно HTML, все браузеры воспринимают его как совокупность абзацев, заголовков и других семантических элементов документа и не более того. Поэтому используйте *структурированные данные*, которые улучшают восприимчивость контента компьютером, что помогает поисковым системам предоставлять более интеллектуальные результаты, лучше соответствующие запросам пользователей, а также получать пользовательский опыт — например, при выборке информации со страницы о событиях и внесения ее в приложение для календаря пользователя.

Существует несколько стандартов для структурирования данных, причем все они основаны на сходных подходах. Во-первых, они идентифицируют и называют «представляемый объект». Затем указывают на свойства этого объекта. «Объектом» может быть человек, событие, продукт, фильм... почти все, что может находиться на веб-странице. Свойства объекта представляются парами «имя – значение». Например, «актер», «режиссер» и «продолжительность» — это свойства фильма. Значения этих свойств отображаются в качестве контента элемента HTML. Совокупность стандартизованных терминов, присвоенных «объектам», а также соответствующих им свойств, формируют так называемый *словарь*.

Наиболее популярными стандартами, описывающими добавление структурированных данных, являются Microformats, Microdata, RDFa (и RDFa Lite) и JSON-LD. Стандарты отличаются синтаксисом, который применяется для добавления информации об объектах и их свойствах.

Microformats (microformats.org)

Одна из первых попыток получить больше пользы из веб-контента привела к формированию стандартизованных значений для имеющихся атрибутов HTML: `id`, `class` и `rel`. Это не документированный стандарт, а широко применяемое соглашение, которое весьма просто в реализации. Существует около десятка устоявшихся словарей Microformat для определения людей, организаций, событий, продуктов и т. п. Рассмотрим небольшой пример, демонстрирующий применение разметки по отношению к человеку с помощью Microformats:

```
<div class="h-card">
<p class="p-name">Cindy Sherman</p>
<p class="p-tel">555.999-2456</p>
</div>
```

Microdata (html.spec.whatwg.org/multipage/microdata.html)

Microdata — это HTML-стандарт WHATWG (Web Hypertext Application Technology Working Group, Рабочая группа по технологиям веб-гипертекста), который применяет для определения объектов и их свойств специфичные для Microdata атрибуты: `itemscope`, `itemtype`, `itemprop`, `itemid` и `itemref`). Рассмотрим пример применения разметки по отношению к человеку с помощью свойств Microdata:

```
<div itemscope itemtype="http://schema.org/Person">
<p itemprop="name">Cindy Sherman</p>
<p itemprop="telephone">555.999-2456</p>
</div>
```

Для получения дополнительных сведений о WHATWG обратитесь к [приложению 3](#).

RDFa и RDFa Lite (www.w3.org/TR/xhtml-rdfa-primer/)

В 2013-м году консорциум W3C исключил Microdata из спецификации HTML5, а бремя обработки структурированных данных было передано RDFa (Resource Description Framework in Attributes, Фреймворк описания ресурсов в атрибутах) и его упрощенному варианту RDFa Lite. Для обработки HTML-контента применяются специфические атрибуты: `vocab`, `typeof`, `property`, `resource` и `prefix`). Далее приведена применяемая по отношению к человеку разметка, аналогичная стандарту Microdata, но реализованная с помощью RDFa:

```
<div vocab="http://schema.org" typeof="Person">
<p property="name">Cindy Sherman</p>
<p property="telephone">555.999-2456</p>
</div>
```

JSON-LD (json-ld.org)

JSON-LD (JavaScript Object Notation to serialize Linked Data, Запись объекта JavaScript, применяемая для сериализации связанных данных) предоставляет другой подход, предусматривающий размещение типов объектов и их свойств в удаленном из разметки HTML сценарии. Далее приводится версия разметки JSON-LD, применяемой по отношению к человеку:

```
▶ <script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type": "Person",
  "name": "Cindy Sherman",
  "telephone": "555.999-2456"
}
</script>
```

Можно создать и собственный словарь для применения на пользовательских сайтах, который будет намного лучше, чем стандартизованный словарь. Благодаря усилиям крупных поисковых систем был создан мега-словарь Schema.org, включающий стандартизованные свойства для сотен «объектов» — таких, как сообщения в блогах, фильмы, книги, продукты, обзоры, люди, организации и т. д. Словари Schema.org могут применяться со стандартами Microdata, RDFa и JSON-LD, а Microformats поддерживает собственные отдельные словари. Ссылки на словарь Schema.org «Person» вы найдете в коде приведенных здесь примеров разметки. Дополнительную доступную информацию можно почерпнуть со страницы Schema.org «Getting Started», где содержится несложное для ознакомления введение: schema.org/docs/gs.html.

Ну вот, вы прочли краткое введение в структурированные данные. Как только вы освоитесь с базовой семантикой HTML, эта тема, безусловно, привлечет ваше пристальное внимание. В Интернете доступно множество информации, посвященной структурированным данным.

УЛУЧШЕНИЕ ДОСТУПНОСТИ ДОКУМЕНТА С ПОМОЩЬЮ ARIA

Занимаясь веб-дизайном, следует учитывать, что пользователи имеют определенный опыт работы со вспомогательными технологиями для навигации по страницам и взаимодействия с веб-приложениями. Пользователи могут воспринимать представленный на странице контент на слух, используя программу чтения с экрана, и применять для навигации по странице возможности клавиатуры, джойстика, голосовые команды или другие средства, которые реализуют процедуры навигации по странице без привлечения мыши.

Функции многих элементов HTML становятся понятнее, если рассматривать (или изучать) именно исходный код HTML. Например, элементы, применяемые для разметки названий, заголовков, списков, изображений и таблиц, имеют в контексте страницы подразумеваемое значение, но обобщающие элементы, такие, как `div` и `span`, не обладают соответствующей семантикой, которая необходима для их интерпретации вспомогательными средствами. При обращении к полнофункциональным веб-приложениям, особенно к тем, которые существенно зависят от JavaScript и AJAX (см. далее), одна лишь разметка не дает исчерпывающих подсказок относительно целей применения элементов или вариантов выбора методики управления формой, которая востребована в тот или иной момент.

К счастью, можно воспользоваться ARIA (Accessible Rich Internet Applications, Доступные многофункциональные интернет-приложения) — стандартизованным набором атрибутов, использование которых упрощает навигацию по страницам и применение интерактивных функций. Эта спецификация создана и поддерживается Рабочей группой инициативы доступности веб-сайтов (WAI), поэтому ее также называют WAI-ARIA. Программный продукт ARIA определяет те роли, состояния и свойства, которые разработчики могут добавлять в разметку и сценарии для поддержки более насыщенной семантической информации.

AJAX

Возможности программного продукта AJAX (Asynchronous JavaScript and XML) рассматриваются во врезке, приведенной в главе 22.

Роли

Роли описывают функции или цели элемента в контексте документа. Среди ролей следует упомянуть: `alert`, `button`, `dialog`, `slider` и `menubar`, хотя имеются и многие другие. Например, с помощью таблицы стилей, как было показано ранее, неупорядоченный список можно преобразовать в меню параметров с вкладками, но как быть, если вы *не видите*, что он стилизован таким образом? Чтобы пояснить назначение списка, добавьте строку `role="toolbar"`:

```
<ul id="tabs" role="toolbar">
  <li>A-G</li>
  <li>H-O</li>
  <li>P-T</li>
  <li>U-Z</li>
</ul>
```

Рассмотрим другой пример, демонстрирующий использование предупреждающего сообщения в идентификаторе `status` элемента `div`:

```
<div id="status" role="alert">You are no longer connected to
the server.</div>
```

Некоторые роли служат как бы «ориентирами», помогающим читателям найти путь в документе, — например: `navigation`, `banner`, `contentinfo`, `complementary` и `main`. Можно заметить, и это не случайно, что определенные «ориентиры» похожи на элементы структурирования страницы, которые появились в HTML5. Серьезным преимуществом применения улучшенных элементов семантического секционирования является тот факт, что их можно использовать в качестве ориентиров, заменяя `<div id="main" role="main">` на `main`, то есть — проще говоря — вместо строки HTML-кода:

```
<div id="main" role="main"> . . . </div>
```

использовать поддерживаемый современным HTML тер `<main>`:

```
<main> . . . </main>
```

При достижении точно такого же результата код получается заметно короче и нагляднее.

Большинство современных браузеров распознают неявные роли новых элементов, но некоторые разработчики явно добавляют роли ARIA, поскольку не все браузеры удовлетворяют единым требованиям. Элементы секционирования объединяются с ролями ориентиров ARIA следующим образом:

Роль `BANNER`

Роль `banner` применяется в том случае, если заголовок относится ко всей странице, а не только к разделу или абзацу.

```
<nav role="navigation">
<header role="banner">
<main role="main">
<aside role="complementary">
<footer role="contentinfo">
```

Состояния и свойства

При работе с ARIA также определяется большой список состояний и свойств, которые применяются к интерактивным элементам, таким, как виджеты форм и динамический контент. Состояния и свойства обозначаются атрибутами с префиксом `aria-`, например: `aria-disabled`, `aria-describedby`, и т. д.

Различия между состоянием и свойством невелики. Для свойств значение атрибута, скорее всего, будет стабильным — например `aria-labelledby`, что свидетельствует о связывании меток с соответствующими элементами управления формой, или `aria-haspopup`, указывающее на наличие у элемента всплывающего меню. Значения состояний с большей вероятностью могут изменяться при взаимодействии пользователя с элементом — например: `aria-selected`.

Источники дополнительной информации

Очевидно, что для изучения возможностей ARIA содержащейся в этом разделе информации недостаточно, и вы вряд ли сейчас сразу сможете уверенно приступить к использованию ARIA, но приведенные здесь сведения дают вам представление о принципах его работы и о его потенциальной ценности. Обратите внимание на несколько рекомендаций, которые следует учесть при наработке профессиональных навыков:

- WAI-ARIA Working Draft (www.w3.org/TR/wai-aria-1.1/) — это текущий рабочий проект спецификации на момент подготовки книги;
- ARIA в HTML (www.w3.org/TR/html-aria/) — этот рабочий проект W3C (W3C Working Draft) помогает разработчикам правильно применять атрибуты ARIA с HTML. Здесь содержится большой список для каждого элемента HTML, независимо от того, имеет ли элемент неявную роль (при которой ARIA не следует применять), а также включает сведения о применяемых ролях, состояниях и свойствах;

ПРИМЕЧАНИЕ

В настоящее время спецификация W3C HTML включает информацию о ролях и свойствах ARIA, которые используются при описании каждого элемента HTML (www.w3.org/TR/html52/).

- ресурсы ARIA на MDN Web Docs (developer.mozilla.org/en-US/docs/Web/Accessibility)

- ARIA) — этот сайт предлагает большое число ссылок на ARIA и современные ресурсы. Он станет хорошей отправной точкой для дальнейшего изучения;
- HTML5 Accessibility (www.html5accessibility.com) — на этом сайте тестируются новые возможности HTML5, поддерживаемые большинством браузеров.

ЭКРАНИРОВАНИЕ СИМВОЛОВ

В заключение главы рассмотрим еще одну тему, относящуюся к обработке текста. Название раздела неточно поясняет суть дела, хотя и отражает реальное положение вещей.

Известно, что когда браузер анализирует HTML-документ, то при встрече с символом < интерпретирует его как начало тега. Но как быть, если необходим просто символ «меньше»? Символы, которые могут неверно толковаться как часть кода, должны быть в исходном документе *экранированы*. При этом экранирование означает, что вместо ввода символа используется числовая или именованная *ссылка на сущность символа*. Если браузер встречает ссылку на символ, то при отображении страницы соответствующий символ и появляется.

Существуют два способа экранирования символа:

- использование заранее заданного сокращенного имени для символа — его *именованной сущности*;
- использование назначенного числового значения, соответствующего позиции в наборе кодированных символов, — его *числовой сущности*. Числовые значения могут записываться в десятичном или шестнадцатеричном форматах.

СОБСТВЕННУЮ СУЩНОСТЬ
СОЗДАТЬ НЕЛЬЗЯ...

В HTML определяются сотни именованных сущностей в качестве составных частей языка разметки, то есть нельзя создать собственную сущность.

Все ссылки на символы начинаются со знака & (амперсанд) и завершаются знаком ; (точка с запятой).

А теперь поясним сказанное на примере. Предположим, что нужно включить в текст символ «меньше, чем», поэтому там, где он должен появиться, вставляется именованная сущность: < или ее числовой эквивалент: < (рис. 5.17):

```
<p>3 tsp. &lt; 3 Tsp.</p>
```

или:

```
<p>3 tsp. &#060; 3 Tsp.</p>
```

3 tsp. < 3 Tsp.

СОВЕТ ПО ВЫПОЛНЕНИЮ РАЗМЕТКИ

Подобные символьные сущности полезны, если необходимо отобразить на веб-странице пример HTML-разметки.

Рис. 5.17. При отображении документа в браузере специальный символ заменяется на символьную ссылку

Причины экранирования символов

Существует несколько ситуаций, когда может понадобиться (или вы пожелаете) применить ссылку на символ.

Символы синтаксиса HTML

Символы <, >, &, «, и ' имеют в HTML специальный синтаксический смысл, поэтому могут неверно толковаться при использовании в коде. Поэтому консорциум W3C рекомендует экранировать в контенте символы <, > и &. Если значения атрибута содержат одинарные или двойные кавычки, рекомендуется в значениях символов кавычек экранировать. Кавычки, имеющиеся в тексте контента, экранироваться не должны (табл. 5.2).

Таблица 5.2. Символы синтаксиса и их символьные сущности

Символ	Описание	Название сущности	Десятичное значение	Шестнадцатеричное значение
<	Символ «меньше, чем»	<	<	<
>	Символ «больше, чем»	>	>	>
«	Знак цитирования	"	"	"
'	Апостроф	'	'	'
&	Амперсанд	&	&	&

Невидимые или неоднозначные символы

Некоторые символы не имеют графического отображения, и их сложно отследить в разметке (табл. 5.3). В частности, идет речь о неразрывном пробеле: , который служит для того, что строка не разрывалась между двумя словами. Так, например, если разметить мои имя и фамилию таким образом:

Jennifer Robbins

они всегда будут располагаться в одной строке. Другое применение неразрывных пробелов — разделение цифр в длинном числе, например: 32 000 000.

Пробел нулевой ширины можно размещать в языках, не использующих пробелы между словами, где это позволит указать место разрыва строки. *Соединитель нулевой ширины* — это не выводимое на печать пространство, в котором соседние символы отображаются в их связанных формах (обычно в арабском и индийском языках). *Разделители нулевой ширины* препятствуют тому, чтобы соседние символы сливались для образования лигатур или других связанных форм.

Таблица 5.3. Невидимые символы и их символьные ссылки

Символ	Описание	Название сущности	Десятичное значение	Шестнадцатеричное значение
(не выводится на печать)	Неразрывный пробел	 	 	
(не выводится на печать)	Нормальный пробел	 	 	 

Табл. 5.3 (окончание)

Символ	Описание	Название сущности	Десятичное значение	Шестнадцатеричное значение
(не выводится на печать)	Широкий пробел	 	 	 
(не выводится на печать)	Пробел нулевой ширины	(нет)	​	​
(не выводится на печать)	Разделитель нулевой ширины (Zero-width non-joiner)	‌	‌	‌
(не выводится на печать)	Соединитель нулевой ширины (Zero-width joiner)	‍	‍	‍

Ограничения ввода

Если ваша клавиатура или программное обеспечение для редактирования не содержит нужный символ (или вы не можете его найти), можно воспользоваться символьной сущностью, которая эквивалентна этому символу. Консорциум W3C не поддерживает эту практику, поэтому по возможности применяйте в коде соответствующие символы. В табл. 5.4 приведены некоторые специальные символы, которые не столь легко ввести в код.

Таблица 5.4. Специальные символы и их символьные ссылки

Символ	Описание	Название сущности	Десятичное значение	Шестнадцатеричное значение
`	Левая наклонная одинарная кавычка	‘	‘	‘
'	Правая наклонная одинарная кавычка	’	’	’
“	Левая наклонная двойная кавычка	“	“	“
”	Правая наклонная двойная кавычка	”	”	”
...	Горизонтальное многоточие	…	…	…
©	Авторское право	©	©	©
®	Зарегистрированная торговая марка	®	®	®
™	Торговая марка	™	™	…
£	Фунт	£	£	£
¥	Йена	¥	¥	¥
€	Евро	€	€	€
-	Короткое тире	–	–	–
-	Длинное тире	—	—	—

Полный список названий сущностей HTML и их кодов-указателей в Юникоде входит в спецификацию HTML5 и доступен на сайте по следующему адресу: www.w3.org/TR/html5/syntax.html#named-character-references. Более удобный список именованных и числовых сущностей можно найти на заархивированной странице Web Standards Project: www.webstandards.org/learn/reference/charts/entities.

ОБЪЕДИНЯЕМ ВСЕ ВМЕСТЕ

Итак, вы узнали, как размечать элементы, и познакомились со всеми HTML-элементами, применяемыми для структурирования текстового контента. Теперь пора приступить к практике. Упражнение 5.3 позволяет опробовать рассмотренный материал: элементы структуры документа, элементы группировки (блочные), элементы фразы (строчные), элементы секционирования и символные сущности.

УПРАЖНЕНИЕ 5.3. СТРАНИЦА «BISTRO BLACK GOOSE»

Ознакомившись со всеми текстовыми элементами, можно приступать к работе по разметке страницы новостей для сайта «Bistro Black Goose». Обработайте текст упражнения, следя приведенным далее инструкциям (результатирующая страница показана на рис. 5.18). Исходный код и готовые файлы разметки находятся на сайте по следующему адресу: learningwebdesign.com/5e/materials.

The Black Goose Bistro News

Home
Menu
News
Contact

Summer Menu Items posted by BGB, June 18, 2017

Our chef has been busy putting together the perfect menu for the summer months. Stop by to try these appetizers and main courses while the days are still long.

Appetizers

Black bean purses

Spicy black bean and a blend of Mexican cheeses wrapped in sheets of phyllo and baked until golden. \$3.95

Southwestern napoleons with lump crab -- new item!

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. \$7.95

Main courses

Shrimp sate kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. \$12.95

Jerk rotisserie chicken with fried plantains -- new item!
Tender chicken slow-roasted on the rotisserie, flavored with
spicy and fragrant jerk sauce and served with fried plantains
and fresh mango. \$12.95

Low and Slow

posted by BGB, November 15, 2016

<p>This week I am
extremely excited about
a new cooking technique called
<dfn><i>sous vide</i></dfn>. In <i>sous vide</i>
cooking, you submerge the food (usually vacuum-sealed in
plastic) into a water bath that is precisely set to the target
temperature you want the food to be cooked to. In his book,
<cite>Cooking for Geeks</cite>, Jeff Potter describes it as
<q>ultra-low-temperature poaching.</q></p>

<p>Next month, we will be serving <i>Sous Vide</i> Salmon
with Dill Hollandaise. To reserve a seat at the chef table,
contact us before <time datetime="20161130">November 30</time>.
</p>

Location: Baker's Corner, Seekonk, MA

Hours: Tuesday to Saturday, 11am to 11pm

All content copyright 2017, Black Goose Bistro and Jennifer
Robbins

1. Начните с добавления объявления DOCTYPE, которое сообщит браузерам о том, что перед ними документ HTML5.
2. Добавьте сначала все элементы, относящиеся к структуре документа: html, head, meta, title и body. Озаглавьте документ так «The Black Goose Bistro News».
3. Сначала определим заголовок верхнего уровня и список ссылок в заглавии документа, помещая их в элемент header (не забудьте указать закрывающий тег). Первым элементом заголовка является h1, а списком ссылок служит неупорядоченный список: ul. Не беспокойтесь сейчас о формировании ссылок на элементы списка — этот вопрос рассматривается в следующей главе. Придайте списку больше смысла, указав его в качестве элемента основной навигации по сайту: nav.
4. Страница News содержит два сообщения, озаглавленные «Summer Menu Items» и «Low and Slow». Разметьте каждый из них как раздел.
5. Теперь оформим первый раздел. Сформируем заголовочную часть для раздела, куда входит заголовок (на этот раз помеченный элементом h2, поскольку

ПРИМЕЧАНИЕ

Абзац «Low and Slow» здесь уже размещен с помощью строчных элементов из упражнения 5.2.

СОВЕТ ПО ВЫПОЛНЕНИЮ РАЗМЕТКИ

Последовательное выравнивание каждого иерархического уровня в HTML-коде облегчает сканирование и последующее обновление документа.

мы переместились по иерархии документа) и информация о публикации: *p*. Определим дату публикации раздела с помощью элемента *time*, как было продемонстрировано в упражнении 5.2.

6. Контент, приведенный после заголовка, является простым абзацем. Однако в меню появились интересные новшества. Оно разделено на два концептуальных раздела (*Appetizers* и *Main Courses*), поэтому пометьте их с помощью элементов *section*. Обратите внимание, чтобы заключительный тег раздела (*</section>*) отображался перед закрывающим тегом статьи (*</article>*) — тогда элементы будут вложены правильно и не станут перекрываться. Наконец, идентифицируем разделы с помощью атрибутов *id*. Назовите первый «*appetizers*», а второй — «*maincourses*».
7. После оформления разделов можно разметить контент. Для заголовков в каждом разделе выбираем теперь элемент *h3*. Для названия пунктов меню и их описания выберите подходящие элементы списка. Разметьте списки и каждый пункт в списках.
8. А теперь добавим несколько мелких деталей — *классифицируйте* каждую цену как «*price*», используя элементы *span*.
9. Два блюда — новинки. Измените двойные дефисы на символы длинного тире и отметьте «*new item!*» как «*strongly important*». Классифицируйте заголовок каждого нового блюда как «*newitem*» (используйте имеющийся элемент *dt*, не обязательно добавлять элемент *span*). Тогда можно выбирать заголовки меню с помощью класса «*newitem*» и стилизовать их иначе, чем другие пункты меню.
10. Обратите внимание на первый раздел. Второй раздел уже, в основном, размечен при выполнении предыдущего упражнения, но нужно пометить заголовок с помощью соответствующего заголовка и внести информацию о дате публикации.
11. Все хорошо, не так ли? Теперь переместите в раздел *footer* относящуюся ко всей странице остальную часть контента. Пометьте в нижнем колонтитуле каждую строку контента как абзац.
12. Внесем информацию о местоположении и времени в некотором контексте, поместив их в элемент *div* с именем «*about*». Добьемся того, чтобы метки «*Location*» и «*Hours*» сами отображались в строке, добавляя после них разрывы строк. Разметьте время с помощью элемента *time* (нет необходимости указывать уточнения, относящиеся к пояснному времени).
13. Наконец, информация об авторских правах. Как правило, она печатается в документе мелким шрифтом, поэтому соответствующим образом пометьте ее. И, как последний штрих, с помощью клавиатуры добавьте символ авторского права после слова «*copyright*» или символьную сущность *©*.

Сохраните файл под именем **bistro_news.html** и просмотрите страницу в современном браузере. Также можете загрузить страницу в validator.nu и убедиться, что он валиден (кстати, отличный способ обнаружить ошибки). Как видите, у вас все получилось!

The Black Goose Bistro News

- Home
- Menu
- News
- Contact

Summer Menu Items

posted by BGB, June 18, 2017

Our chef has been busy putting together the perfect menu for the summer months. Stop by to try these appetizers and main courses while the days are still long.

Appetizers

Black bean purses

Spicy black bean and a blend of Mexican cheeses wrapped in sheets of phyl-lo and baked until golden. \$3.95

Southwestern napoleons with lump crab — new item!

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. \$7.95

Main courses

Shrimp saté kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. \$12.95

Jerk rotisserie chicken with fried plantains — new item!

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. \$12.95

Low and Slow

posted by BGB, November 15, 2016

This week I am *extremely* excited about a new cooking technique called *sous vide*. In *sous vide* cooking, you submerge the food (usually vacuum-sealed in plastic) into a water bath that is precisely set to the target temperature you want the food to be cooked to. In his book, *Cooking for Geeks*, Jeff Potter describes it as "ultra-low-temperature poaching."

Next month, we will be serving *Sous Vide Salmon with Dill Hollandaise*. To reserve a seat at the chef table, contact us before November 30.

Location:

Baker's Corner, Seekonk, MA

Hours:

Tuesday to Saturday, 11am to 11pm

All content copyright © 2017, Black Goose Bistro and Jennifer Robbins

Рис. 5.18. Результирующая страница меню

СОВЕТЫ ПО ВЫПОЛНЕНИЮ РАЗМЕТКИ

- Выберите элемент, который наиболее соответствует смыслу выделенного текста.
- Не забывайте закрывать элементы с помощью закрывающих тегов.
- Поместите для ясности все значения атрибутов в кавычки.
- Применяйте способ копирования и вставки при добавлении одной и той же разметки к нескольким элементам. Просто убедитесь, что скопировано верно, прежде чем вставлять ее по всему документу.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Проверьте, достаточно ли вы внимательны. Далее предлагается следующий набор вопросов. Ответы на эти вопросы находятся в *приложении 1*.

- Добавьте разметку для вставки тематического разрыва между этими абзацами:

```
<p>People who know me know that I love to cook.</p>
<p>I've created this site to share some of my favorite
recipes.</p>
```
- В чем различие между элементами `blockquote` и `q`?
- Какой элемент отображает пробелы именно в том виде, как они введены в исходном документе?
- В чем различие между элементами `ul` и `ol`?
- Как из неупорядоченного списка удаляются маркеры? (В самом общем случае, без конкретики.)
- Какой элемент следует применить для разметки «W3C» и указания полного названия (World Wide Web Consortium)? Можете ли записать полную разметку?
- В чем различие между элементами `dl` и `dt`?
- В чем различие между элементами `id` и `class`?
- В чем различие между элементами `article` и `section`?

ТРЕБУЕТСЯ БОЛЬШЕ ПРАКТИКИ?

Разметьте свое резюме. Начните с исходного текста и внесите элементы структуры документа, элементы группировки контента и строчные элементы, как это было сделано в *упражнении 5.3*. Если не нашли элемент, соответствующий вашей информации, создайте его, применяя элементы `div` или `span`.

ОБЗОР ЭЛЕМЕНТОВ: ТЕКСТОВЫЕ ЭЛЕМЕНТЫ

В табл. 5.5 приведен перечень элементов, рассмотренных при изучении текстовой разметки. Ко всем текстовым элементам применяются глобальные атрибуты. Дополнительные атрибуты приведены здесь под соответствующими элементами.

Таблица 5.5. Перечень элементов, рассмотренных при изучении текстовой разметки

Элементы и атрибуты	Описание
Разделы страницы	
<code>address</code>	Контактная информация автора
<code>article</code>	Автономный контент
<code>aside</code>	Тангенциальный контент (врезка)

Табл. 5.5 (продолжение)

Элементы и атрибуты	Описание
footer	Связанный контент
header	Вводный контент
nav	Основная навигация
section	Концептуально связанная группа контента
Контент заголовка	
h1...h6	Заголовки, уровни с 1 по 6
Группировка элементов контента и атрибутов	
blockquote	Длинная цитата
cite	URL цитируемого контента
div	Обобщенное подразделение
figure	Связанное изображение или ресурс
figcaption	Текстовое описание фигуры
hr	Тематический разрыв на уровне абзаца (горизонтальная линейка)
main	Основная область контента страницы или приложения
p	Абзац
pre	Предварительно отформатированный текст
Элементы и атрибуты списка	
dd	Определение
dl	Список определений
dt	Термин
li	Пункт списка (для ul и ol)
value	Предоставляет значение для li в ol
ol	Упорядоченный список
reversed	Нумерация списка в обратном порядке
start	Предоставляет начальный номер для списка
ul	Неупорядоченный список
Разрывы	
br	Разрыв строки
wbr	Разрыв слова
Элементы и атрибуты для обработки строк	
abbr	Сокращение
b	Добавлено визуальное выделение (полужирный шрифт)
bdi	Двунаправленная изоляция
bdo	Двунаправленное переопределение
cite	Цитирование

Табл. 5.5 (окончание)

Элементы и атрибуты	Описание
code	Пример кода
data	Машиночитаемый эквивалент
del	Удаленный текст
cite	URL цитируемого контента
datetime	Определяет дату и время изменения
dfn	Определение термина
em	Смыслоное выделение (курсив)
i	Альтернативное значение (курсив)
cite	URL цитируемого контента
datetime	Определяет дату и время изменения
kbd	Ввод с клавиатуры
mark	Выделенный текст
q	Короткая текстовая цитата
cite	URL цитируемого контента
ruby	Раздел, содержащий текст ruby
rp	Скобки в тексте ruby
rt	Аннотация ruby
s	Зачеркнутый текст; неправильный текст
samp	Образец вывода
small	Аннотация; «мелкий шрифт»
span	Обобщенная текстовая фраза
strong	Сильное выделение
sub	Нижний индекс
sup	Верхний индекс
time	Машиночитаемые данные о времени
datetime	Предоставляет машиночитаемую дату/время
pubdate	Указывает время, которое относится к публикации
u	Дополнительное внимание (подчеркивание)

ГЛАВА 6

ДОБАВЛЕНИЕ ССЫЛОК

В этой главе...

- ▶ *Ссылки на внешние страницы*
- ▶ *Ссылки на документы, находящиеся на сервере*
- ▶ *Ссылка на определенную область страницы*
- ▶ *Открытие новых целевых окон браузера*

При создании веб-страницы, скорее всего, будете ссылаться на другие веб-страницы и ресурсы, находящиеся на вашем собственном сайте или на каком-либо другом. В конце концов, Интернет — это и есть совокупность ссылок. В этой главе мы рассмотрим разметку, применяемую для создания ссылок. Эти ссылки предназначены для установления связей с другими сайтами, с собственным сайтом и внутри веб-страницы. Для создания ссылок используется единственный элемент якоря: а:

`<a>...`

Элемент якоря (гипертекстовая ссылка)

Для преобразования выделенного текста в ссылку заключите его в закрывающие и открывающие теги: `<a>...` и воспользуйтесь атрибутом `href` для указания интернет-адреса (URL) целевой страницы. Контент элемента якоря становится гипертекстовой ссылкой. Вот пример создания ссылки на сайт O'Reilly Media:

`Go to the O'Reilly Media site`

Для создания ссылки на изображение в элемент якоря включите элемент `img`:

``

Обратите внимание, что для создания ссылки в элемент якоря можно включить любой элемент контента HTML, а не только изображения.

Практически все графические браузеры отображают текст ссылки синим цветом и подчеркивают его по умолчанию. Некоторые устаревшие браузеры заключают изображения, используемые в качестве ссылок, в синюю рамку, но большинство современных браузеров этого не делают. Посещенные ссылки обычно отображаются фиолетовым цветом. Пользователи могут изменять эти цвета в настройках браузера, и, конечно, можно изменять внешний вид ссылок для сайтов с помощью таблиц стилей (эта тема подробнее будет рассмотрена в главе 13).

НЕ НАРУШАЙТЕ ПРИНЦИПОВ ДОСТУПНОСТИ

Будьте внимательными — если вы изменяете цвета ссылок, сохраняйте единообразие на всем сайте, чтобы не создавать неудобства пользователям.

После щелчка (или касания, если речь идет о сенсорном экране) пользователя на тексте или изображении, указанном в элементе якоря, соответствующая страница загружается в окно браузера (на рис. 6.1 показан результат выполнения приведенного ранее примера разметки связанного изображения).

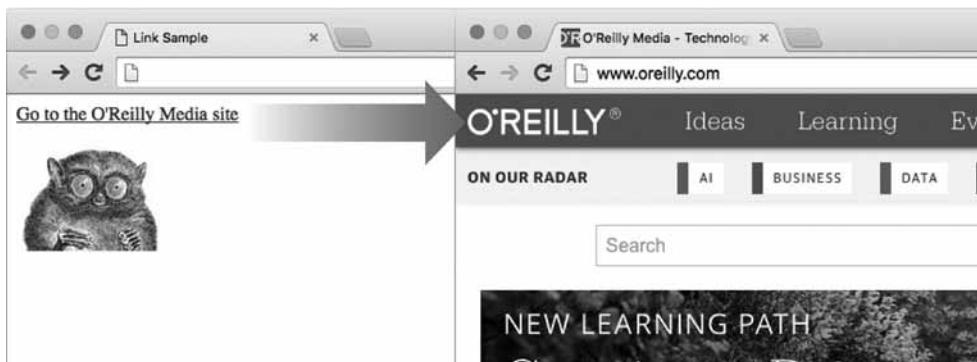


Рис. 6.1. После щелчка (или касания, если речь идет о сенсорном экране) пользователя на связанном тексте или изображении указанная в элементе якоря соответствующая страница загружается в окне браузера.

АТРИБУТ *Href*

При создании ссылок следует указать браузеру, с каким документом связана ссылка, не так ли? Для этого используется атрибут *href*, название которого образовано от слов hypertext reference, гипертекстовая ссылка. Этот атрибут показывает браузеру интернет-адрес страницы или источника (соответствующий URL), который

СТРУКТУРА ЯКОРЯ

Упрощенная структура элемента якоря выглядит следующим образом:

```
<a href="url">связанный контент</a>
```

заключается в кавычки. Чаще всего так указывают на другие HTML-документы, однако можно указывать и на иные веб-ресурсы — например, на изображения, аудио- и видеофайлы.

На практике используются два способа указания URL:

- *абсолютные адреса* — абсолютный URL предоставляет собой полный интернет-адрес документа, включая протокол (`http://` или `https://`), имя домена и, при необходимости, путь. Абсолютный URL следует использовать при указании ссылки на документ, находящийся в Интернете (т. е. не на вашем собственном сервере):

```
href="http://www.oreilly.com/"
```

Иногда, если страница, на которую вы ссылаетесь, имеет длинный URL, ссылка может оказаться весьма громоздкой (рис. 6.2). Следует иметь в виду, что структура, формируемая элементом якоря, является простым контейнерным элементом с одним атрибутом. И пусть вас не пугают длинные пути, указанные в ссылке.

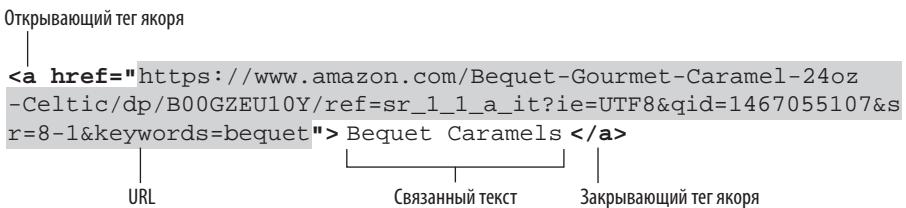


Рис. 6.2. Пример длинного интернет-адреса (URL). И хотя тег якоря выглядит сложным, его структура довольно-таки проста

- *относительные адреса* — относительные URL описывают путь к файлу *относительно* текущего документа. Относительные адреса могут применяться при ссылке на другой документ, находящийся на вашем собственном сайте (т. е. на том же сервере). Тогда не требуется указывать имя протокола или домена, а достаточно лишь указать путь:

```
href="recipes/index.html"
```

КАК ИЗБЕЖАТЬ ОПЕЧАТОК ПРИ НАПИСАНИИ ИНТЕРНЕТ-АДРЕСА?

Если создается ссылка на страницу с длинным интернет-адресом (URL), удобно скопировать адрес этой страницы из адресной строки браузера, после чего и вставить его в документ. Таким способом можно избежать опечаток при вводе символов, да и нарушения структуры самой ссылки.

В этой главе мы добавим ссылки с использованием и абсолютных, и относительных адресов на кулинарный сайт автора — Jen's Kitchen (рис. 6.3). Абсолютные адреса весьма прости, поэтому сначала разберемся с ними.



Рис. 6.3. Веб-страница Jen's Kitchen

ССЫЛКИ НА ВЕБ-СТРАНИЦЫ В ИНТЕРНЕТЕ

У пользователей зачастую возникает желание создать ссылку на обнаруженную в Интернете страницу. В этом случае идет речь о *внешней ссылке*, поскольку она относится к странице, находящейся за пределами вашего сервера или сайта. Для создания внешней ссылки укажите абсолютный интернет-адрес (URL), начиная с имени протокола: `http://`. Встретив такую ссылку, браузер получит указание: «Подключись к Интернету и получи следующий документ».

А теперь добавим на начальную страницу сайта Jen's Kitchen (см. рис. 6.3) несколько внешних ссылок. Во-первых, пункт списка «The Food Network» ссылается на сайт **www.foodnetwork.com**. Текст ссылки размечается с помощью элемента якоря, и добавляются открывающие и закрывающие теги якоря:

```
<li><a>The Food Network</a></li>
```

Обратите внимание, что теги якоря добавляются в элемент списка: `li`. Причина этого заключается в том, что только элементы `li` являются дочерними элементами для элемента `ul`, размещение элемента `a` непосредственно внутри элемента `ul` приведет к некорректной HTML-разметке.

Затем я добавляю атрибут `href` с полным URL для сайта:

```
<li><a href="http://www.foodnetwork.com">The Food Network</a></li>
```

Все в порядке! Теперь текст **The Food Network** имеет вид ссылки и перенаправляет по ней посетителей моей страницы на сайт **http://www.foodnetwork.com** после щелчка на ссылке или касания ее. Выполните только что описанные действия самостоятельно в *упражнении 6.1*.

РАБОТАЕМ ВМЕСТЕ С ДЖЕН

Все файлы для сайта Jen's Kitchen (см. рис. 6.3) доступны в Интернете по адресу learningwebdesign.com/5e/materials. Загрузите весь каталог, но проследите, чтобы организация контента не изменилась. Этот контент может быть не слишком интересным для вас, но зато даст возможность приобрести навыки по созданию ссылок между страницами.

На этом сайте также доступна результирующая разметка для всех упражнений.

УПРАЖНЕНИЕ 6.1. СОЗДАНИЕ ВНЕШНЕЙ ССЫЛКИ

Откройте файл `index.html`, находящийся в папке `jenskitchen`. Создайте ссылку для пункта списка «Epicurious», ведущую на сайт www.epicurious.com. При этом используйте в качестве примера ссылку, созданную ранее для сайта **The Food Network**:

```
<ul>
<li><a href="http://www.foodnetwork.com/">The Food Network
</a></li>
<li>Epicurious</li></ul>
```

После завершения создания ссылки сохраните файл index.html и откройте его в браузере. Если имеется подключение к Интернету, щелкните на только что созданной ссылке, чтобы перейти на сайт **Epicurious**. Если ссылка не приведет на этот сайт, вернитесь обратно и удостоверьтесь, что ничего не пропущено в разметке.

ПРЕДОСТЕРЕЖЕНИЯ ПРИ СОЗДАНИИ ИМЕН ПУТЕЙ

При написании имен относительных путей следуйте приведенным правилам, что позволит избежать ряда распространенных ошибок:

- не применяйте обратные слэши: \. При указании путей для интернет-адресов применяются только прямые слэши: /.
- не указывайте в начале имени название диска (D:, C: и т. п.). И хотя страницы, находящиеся на вашем компьютере, успешно поддерживают подобные ссылки, после загрузки страницы на веб-сервер имя диска уже не будет иметь значения и может привести к нарушению корректности ссылок;
- Не указывайте в начале file:///. Префикс file:// свидетельствует о том, что файл расположен локально, что приводит к разрыву связи, если файл располагается на сервере.

ССЫЛКИ В ПРЕДЕЛАХ САЙТА

На самом деле львиная доля ссылок, как правило, относится к страницам вашего собственного сайта: от начальной страницы к страницам разделов, от страниц разделов к страницам контента и т. д. При этом вполне можно пользоваться относительными интернет-адресами, которые вызывают страницы на вашем собственном сервере.

Если не указан префикс http://, браузер ищет связанный документ на текущем сервере. *Имя пути* — обозначение, используемое для указания на конкретный файл или каталог, сообщает браузеру о местонахождении файла. Имена путей к веб-страницам создаются с учетом соглашений UNIX: имена каталогов и имена файлов разделяются обычными слэшами: /. Относительный путь описывает, как добраться до связанного документа, начиная от местоположения текущего документа.

Относительные пути могут иметь непривычный вид. По моему опыту, именно создание относительных путей вызывает серьезные затруднения у новичков, поэтому этот процесс мы рассмотрим поэтапно. Желательно, чтобы *упражнения с 6.2 по 6.8* выполнялись по мере чтения книги.

Все относящиеся к путям примеры в этом разделе основаны на структуре сайта Jen's Kitchen (рис. 6.4). Иллюстрируя структуру каталогов сайта, обычно изображают перевернутое дерево с корневым каталогом в верхней части

ПАПКИ ИЛИ КАТАЛОГИ?

Пользователям ПК и Mac привычнее система наименований, когда файлы организуются в папки, но в мире веб-разработки чаще используется эквивалентный и более технический термин каталог. Папка — это тот же каталог с симпатичной пиктограммой.

иерархии. Для сайта Jen's Kitchen корневым каталогом является **jenskitchen**. Чтобы вам было понятнее, на рис. 6.4 показана также структура этого каталога и его подкаталогов, которые отображаются в Finder на моем Mac.

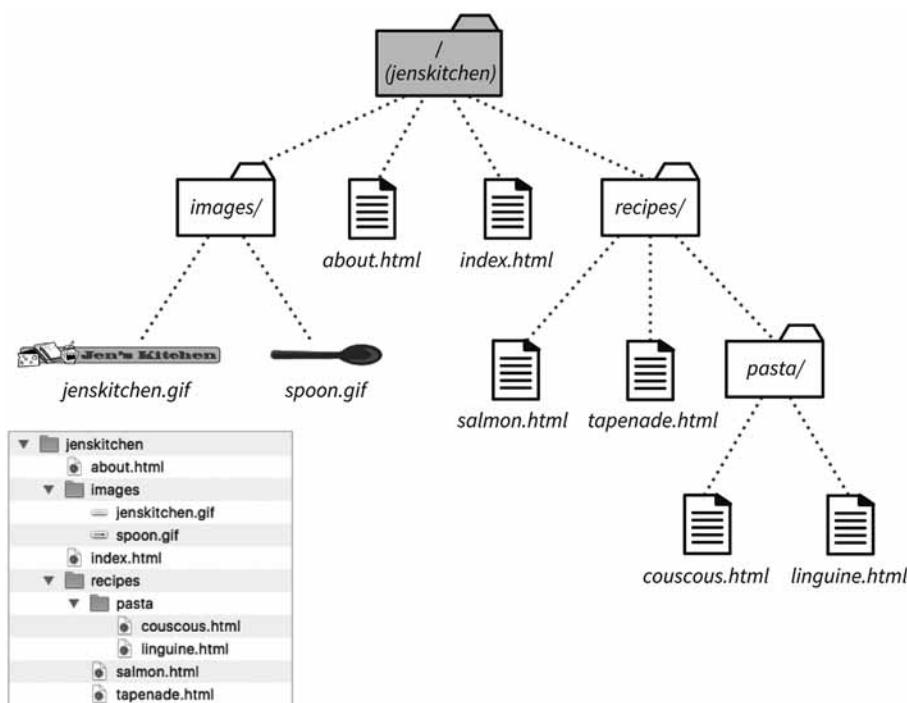


Рис. 6.4. Структура сайта **jenskitchen**

Создание ссылок внутри каталога

Самый простой относительный URL указывает на другой файл, находящийся в том же каталоге. В ссылке на файл, который находится в том же каталоге, следует указывать лишь имя файла. Если URL представлен просто именем файла, сервер ищет файл в текущем каталоге (то есть в каталоге, содержащем связанный документ).

ССЫЛКА НА ИМЯ ФАЙЛА

Ссылка на имя файла указывает, что связанный файл находится в том же каталоге, что и текущий документ.

В следующем примере мы создадим ссылку с начальной страницы (**index.html**) на страницу общей информации (**about.html**). Оба файла находятся в одном каталоге (**jenskitchen**), поэтому с начальной страницы можно создать ссылку на информационную страницу, просто указав в качестве URL имя файла этой страницы (рис. 6.5):

```
<a href="about.html">About the site...</a>
```

Упражнение 6.2 даст вам возможность самостоятельно разметить простую ссылку.

Документы **index.html** и **about.html** находятся в том же каталоге

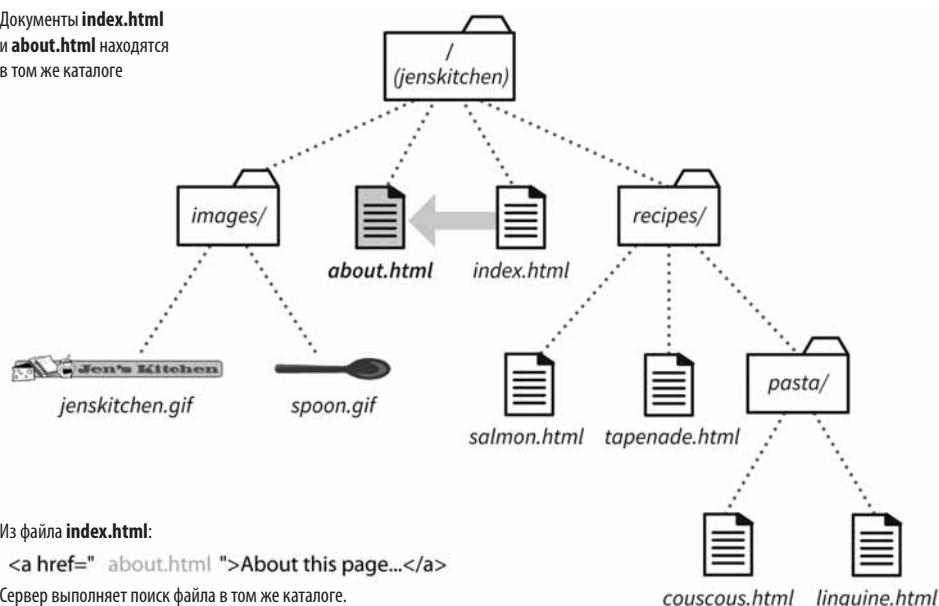


Рис. 6.5. Создание относительной ссылки на документ, находящийся в том же каталоге, что и сам исходный документ

УПРАЖНЕНИЕ 6.2. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛЫ, НАХОДЯЩИЕСЯ В ОДНОМ И ТОМ ЖЕ КАТАЛОГЕ

Откройте файл **about.html**, находящийся в папке **jenskitchen**. В нижней части страницы создайте абзац «Back to the home page», который будет служить ссылкой на начальную страницу **index.html**. Самостоятельно создайте саму ссылку. При этом учтите, что элемент якоря **a** обязательно включается в элемент **p**:

```
<p>Back to the home page</p>
```

После завершения создания ссылки сохраните файл **about.html** и откройте его в браузере. Для проверки локальных ссылок (то есть ссылок на файлы вашего компьютера) подключение к Интернету не нужно. После щелчка на такой ссылке вы вернетесь на начальную страницу.

Ссылки на каталог нижнего уровня

А как быть, если файлы не находятся в одном каталоге? В этом случае нужно создать соответствующие указания для браузера, а также указать путь URL для ссылки. Рассмотрим, как это сделать. Обратимся к примеру (см. рис. 6.4), где файлы моих рецептов сохраняются в подкаталоге, который называется **recipes**, и создадим ссылку из файла **index.html** на файл **salmon.html**, находящийся в каталоге **recipes**. Путь в адресе должен указать браузеру, что следует найти в текущем каталоге каталог с именем **recipes**, а затем найти в нем файл **salmon.html** (рис. 6.6).

Файл **salmon.html** находится в каталоге, размещенном на один уровень ниже каталога файла **index.html**.



Рис. 6.6. Создание относительной ссылки на документ, который находится на один уровень каталога ниже по сравнению с исходным документом

В процессе выполнения *упражнения 6.3* мы создадим ссылку на файл, находящийся в другом каталоге.

УПРАЖНЕНИЕ 6.3. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛ, НАХОДЯЩИЙСЯ В ДРУГОМ КАТАЛОГЕ

Откройте файл **index.html**, находящийся в папке **jenskitchen**. Создайте ссылку, которая связывает пункт списка **Tapenade (Olive Spread)** (см. рис. 6.3) с файлом **tapenade.html**, находящемся в каталоге **recipes**. Не забывайте корректно выполнять вложения элементов:

```
<li>Tapenade (Olive Spread)</li>
```

После завершения создания ссылки сохраните файл **index.html** и откройте его в браузере. Чтобы попасть на страницу рецепта **Tapenade (Olive Spread)**, нужно щелкнуть на только что созданной ссылке. Если перейти на эту страницу не получилось, убедитесь в том, что разметка выполнена верно, а структура каталога **jenskitchen** соответствует структуре, используемой для примеров книги.

Создадим ссылку на файл **couscous.html**, который находится в подкаталоге **pasta**. Для этого нужно указать переход через два подкаталога (**recipes**, потом **pasta**) к файлу **couscous.html** (рис. 6.7):

```
<li><a href="recipes/pasta/couscous.html">Couscous...</a></li>
```

Каталоги, как можно видеть, разделяются обычными слэшами. Получившийся тег якоря предписывает браузеру следующее: «Найди в текущем каталоге каталог под названием **recipes**. В нем найди каталог с именем **pasta** и в нем — файл **couscous.html**».

Файл `couscous.html` находится в иерархии каталогов на два уровня ниже, чем файл `index.html`.



Из файла `index.html`:

```
<a href=" recipes/pasta/couscous.html " >Couscous</a>
```

Сервер выполняет поиск в том же каталоге, где находится текущий документ (в каталоге `recipes`), затем ищет каталог `pasta`.

Рис. 6.7. Создание относительной ссылки на документ, который находится в каталоге, расположеннем на два уровня ниже, чем исходный документ. Создайте такую ссылку самостоятельно при выполнении упражнения 6.4

Создав ссылку на каталоги, находящиеся на двух разных уровнях, уточним, как вообще создаются пути при переходе через произвольное число каталогов. А создаются они аналогично: начинайте с имени каталога, который находится в том же месте, где находится текущий файл, и указывайте после каждого имени каталога слэш, пока не дойдете до имени файла, на которыйсылаетесь.

ССЫЛКА НА ФАЙЛ ИЗ КАТАЛОГА НИЖНЕГО УРОВНЯ

Если высылаетесь на файл, находящийся в каталоге нижнего уровня, имя пути содержит имена каждого подкаталога, через который необходимо пройти для перехода к файлу.

УПРАЖНЕНИЕ 6.4. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛ, НАХОДЯЩИЙСЯ В КАТАЛОГЕ, РАСПОЛОЖЕННОМ НА ДВА УРОВНЯ НИЖЕ

Откройте файл `index.html`, находящийся в папке `jenskitchen`. Создайте ссылку, связывающую пункт меню **Linguine with Clam Sauce** (см. рис. 6.3) с файлом `linguine.html`, находящимся в каталоге `pasta`:

```
<li>Linguine with Clam Sauce</li>
```

Создав ссылку, сохраните файл `index.html` и откройте его в окне браузера. Щелкните на новой ссылке, чтобы ознакомиться с хорошим рецептом.

Создание ссылки на каталог, находящийся на более высоком уровне

До сих пор все получалось, не так ли? Теперь рассмотрим более сложные задачи. А именно, создадим ссылку, связывающую страницу рецепта приготовления лосося

(*salmon recipe*), с начальной страницей, которая находится на один уровень выше в иерархии каталогов.

В этом случае в UNIX используется соглашение об именовании путей в виде «точка-точка-слэш»: `..`. Если путь начинается с `..`, то это равносильно тому, чтобы сказать браузеру: «вернуться обратно вверх на один уровень каталога», а затем следовать по пути к указанному файлу. Если вы знакомы с порядком просмотра файлов на компьютере, полезно знать, что `..` вызывает то же действие, что и щелчок на кнопке **Вверх** (Up) в Проводнике Windows (Windows Explorer) или на кнопке стрелки влево при использовании Finder в macOS.

Сначала создадим ссылку, связывающую файл **salmon.html** с начальной страницей **index.html**. Поскольку файл **salmon.html** находится в подкаталоге **recipes**, нужно вернуться в каталог **jenskitchen**, чтобы найти там файл **index.html**. В соответствии с этим рассуждением следующий путь указывает браузеру «вернуться на один уровень в иерархии каталогов», а затем искать в этом каталоге файл **index.html** (рис. 6.8):

```
<p><a href="..>/index.html">[Back to home page]</a></p>
```

Обратите внимание, что группа символов `..` заменяет имя каталога, находящегося на более высоком уровне, и нет необходимости в имени пути добавлять каталог **jenskitchen**.

Файл **index.html** находится в иерархии каталогов на один уровень выше, чем файл **salmon.html**.

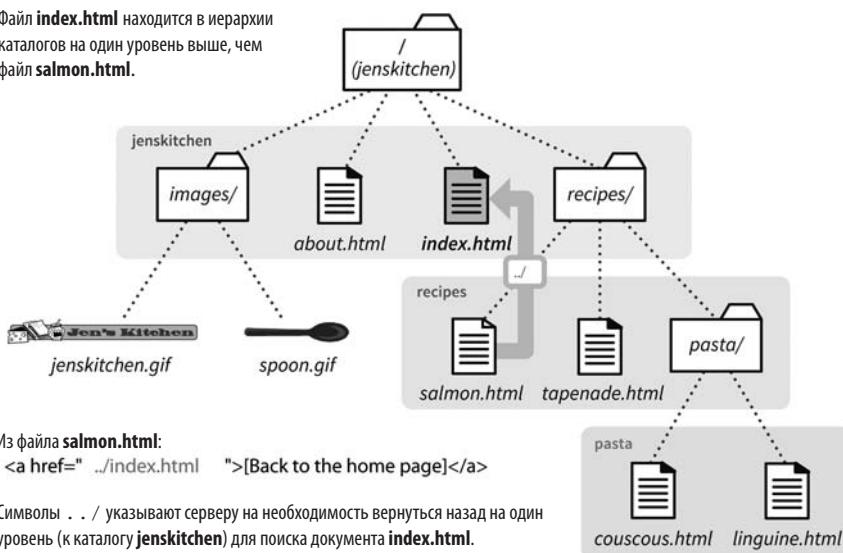


Рис. 6.8. Создание относительной ссылки на документ, который находится в иерархии каталогов на один уровень выше, чем исходный документ

Попробуйте при выполнении приведенного далее *упражнения 6.5* добавить имя пути в формате «точка-точка-слэш» `..` для перехода к каталогу, находящемуся на более высоком уровне.

Обратите внимание на ссылку, которая ведет от файла **couscous.html** на начальную страницу. Догадались уже, как осуществляется возврат на два уровня каталогов? Все

весьма просто: в ссылку добавляются две группы символов в формате «точка-точка-слэш» (рис. 6.9).

При этом ссылка на странице `couscous.html` — то есть возвращение к начальной странице (`index.html`) — примет следующий вид:

```
<p><a href=". . ./index.html">[Back to home page]</a></p>
```

Первая группа символов `. . /` обеспечивает возврат к каталогу рецептов `recipes`, вторая — возвращает к каталогу верхнего уровня (`jenskitchen`), где и может быть найден файл `index.html`. И снова: нет необходимости писать имена каталогов — эту функцию выполняют группы символов `. . /`.

Файл `index.html` находится на два уровня в иерархии каталогов выше, чем файл `couscous.html`.

ВНИМАНИЕ!

Каждая группа символов `. . /` в начале пути указывает браузеру на необходимость перехода на один уровень вверх.

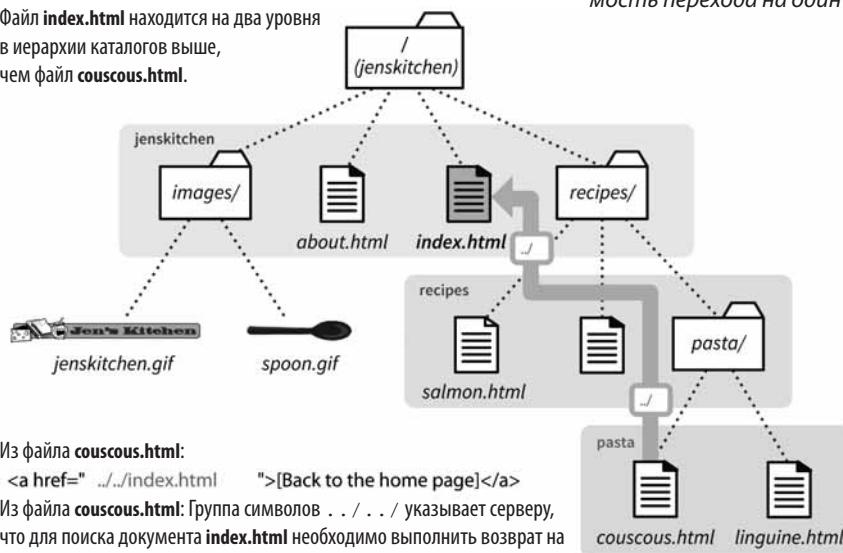


Рис. 6.9. Создание относительной ссылки на документ, который находится в каталоге, расположенному на два уровня выше, чем исходный документ

А теперь выполним упражнения 6.5 и 6.6.

УПРАЖНЕНИЕ 6.5. ССЫЛКА НА КАТАЛОГ, НАХОДЯЩИЙСЯ НА БОЛЕЕ ВЫСОКОМ УРОВНЕ

Откройте файл `tapenade.html`, находящийся в каталоге `recipes`. В нижней части страницы можно найти абзац:

```
<p>[Back to the home page]</p>
```

Применяя описанный в этом разделе формат записи, создайте текстовую ссылку, которая ведет обратно — к начальной странице `index.html`, находящейся на один уровень выше в иерархии каталогов.

УПРАЖНЕНИЕ 6.6. СОЗДАНИЕ ССЫЛКИ НА ФАЙЛ, НАХОДЯЩИЙСЯ НА ДВА УРОВНЯ ВЫШЕ В ИЕРАРХИИ КАТАЛОГОВ

Отлично, теперь попробуйте создать еще одну ссылку. Откройте файл linguine.html и создайте для последнего абзаца ссылку, ведущую обратно — к начальной странице. В процессе создания ссылки применяйте группы символов `.. / .. /` для выполнения возврата на два уровня:

```
<p>[Back to the home page]</p>
```

ИЗ ЛИЧНОГО ОПЫТА...

Признаюсь, что при создании сложной относительной ссылки иногда про себя повторяю «поднимаюсь на уровень, поднимаюсь на уровень» для каждой группы символов `.. / .. /`. И это помогает в работе.

После завершения создания ссылки сохраните файл и откройте его в браузере. У вас должна появиться возможность попадать по ссылке на начальную страницу.

Ссылки, создаваемые с помощью путей, ведущих от корневого каталога сайта

Все сайты имеют *корневой каталог*, то есть каталог, содержащий все каталоги и файлы сайта. Все рассмотренные до сих пор пути к документу вели нас по ссылкам через каталоги вверх или вниз. Иной способ создания пути к файлу заключается в том, чтобы начать с корневого каталога и записать на пути к этому файлу имена всех встречающихся подкаталогов. Такой подход к созданию пути известен как адресация *относительно корневого каталога сайта*.

В соглашении об именах путей UNIX обычная косая черта (слэш) `/`, указанная в начале пути, свидетельствует о том, что путь начинается в корневом каталоге. Показанный в следующей ссылке путь, ведущий к файлу через корневой каталог сайта, гласит: «Перейдите в каталог самого верхнего для этого сайта уровня, откройте каталог `recipes` и найдите файл `salmon.html`» (рис. 6.10):

```
<a href="/recipes/salmon.html">Garlic Salmon</a>
```

Обратите внимание, что не нужно добавлять в название пути имя корневого каталога (**jenskitchen**) — обычный слэш `/`, указанный в начале имени пути, представляет каталог верхнего уровня. А далее просто указывайте каталоги, в которых браузер должен искать нужный вам файл.

ССЫЛКИ ОТНОСИТЕЛЬНО КОРНЕВОГО КАТАЛОГА

Ссылки относительно корневого каталога сайта, как правило, предпочтительны в использовании вследствие разнообразия возможностей их применения.

Ссылки относительно корневого каталога сайта удобно применять для контента, который может и не находиться в одном каталоге, а также — для динамически

Поскольку подобный тип ссылки описывает путь относительно корневого каталога сайта, он реально применим для любого находящегося на сервере документа, независимо от того, в каком подкаталоге этот документ расположен.

В именах путей корневой каталог всегда обозначается косой чертой: /, но не именем каталога.

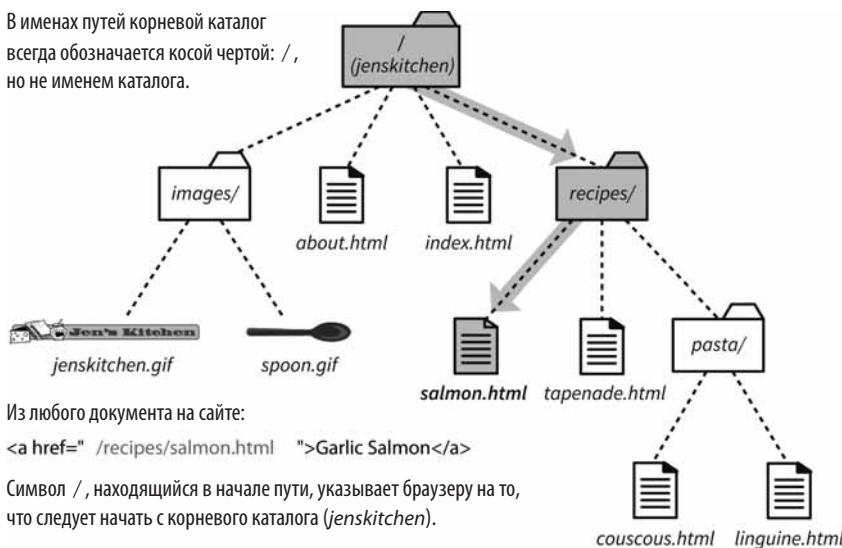


Рис. 6.10. Создание ссылки относительно корневого каталога

генерируемого материала. Использование ссылок относительно корневого каталога облегчает копирование и вставку ссылок между документами.

С другой стороны, ссылки не будут работать на вашем локальном компьютере, поскольку они не включают название дисков компьютера. Для проверки функциональности ссылок нужно будет подождать до тех пор, пока сайт не окажется опубликованным на финальном сервере.

СОСТОРОЖНОСТЬЮ ИСПОЛЬЗУЙТЕ ОТНОСИТЕЛЬНЫЕ ССЫЛКИ

Относительные ссылки на корневой каталог сайта не будут поддерживаться на вашем локальном компьютере, если он не настроен как сервер.

Создание имен путей к изображениям

Атрибут `src` элемента `img` функционирует так же, как атрибут `href` в элементе якоря. Поскольку, скорее всего, будут использоваться изображения, полученные с собственного сервера, значения атрибутов `src` для элементов `image` присваиваются с применением относительных интернет-адресов (URL).

Рассмотрим несколько примеров, взятых с сайта Jen's Kitchen. Во-первых, для добавления изображения на страницу `index.html` нужно воспользоваться следующей разметкой:

```

```

Этот запись предписывает следующее: «Найдите в текущем каталоге (`jenskitchen`) каталог `images` и найдите в нем файл `jenskitchen.gif`».

А теперь добавьте изображение в файл `couscous.html`:

```

```

Выполнить эту операцию немного сложнее, чем предыдущую. Эта ссылка указывает браузеру на необходимость перехода на два уровня вверх в иерархии каталогов, а потом поиск в каталоге `images` файла изображения под именем `spoon.gif`.

Конечно, этот путь можно упростить, используя ссылку относительно корневого каталога сайта. В этом случае путь, ведущий к изображению `spoon.gif` (и любому другому файлу, находящемуся в каталоге `images`), будет следующим:

```

```

Несмотря на то что ссылка корректна, изображение не появится на странице до тех пор, пока сайт не будет загружен на сервер. С другой стороны, пока сайт находится на локальном компьютере, его проще поддерживать.

УПРАЖНЕНИЕ 6.7. ЕЩЕ НЕСКОЛЬКО ПОПЫТОК...

Прежде чем продолжать изучение материала, попробуйте создать несколько относительных ссылок, чтобы убедиться в усвоении материала. Свои ответы можете указать в книге, или, если хотите проверить свою разметку и увидеть, как она работает, внесите изменения в файлы реального сайта. Обратите внимание, что по-

ЕЩЕ ОБ ОТНОСИТЕЛЬНЫХ ССЫЛКАХ

Большинство имен путей, создаваемых при выполнении упражнения 6.7, могут быть заданы относительно корневого каталога сайта, но, для практики, запишите их как рекомендовано здесь.

казанный здесь текст, не включен в страницы упражнений — добавьте его, прежде чем приступите к созданию ссылки (например, для первого задания введите: `Go to the Tapenade recipe`). Готовый код находится в заключительных файлах упражнений в папке `materials` этой главы, а также приведен в *приложении 1*.

1. Создайте ссылку, связывающую файлы `salmon.html` и `tapenade.html`:

`Go to the Tapenade recipe`

2. Создайте ссылку, связывающую файлы `couscous.html` и `salmon.html`:

`Try this with Garlic Salmon`

3. Создайте ссылку, связывающую файлы `tapenade.html` и `linguine.html`:

`Try the Linguine with Clam Sauce`

4. Создайте ссылку, связывающую файлы `linguine.html` и `about.html`:

`About Jen's Kitchen`

5. Создайте ссылку, связывающую файлы `tapenade.html` и `www.allrecipes.com`:

`Go to Allrecipes.com`

Ссылка на определенную область страницы

Известно ли вам, что можно создать ссылку на конкретное место веб-страницы? Такой прием удобен для расположения ярлыков данных в нижней части длинной прокручиваемой страницы или для возврата к верхней части страницы одним

щелчком мыши или касанием стилуса (пальца). Ссылка на конкретную область страницы также называется ссылкой на *фрагмент* документа.

Ссылка на определенную область страницы формируется в два этапа (шага).

Сначала определяется пункт назначения, затем создается ссылка на него. В следующем примере в верхней части страницы создается алфавитный указатель, который ссылается на каждый алфавитный раздел страницы гlosсария (рис. 6.11). Если пользователь щелкает на букве **H**, он переходит к заголовку **H**, расположенному на странице ниже.

ИСПОЛЬЗУЙТЕ ТАКИЕ ССЫЛКИ ДЛЯ СТРАНИЦ С ПРОКРУТКОЙ

Ссылка на другую область той же страницы хорошо подходит для страниц с прокруткой, но в случае страницы без прокрутки применять такие ссылки нецелесообразно.

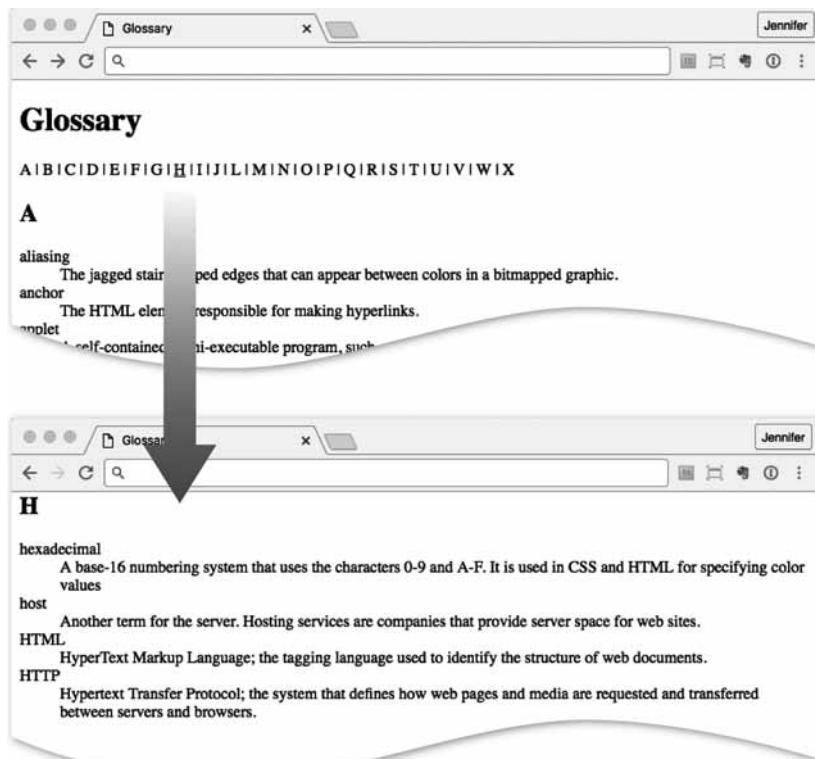


Рис. 6.11. Ссылка на определенное местоположение (фрагмент) в пределах отдельной веб-страницы

Шаг 1. Идентификация местонахождения целевого фрагмента

Можно представлять этот шаг как установку в документе флага, что позволит в дальнейшем к нему вернуться. Чтобы создать место назначения, примените атрибут `id` для присвоения целевому элементу в документе уникального имени. На веб-жаргоне это имя называется *идентификатором фрагмента*.

Помните, как вводился атрибут `id` в главе 5, где он применялся для идентификации обобщающих элементов `div` и `span`? Здесь же он используется для обозначения элемента, который будет служить идентификатором фрагмента — то есть местом назначения ссылки.

Вот пример создания идентификатора для страницы глоссария. Поскольку пользователи должны будут напрямую связываться с заголовком `H` (см. рис. 6.11, *вверху*), добавим к нему атрибут `id` и присвоим значение «`startH`»:

```
<h2 id="startH">H</h2>
<dl>
  <dt>hexadecimal</dt>
  ...

```

Шаг 2: Ссылка на место назначения

При наличии идентификатора можно создать на него ссылку. И в верхней части страницы мы формируем ссылку, направленную в нижнюю часть страницы (см. рис. 6.11, *внизу*), — на фрагмент, помеченный как «`startH`». При создании произвольной ссылки для указания местоположения ссылки используется элемент `a` с атрибутом `href`. Для указания установки ссылки на фрагмент перед именем фрагмента ставится символ «решетки»: `#`, также называемый символом *хешиа, фунта или числа*:

```
<p>... F | G | <a href="#startH">H</a> | I | J ...</p>
```

НЕ ЗАБУДЬТЕ О СИМВОЛЕ «РЕШЕТКИ»

Именам фрагментов предшествует символ «решетки» #.

И на этом все. Если щелкнуть на кнопке `H` в списке, находящемся в верхней части страницы, браузер перемещается вниз и отображает раздел, начинающийся с заголовка `H` (см. рис. 6.11).

ПУТЬ НАВЕРХ

Обычно на длинные текстовые страницы добавляется ссылка, обеспечивающая возврат к началу страницы. В этом случае пользователям не придется выполнять прокрутку обратно после щелчка на каждой ссылке.

ССЫЛКА НА ФРАГМЕНТ ДРУГОГО ДОКУМЕНТА

Для формирования ссылки на фрагмент другого документа добавим имя фрагмента в конец его интернет-адреса (абсолютный или относительный). Например, для создания ссылки на заголовок `H` страницы глоссария из другого документа этого каталога ссылка (URL) приобретет следующий вид:

```
<a href="glossary.html#startH">See the Glossary, letter H</a>
```

Можно ссылаться и на определенные целевые места на страницах других сайтов, помещая идентификатор фрагмента в конце абсолютного интернет-адреса (URL) — например, так:

```
<a href="http://www.example.com/glossary.html#startH">  
See the Glossary, letter H</a>
```

Конечно, возможность создавать именованные фрагменты на веб-страницах других пользователей нам недоступна. Целевое местоположение указывается разработчиком документа, и это доступно лишь им. Единственный способ узнать о наличии подобных ссылок — просмотреть исходный код страницы, чтобы найти эти ссылки в разметке.

Кстати, если помеченные якорями фрагменты во внешних документах перемещаются или исчезают, страница будет загружаться все равно — браузер просто перейдет в верхнюю часть страницы, как при реализации обычных ссылок.

При выполнении упражнения 6.8 мы добавим ссылки на фрагменты для примера глоссария страницы.

ДОБАВЛЯЙТЕ В КОНТЕНТ ИДЕНТИФИКАТОРЫ-ЯКОРЯ

Некоторые разработчики активно добавляют идентификаторы в начале любого тематического раздела контента — в качестве якорей (в пределах разумного уровня и в зависимости от имеющегося сайта). При этом пользователи получают возможность переходить по ссылке к любому разделу контента.

УПРАЖНЕНИЕ 6.8. ССЫЛКА НА ФРАГМЕНТ ДОКУМЕНТА

Необходима некоторая практика по созданию ссылок с определенным местом назначения? Откройте файл **glossary.html**, находящийся в папке **materials** для этой главы. Этот файл будет иметь точно такой же вид, как показано на рис. 6.11.

1. Идентифицируйте заголовок **h2 «A»** как место назначения для ссылки, именуя его с помощью атрибута **id** как **«startA»**:

```
<h2 id="startA">A</h2>
```

2. Превратите букву **A**, находящуюся в верхней части страницы, в ссылку на идентифицированный фрагмент. Не забывайте применять символ **#**:

```
<a href="#startA">A</a>
```

Повторите шаги 1 и 2 для каждой буквы, находящейся в верхней части страницы, пока к вам не придет понимание того, что вы делаете.

3. Вы также можете оформить предоставление помощи пользователям, желающим вернуться к началу страницы. Для этого создайте заголовок **Glossary** как место назначения под названием "**top**":

```
<h1 id="top">Glossary</h1>
```

4. Добавьте элемент абзаца, включающий в конце каждого буквенного раздела символы "**TOP**". Используйте "**TOP**" в качестве ссылки на идентификатор, который только что создан в верхней части страницы:

```
<p><a href="#top">TOP</a></p>
```

Скопируйте и вставьте этот код в конец каждого буквенного раздела. Теперь при просмотре любой части документа читатели смогут легко вернуться к началу страницы.

ОТКРЫТИЕ НОВЫХ ЦЕЛЕВЫХ ОКОН БРАУЗЕРА

Одна из проблем, связанных с размещением на странице ссылок, заключается в том, что после перехода по такой ссылке пользователь может не вернуться обратно к вашему контенту. Традиционным решением этой проблемы служит возможность вывода связанной страницы в новом окне браузера. При этом пользователи могут посмотреть, что там открылось по ссылке, не закрывая страницу контента.

Тем не менее помните, что открытие новых окон браузера может привести к сбоям в работе сайта. Открытие новых окон также проблематично с точки зрения доступности и может вызывать недоумение у некоторых пользователей. Они могут и не понять, что открылось новое окно, или же не вернуться к исходной странице. Новые окна часто воспринимаются как досадное недоразумение, а не дополнительное удобство. Продумайте, нужно ли вам новое окно и перевешивают ли выгоды от его наличия имеющиеся потенциальные недостатки.

Реализация способа открытия ссылки в новом окне браузера зависит от того, необходимо ли контролировать размеры этого окна. Если размеры окна не имеют значения, можно применять только HTML-разметку. Однако, если нужно открыть новое окно, имеющее определенные размеры (в пикселях), необходимо применять JavaScript (см. врезку «Всплывающие окна»).

ВСПЛЫВАЮЩИЕ ОКНА

Можно открыть окно браузера определенных размеров как с подключенными, так и отключенными компонентами браузера Chrome (панелями инструментов, полосами прокрутки и т. д.), но я не буду подробно останавливаться на этом. Прежде всего, для выполнения подобных действий нужен JavaScript. Во-вторых, в эпоху мобильных устройств открытие нового окна браузера, имеющего определенные пиксельные размеры, не требуется. Поэтому пользователи часто отключают всплывающие окна.

Как бы то ни было, небольшие баннеры, которые можно видеть на каждой веб-странице и в которых вас просят либо подписаться на список рассылки, либо ознакомиться с рекламой, созданы с помощью элементов HTML и JavaScript и не являются полноразмерными новыми окнами браузера.

Если же вы действительно желаете открыть окно браузера определенного размера, обратитесь к уроку Питера-Пауля Коша (Peter-Paul Koch) из Quirksmode: www.quirksmode.org/js/popup.html.

1. Для открытия нового окна с разметкой примените атрибут `target` элемента якоря `a`, чтобы сообщить браузеру имя окна, в котором желательно открыть связанный документ. Установите значение цели `_blank` или любое имя по вашему выбору. Помните, что с помощью этого метода нельзя управлять размером окна — оно обычно открывается как новая вкладка или в новом окне того же размера, что и последнее открытое окно в браузере пользователя. Новое окно — в зависимости от используемого браузера и устройства — может отображаться или не отображаться на переднем плане.

1. Настройка `target = "_blank"` всегда приводит к тому, что браузер открывает новое окно, например:

```
<a href="http://www.oreilly.com" target="_blank">O'Reilly</a>
```

Если указывать `target = "_blank"` для каждой ссылки, то после щелчка на такой ссылке откроется новое окно, потенциально оставляя вашего пользователя с кучей открытых окон. В этом нет ничего предосудительного, если только не пользоваться таким приемом слишком часто. Другой метод состоит в назначении целевому окну определенного имени, которое в дальнейшем может использоваться при последующих ссылках. Можно назвать окно любым именем, которое вам нравится («new», «sample», как угодно), но нельзя начинать имя с подчеркивания. Следующая ссылка откроет новое окно с именем «display»:

```
<a href="http://www.oreilly.com" target="display">O'Reilly</a>
```

При выборе окна «display» для каждой ссылки на странице каждый связанный документ будет открываться в том же втором окне. К сожалению, если второе окно останется скрытым за текущим окном пользователя, может создаться впечатление, что ссылка не работает. Так что вы уже сами решайте, какой из методов: новое окно для каждой ссылки или повторное использование именованных окон — наиболее подходит для вашего контента и интерфейса.

ПОЧТОВЫЕ ССЫЛКИ

Рассмотрим изящную ссылку `mailto`. С помощью протокола `mailto`, используемого в ссылке, можно ссылаться на почтовый адрес. Пользователь щелкает на ссылке `mailto`, браузер открывает в указанной почтовой программе новое сообщение, предварительно содержащее этот адрес (см. врезку «Спам-боты»).

Вот пример ссылки `mailto`:

```
<a href="mailto:alklecker@example.com">Contact Al Klecker</a>
```

СПАМ-БОТЫ

Имейте в виду, что размещение адреса электронной почты в исходном документе делает его уязвимым в случае пересылки по этому адресу нежелательных сообщений (известных как *спам*). Люди, создающие списки спама, иногда для поиска в Интернете адресов электронной почты используют программы автоматического поиска (так называемые *боты*).

Если необходимо, чтобы адрес электронной почты отображался на странице и чтобы люди могли увидеть его, а робот — нет, можно записать адрес таким образом, чтобы он все еще был понятен людям — например, в виде: `you[-at-]example[dot]com`.

Этот трюк не сработает в ссылке `mailto`, потому что точный адрес электронной почты указывается там как значение атрибута. Одним из решений является шифрование адреса электронной почты с помощью JavaScript. Для этого можно воспользоваться формой шифрования от Hivelogic (hivelogic.com/enkoder/). Введите текст ссылки и адрес электронной почты, а форма шифрования создаст код, который можно скопировать и вставить в документ.

Другой подход, позволяющий избежать спама, состоит в том, что не следует указывать свой адрес электронной почты в своем HTML-документе. Хорошей альтернативой при этом является использование контактной формы (веб-формы рассматриваются в главе 9).

Как можно видеть, речь идет о стандартном элементе якоря с атрибутом `href`. При этом значение устанавливается как `mailto: name@address.com`.

Браузер должен быть настроен на запуск почтовой программы, поэтому этот эффект не будет наблюдаться у всех 100% вашей аудитории. Впрочем, если функция `mailto` не работает, используйте в качестве связанного текста адрес электронной почты, и никто не останется в стороне (небольшой пример постепенного улучшения).

ТЕЛЕФОННЫЕ ССЫЛКИ

Обратите внимание, что смартфоны, которые люди используют для доступа к сайту, также могут применяться и для телефонных звонков! Почему бы не сэкономить усилия пользователей, позволяя им набрать номер телефона на сайте, просто выбрав его на странице? Соответствующий синтаксис использует протокол `tel:` и очень прост:

```
<a href="tel:+01-800-555-1212">Call us free at (800) 555-1212</a>
```

Когда мобильные пользователи касаются ссылки, то дальнейшие действия полностью зависят от устройства: Android запускает телефонное приложение, BlackBerry и IE11 Mobile инициируют вызов немедленно, а iOS запускает диалоговое окно, с помощью которого можно звонить, отправлять сообщение или добавлять номер в Контакты. Настольные браузеры могут запускать диалоговое окно для переключения приложений (например, FaceTime в Safari) или же игнорировать ссылку.

Если нежелательно прерывать работу в браузерах настольных компьютеров, можно применять правило CSS, которое скрывает ссылку для немобильных устройств (к сожалению, описание этой возможности выходит за рамки нашего обсуждения).

Вот несколько рекомендаций по использованию телефонных ссылок:

- Для значения `tel:` рекомендуется указывать полный международный телефонный номер, включая код страны, поскольку нет возможности уточнить, откуда пользователь будет обращаться к вашему сайту.
- Также включите номер телефона в содержимое ссылки, чтобы в случае сбоя ссылки номер телефона остался доступным.
- Смартфоны Android и iPhone имеют функцию обнаружения номера телефона и его автоматического превращения в ссылку. К сожалению, некоторые 10-значные числа, не являющиеся телефонными номерами, могут также стать ссылками. Если в документе есть строки чисел, которые могут быть перепутаны с номерами телефонов, можно отключить автоопределение, включив в заголовок документа элемент `meta`. Тогда нельзя будет переопределять любые стили, примененные к телефонным ссылкам:

```
<meta name="format-detection" content="telephone=no">
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

Самый важный навык, полученный при чтении этой главы, — умение создавать интернет-адреса (URL) для ссылок и изображений. Рассмотрим еще одну возможность, для того чтобы освежить навыки по созданию имен путей.

Применяя показанную на рис. 6.12 иерархию каталогов, напишите разметку для следующих ссылок и рисунков.

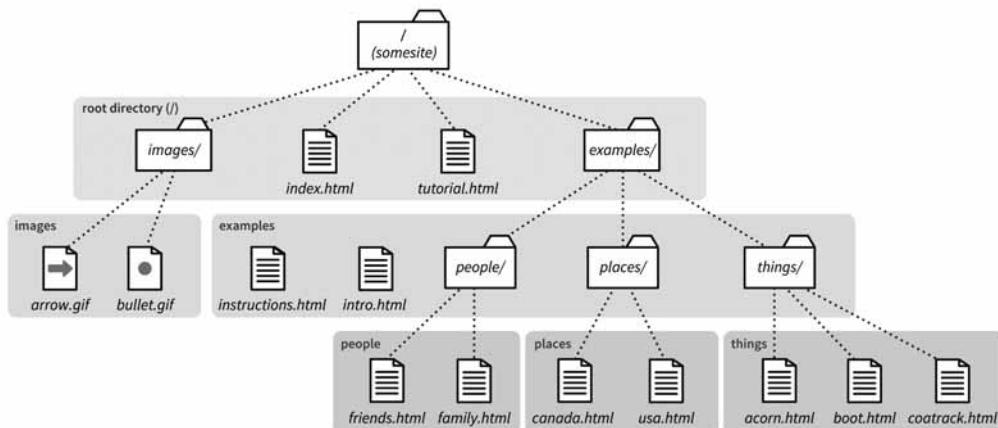


Рис. 6.12. Структура каталога размещения вопросов для разд. «Контрольные вопросы»

Диаграмма должна содержать достаточно информации для ответа на вопросы. Если для овладения навыками вам нужно выполнить практическую работу, обратите внимание на структуру каталогов, которая доступна в каталоге test, находящемся в материалах этой главы. Документы, находящиеся в каталогах, являются просто фиктивными файлами и не содержат контента. Как пример, я создала ссылку на первый документ. Ответы на контрольные вопросы приведены в *приложении 1*.

- На странице **index.html** (начальная страница) создайте разметку, реализующую ссылку на страницу **tutorial.html**.


```
<a href="tutorial.html">...</a>
```
- На странице **index.html** создайте элемент якоря для ссылки на страницу **instructions.html**.


```
<a href="#instructions.html">...</a>
```
- Создайте ссылку на страницу **family.html** со страницы **tutorial.html**.


```
<a href="family.html">...</a>
```
- Создайте ссылку на страницу **boot.html** со страницы **family.html**, но в этом случае начните с корневого каталога.


```
<a href="/boot.html">...</a>
```
- Создайте обратную ссылку на начальную страницу (**index.html**) со страницы **instructions.html**.


```
<a href="#index.html">...</a>
```
- Создайте ссылку на сайт для этой книги (**learningwebdesign.com**) в файле **intro.html**.


```
<a href="http://learningwebdesign.com">...</a>
```

7. Создайте ссылку на файл **instructions.html** со страницы **usa.html**.
8. Создайте обратную ссылку на начальную страницу (**index.html**) со страницы **acorn.html**.
9. Элемент разметки изображения **img** будет подробнее рассмотрен позднее, сейчас же заметим, что после атрибута **src** нужно заполнить относительные ссылки, указывая расположение файлов изображений для этих примеров.
10. Для размещения изображения **arrow.gif** на странице **index.html** воспользуйтесь следующей ссылкой:

```
<img src="" alt="">
```

11. Для размещения изображения **arrow.gif** на странице **intro.html** обратитесь к следующей ссылке:

```
<img src="" alt="">
```

12. Для размещения изображения **bullet.gif** на странице **friends.html** воспользуйтесь следующей ссылкой:

```
<img src="" alt="">
```

ОБЗОР ЭЛЕМЕНТОВ: ССЫЛКИ

Для создания гипертекстовых ссылок используется единственный элемент (табл. 6.1).

Таблица 6.1. Элемент, используемый для создания гипертекстовых ссылок

Элементы и атрибуты	Описание
a	Элемент якоря (гипертекстовая ссылка)
href = "URL-ссылка"	Расположение целевого файла
target = "текстовая строка"	Выбор целевого окна браузера по имени

ГЛАВА 7

ДОБАЛЕНИЕ ИЗОБРАЖЕНИЙ

В этой главе...

- ▶ *Добавление изображений с помощью элемента img*
- ▶ *Доступность изображений*
- ▶ *Добавление SVG-изображений*
- ▶ *Адаптивные (отзывчивые) изображения*

Резкий рост популярности Интернета обусловлен появлением на веб-страницах изображений. До этого в Интернете господствовал исключительно текст.

Изображения могут появляться на веб-страницах двумя способами: путем встраивания в строчный контент или в качестве фоновых изображений. Если изображения включены в редактируемый контент — например, те же фотоснимки продуктов, галереи изображений, рекламные объявления, иллюстрации и т. п., их следует поместить в поток HTML-документа. Если изображение носит декоративный характер — например, это забавное фоновое изображение для заголовка или узорчатая каемка вокруг элемента, тогда его надо добавлять на веб-страницу с помощью каскадных таблиц стилей. Этот способ рекомендуется использовать прежде всего для тех изображений, которые оказывают существенное влияние на общее представление, — тогда и документ становится чище и доступнее, и в дальнейшем значительно облегчается обновление дизайна (обратитесь к главе 13, в которой подробно рассматривается добавление фоновых изображений с помощью таблиц стилей CSS).

Эта глава посвящена процессу добавления изображений в поток документа, причем материал ее я разделила на три раздела:

- в ее первом разделе для добавления основных изображений на страницу используется хорошо известный элемент `img`, как это было принято, начиная с 1992-го года. Этот элемент отлично зарекомендовал себя за более чем 25 лет использования и будет особо полезным для тех, кто только начинает осваивать веб-дизайн;
- во втором разделе главы представлены методы, которые применяются в HTML-документах для встраивания изображений *масштабируемой векторной графики* (SVG, Scalable Vector Graphics). Использование изображений SVG требует особого подхода, поэтому они рассматриваются отдельно;

- а в третьем ее разделе рассматривается методика разметки изображений, применяемых для широкого спектра мобильных устройств, причем особое внимание уделяется относительно новым адаптивным элементам изображения (`picture` и `source`) и атрибутам (`srcset` и `sizes`). Поскольку в наше время постоянно появляются все новые и новые устройства, пригодные для просмотра веб-страниц в Интернете, понятно, что единообразный подход к этой проблеме вряд ли удовлетворит запросы по просмотру изображений во всех возможных средах: от экранов размером с ладошку устройств, работающих в медленных сетях мобильной связи, до дисплеев высокой четкости кинотеатров. Поэтому требовалось найти способ наделения изображений «отзывчивостью», при которой их отображение должно осуществляться в соответствии с условиями их просмотра. После нескольких лет общения консорциума W3C и сообщества разработчиков в спецификации HTML 5.1 появились функции адаптивных изображений, которые повсеместно поддерживаются современными браузерами.

Однако разметка адаптивного (отзывчивого) изображения не столь проста, как рассмотренные ранее в книге примеры. Она основана на более современных концепциях веб-дизайна, и синтаксис ее может оказаться сложным для понимания, особенно новичкам в программировании на HTML (следует признать, что это не столь просто и для опытных профессионалов!). Эта тема рассматривается здесь, поскольку она имеет отношение к добавлению встроенных изображений, однако вполне вероятно, что вы пока пропустите раздел, описывающий разметку адаптивных изображений, и вернетесь к нему уже после приобретения некоторого опыта работы с адаптивным веб-дизайном, располагая навыками работы с HTML и CSS.

РАЗДЕЛ ПЕРВЫЙ: ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ ЭЛЕМЕНТА `IMG`

Форматы изображений

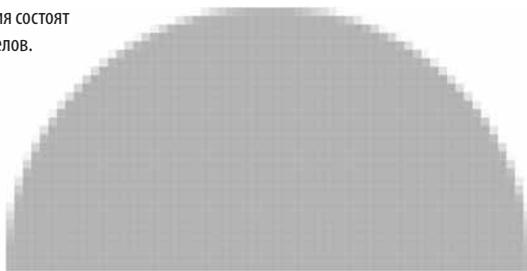
Приступая к изучению функций элемента `img` и других примеров разметки, учитите, что вы не можете разместить на веб-странице любое изображение — оно должно находиться в одном из поддерживаемых веб-форматов.

В общем случае изображения, образованные матрицей, состоящей из цветных пикселов (рис. 7.1, *вверху*), — их называют *растровыми* (*bitmapped*), для встраивания в контент должны сохраняться в форматах файлов: PNG, JPEG или GIF. Постепенно, особенно в последнее время, приобретают популярность новые, оптимизированные форматы растровых изображений: WebP и JPEG-XR, которые с помощью соответствующей разметки можно сделать доступными для браузеров, которые поддерживают такие форматы.

Для *векторных* изображений (рис. 7.1, *внизу*), таких, как значки и иллюстрации, созданные с помощью графических программ типа Adobe Illustrator, используется

формат SVG. Вопросам, связанным с форматом SVG и его функциями, полностью посвящена 25-я глава книги, но и в этой главе мы также коснемся добавления подобных изображений в HTML-документы.

Растровые изображения состоят из сетки цветных пикселов.



Векторные изображения включают контуры, которые определены математическими методами.

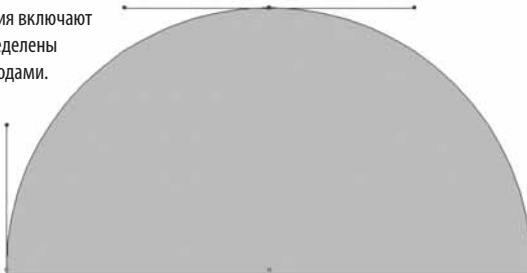


Рис. 7.1. Изображения окружностей, которые сохранены в растровом (вверху) и векторном (внизу) форматах

Если, предположим, у вас имеется исходное изображение, находящееся в другом распространенном формате, — таком, как TIFF, BMP или EPS, то, прежде чем добавить его на страницу, нужно преобразовать его в веб-формат. Если нужно обязательно сохранить графический файл в исходном формате (например, файл для программы САПР), можно сделать его доступным как *внешнее изображение*, создав ссылку непосредственно на файл изображения, например:

```
<a href="architecture.eps">Get the drawing</a>
```

Файлы изображений следует называть с указанием соответствующих расширений: **png**, **jpg** (или **jpeg**), **gif**, **webp** и **jxr**. Кроме того, сервер должен быть настроен на корректное распознавание и поддержку этих типов изображений. Все программное обеспечение веб-сервера в настоящее время настроено на обработку файлов форматов PNG, JPEG и GIF «из коробки», но при использовании SVG или одного из более новых форматов может возникнуть необходимость в добавлении такого типа мультимедиа в официальный список сервера.

На этом этапе будет полезно дать вам следующую краткую справочную информацию. Находящиеся на сервере файлы изображений, а также мультимедийные файлы имеют официальный мультимедийный тип (также называемый *тиปом MIME*) и соответствующие расширения их имен. Например, файл формата SVG имеет MIME-тип изображения **/svg+xml** и расширения **svg** и **svgz**.

Серверные пакеты располагают различными способами обработки информации MIME. В корневом каталоге популярного серверного программного обеспечения Apache имеется файл с именем `htaccess`, который содержит список всех типов файлов и возможных для них расширений. Обязательно добавьте (или попросите об этом администратора сервера) типы MIME для новых форматов изображений, чтобы их поддержка выполнялась корректно. Сервер разыскивает в списке расширение имени (например, `webp`) запрошенного файла и сопоставляет его с типом контента (`image/webp`), который включает в HTTP-отклик браузеру. При этом браузеру становится известно, какие данные к нему поступают и каким образом их следует анализировать.

Иногда для отображения мультимедиа браузеры применяют вспомогательные приложения, если иначе не удается обработать изображение. Браузер сопоставляет расширение имени файла в ссылке с соответствующим вспомогательным приложением. Если вспомогательное приложение является подключаемым модулем браузера, внешнее изображение может открываться в отдельном окне приложения или в окне браузера. Браузер также может обращаться к пользователю с предложением о сохранении файла или открытии приложения в ручном режиме. Случается, что приложение не удается открыть.

Вернемся к нашему изложению и рассмотрим элемент `img` и его обязательные и рекомендуемые атрибуты.

Элемент `img`

``

Добавляет встроенное изображение

Элемент `img` указывает браузеру: «Поместите изображение здесь». Этот элемент использовался нами для размещения графических баннеров в примерах главы 4. Элемент `img` можно поместить и в поток текста — если желательно, чтобы изображение появилось прямо в нем:

```
<p>This summer, try making pizza   
on your grill.</p>
```

Тогда изображения остаются в потоке текста (рис. 7.2), они выровнены по базовой линии текста и не приводят к разрывам строк (в HTML5 подобный вариант трактуется как *строчный элемент*).



This summer, try making pizza  on your grill.

Рис. 7.2. По умолчанию изображения выравниваются по базовой линии окружающего текста и не приводят к разрывам строк

Когда браузер идентифицирует элемент `img`, на сервер направляется запрос и файл изображения извлекается для отображения на странице. При работе в скоростной сети с достаточно быстрым компьютером или устройством, даже при формировании отдельного запроса для каждого файла изображения, веб-страница обычно отображается почти мгновенно. Однако на мобильные устройства с медленными сетевыми подключениями изображения загружаются по очереди. То же характерно и для пользователей, соединяющихся с Интернетом через коммутируемое интернет-соединение или другие медленные сети — например, Wi-Fi.

Показанные в примере атрибуты `src` и `alt` являются обязательными. Атрибут `src` (от source, источник) указывает местоположение файла изображения (его URL). Атрибут `alt` поддерживает альтернативный текст, который отображается при недоступности изображения. Об атрибуте `src` речь пойдет в следующих разделах.

А теперь несколько замечаний, относящихся к элементу `img`:

- этот элемент пуст — у него отсутствует контент. Элемент помещается в поток текста, где будет находиться изображение;
- этот элемент является встроенным, поэтому он походит на любой встроенный элемент в текстовом потоке. На рис. 7.3 демонстрируется встроенная природа элементов `img`: при изменении размера окна браузера блок изображений перформатируется для заполнения новой ширины;
- элемент `img` также представляет собой заменяемый элемент — при отображении страницы он заменяется внешним файлом. В этом его отличие от текстовых элементов, контент которых находится в источнике (и, следовательно, не заменяется);
- по умолчанию нижняя граница изображения совпадает с базовой линией текста, как показано на рис. 7.2. Применяя CSS, можно перемещать изображение к правому или левому краю, что позволит тексту его обтекать, можно обрезать его, контролировать пространство и границы вокруг изображения, изменять его вертикальное выравнивание. О соответствующих стилях CSS пойдет речь в *части III* книги.



Рис. 7.3. Встроенные изображения являются частью обычного потока документов. Изменения их расположения в потоке реализуются при изменении размеров окна браузера

АТРИБУТЫ `src` И `alt`

Атрибуты `src` и `alt` являются обязательными для элемента `img`.

Указание местоположения файла изображения с помощью атрибута *src*

src="URL"

Источник (местоположение) изображения

Значением атрибута *src* является интернет-адрес (URL) файла изображения. Чаще всего изображения, используемые на страницах, будут располагаться на вашем сервере, поэтому для указания на них могут применяться относительные URL-адреса.

УБЕДИТЕСЬ В СВОЕМ ПРАВЕ ИСПОЛЬЗОВАТЬ ИЗОБРАЖЕНИЯ

Прежде чем размещать на своей веб-странице любое изображение, убедитесь, что оно принадлежит вам, или у вас имеется письменное разрешение его правообладателя, или это изображение находится в свободном доступе. Ссылка на изображение, находящееся на чужом сервере (так называемая горячая ссылка), считается серьезным нарушением, поэтому не используйте подобный трюк, если у вас нет разрешения на использование изображения. Кроме того, при формировании «горячей ссылки» помните, что вы не сможете проконтролировать размещаемое изображение — есть риск его перемещения или переименования, что может привести к разрушению вашей ссылки.

изображений. Если изображение не располагается в том же каталоге, что и документ, нужно указать путь к файлу изображения:

```

```

Конечно, можно размещать изображения и с других сайтов, используя полный интернет-адрес (URL), как показано в следующем примере, но подобный подход не рекомендуется:

```

```

ВОСПОЛЬЗУЙТЕСЬ ПРЕИМУЩЕСТВАМИ КЭШИРОВАНИЯ

Когда браузер загружает изображение, файл сохраняется в кэше диска (в пространстве жесткого диска, выделенном для временного хранения файлов). И, если нужно заново отобразить веб-страницу, можно получить локальную копию изображения, не создавая новый запрос серверу.

Если одно изображение используется несколько раз, убедитесь в том, что атрибут *src* для каждого элемента *img* указывает на один и тот же URL-адрес на сервере. Изображение загружается один раз, а затем вызывается из кэша для последующего использования. Благодаря этому уменьшается трафик сервера и ускоряется отображение изображения для пользователя.

Если вы уже прочли главу 6, то вряд ли у вас возникнут затруднения при создании относительных URL-ссылок. Проще говоря, если изображение находится в одном каталоге с HTML-документом, на него можно ссылаться по имени файла с помощью атрибута *src*:

```

```

Разработчики обычно помещают изображения для сайта в каталоги с названиями *images* или *img* (фактически упрощая тем самым работу поисковых систем). Для каждого раздела сайта могут даже создаваться отдельные каталоги

изображений. Если изображение не располагается в том же каталоге, что и документ, нужно указать путь к файлу изображения:

```

```

Конечно, можно размещать изображения и с других сайтов, используя полный интернет-адрес (URL), как показано в следующем примере, но подобный подход не рекомендуется:

```

```

ВОСПОЛЬЗУЙТЕСЬ ПРЕИМУЩЕСТВАМИ КЭШИРОВАНИЯ

Когда браузер загружает изображение, файл сохраняется в кэше диска (в пространстве жесткого диска, выделенном для временного хранения файлов). И, если нужно заново отобразить веб-страницу, можно получить локальную копию изображения, не создавая новый запрос серверу.

Если одно изображение используется несколько раз, убедитесь в том, что атрибут *src* для каждого элемента *img* указывает на один и тот же URL-адрес на сервере. Изображение загружается один раз, а затем вызывается из кэша для последующего использования. Благодаря этому уменьшается трафик сервера и ускоряется отображение изображения для пользователя.

Поддержка альтернативного текста с помощью атрибута *alt*

`alt="text"`

Альтернативный текст

Каждый элемент `img` должен иметь атрибут `alt`, который поддерживает для размещаемого изображения альтернативный текст, предназначенный для пользователей, которые этого изображения не видят. *Альтернативный текст* (также называемый *alt text*) должен заменять содержимое изображения, передавая ту же информацию. Альтернативный текст используется программами чтения с экрана, поисковыми системами и графическими браузерами, если изображение не загружается (рис. 7.4).

В следующем примере значок PDF указывает на то, что связанный текст загружает файл в формате PDF. При этом изображение передает важный контент, который не будет донесен пользователю, если изображение не отобразится. Предоставление альтернативного текста PDF file дублирует назначение изображения:

```
<a href="application.  
pdf">High school  
application</a> 
```

Программа чтения с экрана может пояснить изображение, прочитав его значение `alt` следующим образом: «High school application. Image: PDF file».

Иногда изображения выполняют роль ссылок, в этом случае поддержка альтернативного текста имеет решающее значение, поскольку программе чтения с экрана нужно что-то читать для восприятия ссылки. В следующем примере изображение обложки книги используется в качестве ссылки на сайт книги. Альтернативный текст не описывает обложку, а выполняет функцию изображения обложки на странице (и указывает ссылку на сайт):

```
<a href="http://  
learningwebdesign.com"></a>
```

Если изображение не содержит важных добавлений к текстовому контенту страницы, рекомендуется оставить значение

`<p>If you're
and you know it clap your hands.</p>`

Здесь изображение отображается



If you're and you know it clap your hands.

Firefox

If you're happy and you know it clap your hands.

Chrome (Mac & Windows)

If you're and you know it clap your hands.

MS Edge (Windows)

If you're and you know it clap your hands.

Safari (iOS)

If you're and you know it clap your hands.

Safari (Mac)

If you're and you know it clap your hands.

Рис. 7.4. Большинство браузеров отображают альтернативный текст вместо изображения, если изображение недоступно. Лучше всего это делать Firefox, и только Safari для macOS служит исключением

атрибута `alt` пустым (незаполненным). В следующем примере декоративный цветочный узор не влияет на контент страницы, поэтому соответствующее значение `alt` не заполнено. (Можно подумать о том, будет ли подобное изображение подходящим образом обрабатываться как фоновое изображение в CSS, но сейчас не время вдаваться в подобные детали.) Обратите внимание на отсутствие символьного пространства между кавычками:

```

```

Для каждого встроенного на страницу изображения продумайте альтернативный текст, который читается вслух, а также оцените, улучшится ли впечатление от страницы или же текст, передаваемый с помощью вспомогательных средств, несколько навязчив для пользователя.

ПРИ СОСТАВЛЕНИИ АЛЬТЕРНАТИВНЫХ ТЕКСТОВ ИСПОЛЬЗУЙТЕ ОПИСАТЕЛЬНЫЕ ТЕРМИНЫ

При составлении альтернативных текстов желательно избегать применения терминов «изображения» или «графика» — и так ясно, что речь идет об изображении. Если носитель изображения — например, картина, фотография или иллюстрация, имеет отношение к контенту, лучше использовать какой-либо описательный термин.

Альтернативный текст может сослужить добрую службу пользователям, располагающим графическими браузерами. Если пользователь отключит вывод изображений в настройках браузера или если изображение не загрузится, браузер может отображать альтернативный текст, что даст пользователю представление о недостающем изображении. Однако обработка альтернативного текста несовместима с некоторыми современными браузерами, как показано на рис. 7.4.

Поддержка размеров с помощью атрибутов `width` и `height`

`width="number"`

Ширина изображения в пикселях

`height="number"`

Высота изображения в пикселях

Атрибуты `width` и `height` задают размеры изображения в пикселях. Браузеры используют указанные размеры для предварительного выделения необходимого места в макете во время загрузки изображений, чтобы всякий раз при появлении следующего изображения не приходилось восстанавливать страницу. Подобный подход ускоряет отображение веб-страницы. При указании только одного размера изображение пропорционально масштабируется.

Впрочем, эти атрибуты не столь и полезны при создании современного веб-дизайна. Не следует пользоваться ими для изменения размера изображения (используйте для этого программу редактирования изображений или CSS), а также надо полностью исключить задание этих размеров, если используется один из методов

адаптивного изображения, представленных в этой главе далее. Эти атрибуты могут применяться с изображениями, которые отображаются в одном фиксированном размере на всех устройствах, — например, логотипов или пиктограмм, что позволит браузеру получить подсказку при формировании макета.

Убедитесь, что указанные в пикселях размеры являются фактическими размерами изображения. Если значения в пикселях отличаются от фактических размеров изображения, браузер внесет корректизы в размер изображения с учетом указанных значений (рис. 7.5). Если применяются атрибуты `width` и `height`, а изображение получается искаженным или слегка размытым, убедитесь, что значения в пикселях и фактические размеры изображения синхронизированы.

ДОСТУПНОСТЬ ИЗОБРАЖЕНИЙ: ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ

Изображения некоторых типов — например, графиков и диаграмм, должны сопровождаться длинными описаниями, которые непрактичны в качестве альтернативных значений. В таких случаях необходимо использовать альтернативные стратегии доступности, с которыми можно ознакомиться с помощью следующих ресурсов:

- «Accessible Images», доступный на сайте на WebAIM: webaim.org/techniques/images/;
- «Alternative Text», доступный на сайте WebAIM: webaim.org/techniques/alttext/;
- The Web Content Accessibility Guidelines (WCAG 2.0), доступный на сайте W3C (www.w3.org/TR/WCAG20-TECHS/) — содержит методы, улучшающие доступность всего веб-контента. Следует, однако, предупредить: изложение весьма скжатое.

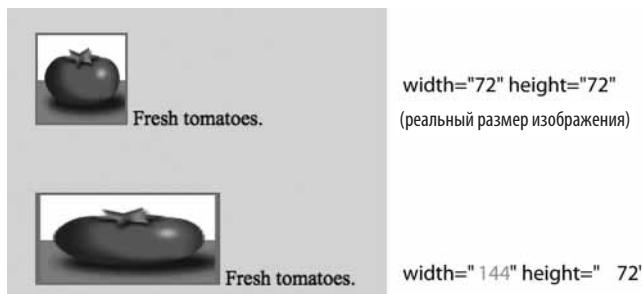


Рис. 7.5. Браузеры правильно воспринимают размеры изображений, если эти размеры, указанные в атрибутах `width` и `height`, соответствуют их реальным размерам (вверху), но вам не следует с помощью этих атрибутов пытаться вносить изменения в размеры изображений (внизу)

Ознакомившись с основными особенностями элемента `img`, выполните *упражнение 7.1* и добавьте несколько фотографий на сайт «Black Goose Bistro Gallery».

УПРАЖНЕНИЕ 7.1. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ И ИЗОБРАЖЕНИЙ-ССЫЛОК

В процессе выполнения этого упражнения мы добавим на веб-страницы миниатюры (небольшие версии) изображений, которые затем можно будет использовать в качестве ссылок. Необходимые полноразмерные фотографии и их миниатюры для вас уже созданы, вы также располагаете представлением о базовых стилях

HTML-файлов. Файлы изображений и исходный код доступны по адресу: learnin-gwebdesign.com/5e/materials. Скопируйте на жесткий диск папку **gallery** и удостоверьтесь, что она организована должным образом.

Этот небольшой сайт состоит из главной страницы **index.html** (рис. 7.6, а) и трех отдельных HTML-документов, каждый из которых содержит изображения большего размера (рис. 7.6, б). Сначала добавим миниатюры, затем на соответствующие страницы добавим их полноразмерные версии, после чего создадим на основе миниатюр ссылки на эти страницы.

Итак, приступаем к делу.

Откройте файл **index.html** и добавьте на эту страницу миниатюры, сопровождающие текст. Первую миниатюру для вас я уже добавила:

```
<p><br>We start our day at the...
```

Изображение помещается в начало абзаца после открывающего тега **<p>**. Поскольку все файлы миниатюр находятся в каталоге миниатюр, путь к ним указывается в виде URL. Добавлено описание изображения с атрибутом **alt**. Кроме того, поскольку известно, что эти миниатюры будут отображаться на всех устройствах с шириной и высотой, равной точно 200 пикселам, установленные атрибуты **width** и **height** сообщают браузеру о том, сколько места следует оставить в макете. Теперь ваша очередь.

1. Добавьте миниатюры изображений **burgers-200.jpg** и **fish-200.jpg** в начало абзацев соответствующих разделов, следя только что приведенному моему примеру. Не забудьте включить пути и содержательные альтернативные текстовые описания. Наконец, добавьте разрыв строки **
** после элемента **img**.

Завершив добавление миниатюр, сохраните файл и откройте его в браузере, чтобы убедиться, что изображения имеются и отображаются в нужном размере.

2. Затем добавьте изображения в отдельные HTML-документы. Обработку документа **bread.html** я уже выполнила:

```
<h1>Gallery: Baked Goods</h1>  
<p></p>
```

Обратите внимание, что полноразмерные изображения находятся в каталоге **photos**, поэтому необходимо отразить этот факт в записи путей. Обратите также внимание, что, поскольку эта страница не предназначена для адаптивной работы и для всех устройств изображения будут иметь фиксированный размер, я пошла дальше и включила здесь также атрибуты **width** и **height**.

Самостоятельно добавьте изображения в документы **burgers.html** и **fish.html**, следя приведенному мной примеру. Подсказка: все изображения имеют ширину 800 пикселов и высоту 600 пикселов.

Сохраните каждый файл и проверьте свою работу, открыв их в окне браузера.

3. Вернитесь к документу **index.html** и создайте миниатюры-ссылки на соответствующие файлы. Первую ссылку я уже создала:

```
<p><a href="bread.html"></a><br>We start our day at the crack of dawn...
```

Обратите внимание, что URL-ссылки относятся к текущему документу (**index.html**), а не к местоположению изображения (каталогу **thumbnails**).

Создайте ссылку на оставшиеся миниатюры для каждого из документов. Если все изображения видны и можно перейти по ссылке на каждую страницу и снова вернуться на начальную страницу, то примите мои поздравления!

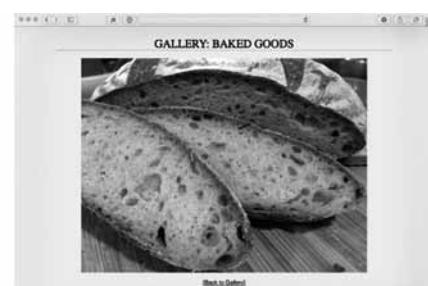
НУЖНА ДОПОЛНИТЕЛЬНАЯ ПРАКТИКА?

Если хотите еще попрактиковаться, найдите в соответствующих каталогах три дополнительных изображения: **chicken-800.jpg**, **fries-800.jpg** и **tabouleh-800.jpg** и их миниатюрные версии: **chicken-200.jpg**, **fries-200.jpg** и **tabouleh-200.jpg**. При этом желательно добавить на домашнюю страницу собственные описания и создать HTML-документы для полноразмерных изображений.

В приведенном в этом разделе главы изложении затронуты лишь основные моменты, связанные с процессом добавления изображений на страницу. Далее мы рассмотрим добавление на страницу SVG-изображений, что представляет особый интерес — как по отношению к применяемому формату, так и по способам добавления изображений в HTML.



a



b

Рис. 7.6. Страницы галереи фотографий: *a* — главная страница сайта с миниатюрами изображений; *б* — страницы сайта с полноразмерными изображениями

РАЗДЕЛ ВТОРОЙ: ДОБАВЛЕНИЕ SVG-ИЗОБРАЖЕНИЙ

Изложение методик по добавлению изображений на веб-страницы будет неполным без рассмотрения добавления изображений SVG (Scalable Vector Graphics, масштабируемая векторная графика). В наше время распространенность SVG-изображений в немалой степени связана с почти повсеместной их поддержкой браузерами и популярностью изображений, размер которых можно изменять без потери качества. Для изображений, выполняющих роль иллюстраций, именно такие изображения воплощают мечту в реальность. Более подробно SVG-изображения рассматриваются в главе 25, а в этом разделе приводится лишь обзор основных моментов, имеющих ключевое значение для разметки их добавления.

Уже упоминалось, что формат SVG хорошо подходит для хранения векторных изображений (см. рис. 7.1, *внизу*). В отличие от матрицы пикселов растровых изображений, векторные изображения состоят из форм и контуров, которые рассчитываются математически. И что важно — для SVG эти формы и контуры определяются записанными в текстовом файле инструкциями. Представьте себе — изображения описываются текстом! Все формы и контуры, а также их свойства записаны на стандартизированном языке разметки SVG. Точно так же как в HTML имеются элементы для разметки абзацев: *p* и таблиц: *table*, SVG располагает элементами, определяющими формы, — такими, как прямоугольник: *rect*, окружность: *circle* и пути: *path*.

ЯЗЫК РАЗМЕТКИ SVG

Язык разметки SVG представляет собой приложение языка XML (Extensible Markup Language, расширяемого языка разметки), который задает правила и стандарты того, как создавать языки разметки и как они должны работать совместно друг с другом. В результате разметка SVG хорошо сочетается с HTML-контентом.

Рассмотрим простой пример, иллюстрирующий основной принцип SVG-разметки. В следующем SVG-коде описывается прямоугольник (*rect*) с закругленными углами *rx* и *ry* (x-радиус и y-радиус, соответственно), слово *hello* установлено как *text* с атрибутами для шрифта и цвета (рис. 7.7). Браузеры, поддерживающие SVG, просматривают эти инструкции и в точности отображают то, что придумано автором:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 300 180">
    <rect width="300" height="180" fill="purple" rx="20" ry="20"/>
    <text x="40" y="114" fill="yellow" font-family="'Verdana-Bold'" 
    onsize="
        72">
        hello!
    </text>
</svg>
```

Для определенных типов изображений именно SVG-изображения характеризуются некоторыми важными преимуществами по сравнению с их растровыми аналогами:

- поскольку при создании векторных изображений в файлах сохраняются только инструкции для их рисования, они обычно нуждаются в меньших объемах дан-

- ных, чем изображение, сохраненное в растровом формате. В результате ускоряется их загрузка, а также улучшается производительность сайта;
- поскольку SVG-изображения являются векторными, они могут без потери качества по мере необходимости изменять свои размеры в адаптивном макете. SVG-изображения всегда имеют отличный вид и хорошую четкость — никаких размазанных краев!
 - SVG-изображения, будучи по сути текстовыми файлами, хорошо интегрируются с HTML/XML и могут быть — как и HTML-файлы — сжаты с помощью таких инструментов, как Gzip и Brotli;
 - SVG-изображения могут применяться для создания эффектов анимации;
 - для изменения внешнего вида таких изображений применяются каскадные таблицы стилей;
 - к SVG-изображениям с помощью JavaScript можно добавить интерактивность — чтобы при наведении пользователем курсора мыши или по щелчку на изображении происходили необходимые его изменения.



Рис. 7.7. Простое SVG-изображение, составленное из прямоугольника и текста

Напомню, что необходимые нюансы, связанные с формированием SVG-изображений, а также ряд их функций, подробно обсуждаются в главе 25. Сейчас же мы сосредоточимся на коде HTML, который необходим для размещения подобных изображений в потоке веб-страницы. Для этого имеется несколько вариантов: вложение с помощью элемента `img`, запись кода в виде строчного элемента `svg`, вложение с помощью элемента `object` и применение их в качестве фонового изображения с помощью CSS.

Размещение SVG-изображений с помощью элемента `img`

Текстовые файлы SVG, которые сохранены с расширением `svg` (иногда их называют *автономными файлами SVG*), можно рассматривать как изображение и размещать их в документе с помощью элемента `img`. Поскольку к этому моменту вы уже располагаете достаточным объемом сведений об элементе `img`, вряд ли следующий пример вызовет у вас затруднения:

```

```

Преимущества и недостатки этого способа

Преимущество размещения SVG-изображений с помощью элемента `img` заключается в том, что этот элемент универсальным образом поддерживается в браузерах, которые поддерживают SVG-изображения.

Этот подход удобен в случае применения автономного изображения SVG как простой замены GIF или PNG, но не следует сбрасывать со счетов и ряд недостатков, сопровождающих процесс добавления SVG-изображения с помощью элемента `img`:

- к элементам в SVG нельзя применять стили с помощью внешней таблицы стилей — такой, как примененный ко всей странице файл с расширением `css`. Файл с расширением `svg` может содержать собственную внутреннюю таблицу стилей, для чего используется элемент `style`, но он обеспечивает стайлинг элементов только внутри этого файла. Впрочем, имеется возможность применить стили к самому элементу `img`;
- поскольку элементами в SVG невозможно манипулировать с помощью JavaScript, вы не сможете добавлять в документы интерактивность. Сценарии веб-документа не видят контент SVG, причем сценарии для файлов SVG не запускаются вообще. Другие интерактивные эффекты, такие, как ссылки или стили `:hover`, также не будут запускаться внутри SVG-файла, вложенного с помощью элемента `img`;
- нельзя применять в SVG и внешние файлы — такие, как встроенные изображения или веб-шрифты.

Другими словами, автономные SVG-файлы ведут себя, как будто они находятся в своем собственном маленьком, но автономном мирке. Но при обработке статических иллюстраций — это просто отличный вариант.

Поддержка отображения браузерами SVG-изображений, размещенных с помощью элемента `img`

СЕРВЕРНАЯ КОНФИГУРАЦИЯ SVG

Если при использовании SVG-изображений отображение вашего сайта имеет недостатки, может потребоваться настройка сервера для распознавания типа SVG-изображения, о чем шла речь в начале этой главы. Далее показано, как этого добиться при работе с сервером Apache, но аналогичные конфигурации могут быть реализованы и для других серверных языков:

```
AddType image/svg+xml .svg
```

Несомненным преимуществом является тот факт, что все современные браузеры поддерживают SVG-изображения, вложенные с помощью элемента `img`. Двумя известными исключениями служат Internet Explorer версии до 8-й включительно, а также браузер Android до версии 3. И эти браузеры все еще используются на практике. Если вы вынуждены поддерживать подобные устаревшие браузеры, можно воспользоваться одним из обходных способов, которые будут рассмотрены в разд. «Обходные пути для SVG».

Встраивание контента SVG-изображения в исходный код HTML

`<svg>`

Вложенное SVG-изображение

Другим вариантом для размещения на веб-странице SVG-изображений является копирование контента SVG-изображения и непосредственная вставка его в

HTML-документ. Подобный подход называется использованием *встроенного SVG*-изображения. Вот пример, очень похожий на пример добавления изображения с помощью элемента `img`, приведенный на рис. 7.2, только в этом случае пицца представлена как векторное изображение, нарисованное с использованием элементов `circle` и вложенное с помощью элемента `svg` (рис. 7.8). Каждый элемент `circle` имеет атрибуты, описывающие цвет заливки, положение центральной точки (`cx` и `cy`) и длину радиуса (`r`):

```
<p>This summer, try making pizza

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 72 72" width="100"
height="100">
  <circle fill="#D4AB00" cx="36" cy="36" r="36"/>
  <circle opacity=".7" fill="#FFF" stroke="#8A291C" cx="36.1" cy="35.9"
r="31.2"/>
  <circle fill="#A52C1B" cx="38.8" cy="13.5" r="4.8"/>
  <circle fill="#A52C1B" cx="22.4" cy="20.9" r="4.8"/>
  <circle fill="#A52C1B" cx="32" cy="37.2" r="4.8"/>
  <circle fill="#A52C1B" cx="16.6" cy="39.9" r="4.8"/>
  <circle fill="#A52C1B" cx="26.2" cy="53.3" r="4.8"/>
  <circle fill="#A52C1B" cx="42.5" cy="27.3" r="4.8"/>
  <circle fill="#A52C1B" cx="44.3" cy="55.2" r="4.8"/>
  <circle fill="#A52C1B" cx="54.7" cy="42.9" r="4.8"/>
  <circle fill="#A52C1B" cx="56" cy="28.3" r="4.8"/>
</svg>

on your grill.</p>
```



This summer, try making pizza

on your grill.

Рис. 7.8. Здесь представлено SVG-изображение пиццы, состоящее из 11 элементов `circle`. Вместо элемента `img` исходный SVG-код размещается непосредственно в HTML-документе с помощью элемента `svg`

Этот код сгенерирован с помощью программы Adobe Illustrator, в которой предварительно была создана иллюстрация, сохраненная затем в формате SVG. Я также выполнила оптимизацию кода, что позволило избавиться от ряда необязательных дополнений, внесенных в нее Illustrator (процедура оптимизации изображений SVG будет рассмотрена в главе 25).

Преимущества и недостатки этого способа

Встроенные SVG-изображения позволяют разработчикам в полной мере использовать возможности SVG. Если SVG-разметка располагается рядом с HTML-разметкой, все ее элементы являются частью основного DOM-дерева. Поэтому можно

обращаться к SVG-объектам и манипулировать ими с помощью JavaScript, заставляя реагировать на взаимодействие с пользователем или ввод данных. Для таблиц стилей имеются аналогичные преимущества, поскольку элементы в SVG могут наследовать стили HTML-элементов. В этом случае легко применять одинаковые стили как к элементам на странице, так и для SVG-графики.

С другой стороны, код для SVG-иллюстраций большой и громоздкий, что приводит к увеличению размеров HTML-документов, которые трудно воспринять. Даже размещение картинки небольшой пиццы уже потребует добавления солидного блока кода. Подобные особенности затрудняют поддержку изображений для сайта, поскольку скрыты в HTML-документах. Еще одним недостатком является отсутствие кэширования браузером вложенных SVG-файлов отдельно от HTML-файла, поэтому подобной методики следует избегать при обработке больших изображений, которые используются повторно большим числом HTML-страниц.

Поддержка отображения браузерами SVG-изображений, размещенных с помощью элемента *svg*

Хорошая новость заключается в том, что SVG-изображения, встроенные с помощью элемента *svg*, поддерживают все современные браузеры. Но более старые версии браузеров такой поддержки осуществить не могут. Среди них: Internet Explorer версии до 8-й включительно, Safari версии до 5-й включительно, мобильный браузер Android до версии 3 и iOS до версии 5.

Размещение SVG-изображений с помощью элемента *object*

В состав HTML входит универсальный элемент, предназначенный для встраивания мультимедийных объектов, — элемент *object*. Более подробно возможности этого элемента будут рассмотрены в главе 10, но пока убедимся в том, что элемент *object* представляет собой еще один вариант для размещения на веб-странице SVG-изображения. Этот вариант является хорошим компромиссом между применением элемента *img* и вложенного контента SVG. В результате можно получить полностью функциональное SVG-изображение, но инкапсулированное в отдельный кэшируемый файл.

Открывающий тег *object* определяет тип мультимедиа (изображение *svg + xml*) и указывает на файл, который будет применяться с атрибутом *data*. Элемент *object* поставляется с собственным механизмом резервирования — любой контент в объекте визуализируется, даже если указанный с помощью атрибута *data* мультимедийный объект не может быть отображен. Так что, если файл SVG не поддерживается или не загружается, то с помощью элемента *img* отображается PNG-версия изображения:

```
<object type="image/svg+xml" data="pizza.svg">

</object>
```

Однако тут имеет место одна проблема. Некоторые браузеры загружают резервное изображение, даже если они поддерживают SVG и не нуждаются в такой подмене.

Подобная загрузка не столь безобидна. Обходной путь состоит в том, чтобы превратить находящееся в пустом контейнере `div` резервное изображение в фоновое CSS-изображение. К сожалению, оно не столь эластично при масштабировании и определении размеров, но зато вполне решает проблему подменной загрузки:

```
<object type="image/svg+xml" data="pizza.svg">
  <div style="background-image: url(pizza.png); width 100px; height: 100px;" role="img" aria-label="pizza">
    </object>
```

Преимущества и недостатки этого способа

Основным преимуществом, проявляющимся в процессе вложения SVG-изображений с помощью элемента `object`, является возможность создания сценария и выполнения загрузки внешних файлов. Также могут применяться сценарии для доступа к родительскому HTML-документу (с некоторыми ограничениями по безопасности). Однако, поскольку эти изображения являются отдельными файлами и не входят в модель DOM для страницы, в HTML-документе для стайлинга элементов в пределах SVG нельзя применять таблицу стилей. Вложенные SVG-файлы также могут вызывать ошибки при отображении в браузерах, поэтому обязательно тщательно их тестируйте.

ДРУГИЕ ВОЗМОЖНОСТИ РАЗМЕЩЕНИЯ SVG-ИЗОБРАЖЕНИЙ

Более ранние способы добавления SVG-изображений предлагали для встраивания мультимедийных объектов использовать два других HTML-элемента: `embed` и `iframe` (речь о них пойдет в главе 10). Делать так при работе с SVG-изображениями вполне возможно, они прекрасно работают в поддерживающих SVG браузерах, но большинство разработчиков относятся к ним как к устаревшим методикам. Сейчас принято применять элементы: `img`, `inline svg` и `object` и дополнительно обрабатывать изображение с помощью CSS.

Использование фоновых изображений с помощью CSS

И хотя эта глава посвящена HTML, но именно здесь следует упомянуть о том, что SVG-изображения могут служить фоновыми изображениями при использовании возможностей CSS. Далее приводится пример стилевого правила, когда декоративное изображение является фоном для раздела `header`:

```
header {
  background-image: url(/images/decorative.svg);
}
```

Обходные пути для SVG

Как упоминалось ранее, все современные браузеры поддерживают SVG: либо через добавление изображений с помощью элементов `img` или `object`, либо через встраивание контента SVG-изображения, что уже хорошо. Однако, если журналы вашего сервера показывают значительный трафик из Internet Explorer версии до 8-й

и более ранних, Android версии 3 и более ранних или Safari 5 и более ранних, или если вашему клиенту нужна поддержка именно этих браузеров, может возникнуть необходимость в использовании обходных путей. Одним из таких вариантов является применение элемента `object` для вложения SVG-изображений на страницу и использование свойства по резервированию контента (см. разд. «Размещение SVG-изображений с помощью элемента `object`»).

Если SVG применяется как изображение, вкладываемое с помощью элемента `img`, другой вариант обходных путей заключается в использовании элемента `picture` (о нем см. далее «Раздел третий: разметка адаптивных изображений»). Элемент `picture` может применяться для создания нескольких версий изображения с использованием различных форматов. Каждая версия предлагается с элементом `source`, который в следующем примере указывает на изображение `pizza.svg` и определяет его мультимедийный тип. Элемент `picture` также имеет вложенный запасной механизм резервирования. Если браузер не поддерживает предложенные исходные файлы или не поддерживает элемент `picture`, пользователи познакомятся с PNG-изображением с помощью старого доброго элемента `img`:

```
<picture>
  <source type="image/svg+xml" srcset="pizza.svg">
    <img srcset="pizza.png" alt="No SVG support">
</picture>
```

Если прибегнуть к услугам Google, осуществив поиск по фразе: «Обходные пути для SVG» (SVG fallbacks), вы, вероятно, получите серию ссылок, во многих из которых для определения поддержки используется JavaScript. Если ваше любопытство по части обходных путей для SVG так и не будет удовлетворено, рекомендую ознакомиться со статьёй Амелии Беллами-Ройд (Amelia Bellamy-Royd) «A Complete Guide to SVG Fallbacks» (css-tricks.com/a-complete-guide-to-svgfallbacks/) или обратиться к книге Криса Койера (Chris Coyier) «Practical SVG», после чего вы получите исчерпывающее представление о современном состоянии вопроса. Как бы там ни было, устаревшие версии Internet Explorer и Android больше не представляют проблему.

Готовы ли вы «покрутить» SVG-изображение? Попробуйте, применяя некоторые методы вложения, которые рассматриваются в *упражнении 7.2*.

УПРАЖНЕНИЕ 7.2. ДОБАВЛЕНИЕ SVG-ИЗОБРАЖЕНИЯ НА СТРАНИЦУ

При выполнении этого упражнения на страницу «Bistro Black Goose», которая нам знакома по главе 4, добавляются некоторые SVG-изображения. Материалы для этого упражнения доступны в Интернете по адресу: learningwebdesign.com/5e/materials. Они, а также результатирующий код содержатся в каталоге под именем `svg`. Упражнение состоит из двух частей: в первой части с помощью версии SVG заменяется логотип, во второй части в нижней части страницы добавляется ряд значков социальных сетей (рис. 7.9).



Рис. 7.9. Страница «Bistro Black Goose», сформированная с помощью SVG-изображений

Часть I. Замена логотипа

1. Откройте в текстовом редакторе файл `blackgoosebistro.html`. Вы получите изображение, которое имеет тот же вид, который имело при выполнении упражнений из главы 4.
2. Просто для интереса посмотрим, что же произойдет, если создать увеличенный вариант текущего PNG-логотипа. Добавьте на вкладку `img` значения: `width="500"` `height="500"`. Сохраните этот файл и откройте его в браузере, чтобы убедиться, какие размытые растровые изображения получаются при увеличении размеров. Просто ужас!
3. Давайте заменим теперь исходное изображение на SVG-версию того же логотипа, используя метод встраивания SVG. В папке `svg` находится файл `blackgoose-logo.svg`. Откройте его в текстовом редакторе и скопируйте текст полностью (от тега `<svg>` до тега `</svg>`).
4. Вернитесь к файлу `blackgoosebistro.html` и полностью удалите элемент `img` (действуйте осторожно, чтобы сохранить окружающую его разметку). Вставьте текст SVG вместо удаленных элементов `img`. Если вы посмотрите на него внимательно, то увидите, что SVG включает два круга, определение градиента и два пути (один представляет взрыв звезды, другой — изображение гуся).
5. Затем установите размер, с учетом которого на странице отображается SVG. В открывающем теге `svg` добавьте атрибуты `width` и `height`, причем каждому атрибуту присвойте значения `200px`:

```
<h1><svg width="200px" height="200px" ...>
```

Сохраните этот файл и откройте страницу в браузере. SVG-логотип, очень похожий на старое изображение, должен отобразиться на своем месте.

6. Попробуйте поэкспериментировать: что произойдет, если сделать логотип SVG действительно большим! Измените значения для ширины и высоты до значения, равного 500 пикселам, сохраните файл и перезагрузите страницу в браузере. Логотип должен отображаться как крупное и четкое изображение! Никаких размытых краев, как в случае PNG-изображения, быть не должно. Теперь верните значения размеров к 200×200 или оставьте измененные значения — на ваш выбор.

Часть II. Добавление значков

1. Создадим в нижней части страницы нижний колонтитул для значков социальных сетей. Под разделом Location & Hours добавьте следующее (в пустой абзац будут добавлены логотипы):

```
<footer>
    <p>Please visit our social media pages</p>
    <p> </p>
</footer>
```

2. Примените элемент `img` для размещения трех значков SVG: `twitter.svg`, `facebook.svg` и `instagram.svg`. Заметьте, что они размещаются в каталоге значков `icons` (там также имеются значки для Tumblr и GitHub — если вы желаете потренироваться). Для примера далее приводится код, который нужно записать для размещения значка `twitter.svg`:

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

3. Сохраните этот файл и откройте его в браузере. Значки должны отображаться на своем месте, но размер их просто огромен. Создадим пару стилевых правил, что значительно улучшит вид нижнего колонтитула. Имея недостаточно практики при работе с правилами стиля, просто скопируйте то, что отображено внутри элемента `style` в заголовке документа:

```
footer {
    border-top: 1px solid #57b1dc;
    text-align: center;
    padding-top: 1em;
}
footer img {
    width: 40px;
    height: 40px;
    margin-left: .5em;
    margin-right: .5em;
}
```

4. Снова сохраните файл и откройте его в браузере (отобразится страница, похожая на представленную на рис. 7.9). Если хотите поэкспериментировать, попробуйте изменять настройки стиля или даже код вложенного SVG, что позволит узнать, каким образом они влияют на внешний вид изображений. Веселое занятие, не правда ли?!

РАЗДЕЛ ТРЕТИЙ: РАЗМЕТКА АДАПТИВНЫХ ИЗОБРАЖЕНИЙ

Вскоре после того как в обиход вошли смартфоны, планшеты, «фаблеты» и другие подобные им устройства, стало ясно, что большие изображения, которые отлично смотрятся на большом экране, представляют собой форменное излишество при отображении на экранах меньшего размера. Все эти изображения загружаются, но толку от них никакого. Обращение к крупным изображениям при работе с устройствами небольшого размера замедляет отображение страниц и приводит лишь к дополнительным расходам — в зависимости от тарифного плана пользователя (и стоимости сервера). И наоборот, небольшие изображения, хоть и быстро загруженные, могут иметь весьма неопрятный вид при рассмотрении их на больших экранах с высокими разрешениями. Таким образом, нам нужно сделать так, чтобы как целые веб-страницы, так и изображения, находящиеся на этих страницах, научились приспособливаться — адаптироваться — к различным размерам экрана. Наш верный элемент `img` с единственным атрибутом `src` просто незаменим в таких случаях.

Потребовалось несколько лет экспериментов и обсуждений между создателями браузеров и сообществом веб-разработчиков, но теперь у нас есть способ предложить альтернативные изображения, используя только разметку HTML.

На разработку методики формирования альтернативных изображений, ориентированную лишь на HTML-разметку и не требующую никаких сложных ухищрений с применением JavaScript или преобразований на стороне сервера, ушло несколько лет совместной работы и создателей браузеров, и сообщества веб-разработчиков. В результате появились функции адаптивного (отзывчивого) изображения (атрибуты `srcset` и `sizes`, а также элемент `picture`), которые включены в спецификацию HTML 5.1. Нарастает и поддержка их со стороны браузеров — так, в сентябре 2014-го года самым продвинутым в этом плане стал браузер Chrome.

Благодаря надежной защите от ошибок и наличию сценариев, обеспечивающих поддержку и для старых браузеров, мы можем сразу же приступить к применению этих методов. Тем не менее нет ничего, что не может быть усовершенствовано. Принятые адаптивные решения для изображений вполне возможно будут доработаны и улучшены и не исключено, что устареют. И если вы собираетесь включить их в сайты, хорошо бы вам сначала ознакомиться с работами группы сообщества Responsive Images (RICG, Responsive Images Community Group). Группа RICG (responsiveimages.org) представляет собой объединение специалистов, которые совместно с разработчиками браузеров работали над созданием текущей спецификации. Именно им известны все направления работы в этом направлении. Следует также поискать свежие статьи на эту тему и, не исключено, даже предложить в эту спецификацию возможные изменения.

Принципы обработки адаптивных изображений

Когда мы говорим «адаптивные изображения», речь идет об изображениях, которые адаптированы к среде просмотра пользователя. Прежде всего, адаптивные графические технологии не позволяют браузерам загружать на небольшие экраны больше

КАК ЭТО РАБОТАЕТ?

Вы предоставляеме несколько изображений, подогнанных по размеру или обрезанных так, чтобы они подходили для экранов разных размеров, а браузер выбирает наиболее соответствующее изображение на основе сведений о текущей среде для просмотра.

того, в подобных сетях можно использовать преимущества новых, более эффективных форматов изображений.

Если вкратце, то аддаптивные изображения работают следующим образом. Вы предоставляете несколько изображений, подогнанных по размеру или обрезанных так, чтобы они подходили для экранов разных размеров, а браузер выбирает наиболее соответствующее изображение на основе сведений о текущей среде для просмотра. Размеры экрана служат одним из факторов, но также учитываются разрешение, быстродействие сети, наличие данных в кэше, пользовательские настройки и другие факторы.

Атрибуты и элементы аддаптивного изображения предназначены для реализации следующих четырех основных сценариев:

- поддержка очень больших изображений, которые четко отображаются на экранах с высоким разрешением (*плотностью*);
- поддержка набора изображений различных размеров, отображаемых на экранах с разными размерами;
- поддержка версий изображения с различной степенью детализации в зависимости от размера и ориентации устройства (известны как сценарии использования *art direction* или *принципов художественного оформления*);
- поддержка альтернативных форматов изображений, которые сохраняют одинаковые изображения при использовании файлов гораздо меньших размеров.

Рассмотрим каждый из этих вариантов более подробно.

Мониторы высокой плотности (x-дескриптор)

Представленное на экране изображение состоит из маленьких цветных квадратиков, называемых *пикселями*. Пиксели, образующие сам экран, называют *пикселями устройства* (они также иногда называются *аппаратными* или *физическими пикселями*). До недавнего времени экраны обычно вмещали 72 или 96 пикселов устройства на дюйм (сейчас нормой является от 109 до 160 пикселов устройства на дюйм). Количество пикселов на дюйм (ppi) — это *разрешение* экрана.

Растровые изображения, такие, как JPEG, PNG и GIF, также состоят из матрицы пикселов. Принято полагать, что пиксели в изображениях, а также указанные в таблицах стилей размеры в пикселях находятся с пикселями устройств в соотношении один к одному. Элемент изображения или блок шириной 100 пикселов размещается на 100 пикселях устройства. Просто и красиво.

графических данных, чем это необходимо. Они также содержат механизм, который позволяет при работе в сетях, обеспечивающих быструю передачу изображений, представлять изображения с высоким разрешением, которые выглядят просто великолепно. Кроме

Соотношение пикселов устройств

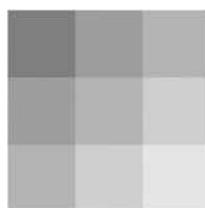
Неудивительно, что сегодня все уже не так просто. Производители увеличивают разрешение экранов, стремясь улучшить качество изображения. В результате пиксели устройства становятся все меньше и меньше, причем уже и настолько маленькими, что изображения и текст становятся неразборчивыми, если выводить их в соотношении один к одному.

Для компенсации этого эффекта на устройствах применяются единицы измерения, называемые *опорными пикселями*. Опорные пиксели также называются *пиксельными точками* (PT) в iOS, *независимыми от устройства пикселями* (DP или DiP) в Android или *CSS-пикселями*, поскольку служат единицей измерения в таблицах стилей. Экран iPhone 8 содержит 750×1334 пикселов устройства, но использует макетную сетку величиной 375×667 точек, или CSS-пикселов (соотношение 2 пикселя устройства на 1 пиксель разметочной сетки — 2:1 или 2x). То есть блок размером до 100 пикселов CSS размещается на 200 пикселях устройства iPhone 8. Экран iPhone X содержит 1125×2436 пикселов, но использует макетную сетку 375×812 точек (соотношение 3 пикселя устройства к одной точке — или 3x). Область размером до 100 пикселов размещается на 300 пикселях экрана iPhone X.

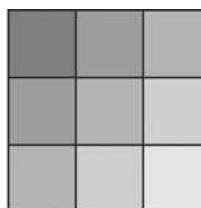
ОПОРНЫЕ ПИКСЕЛЫ

Для удобства компоновки изображения на экране устройства используют единицу измерения, называемую *опорным (эталонным) пикселом*.

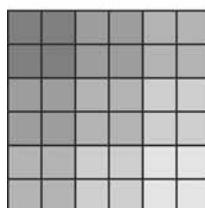
Отношение количества пикселов устройства к пикселям CSS называется *соотношением логических и физических пикселов на устройстве* (рис. 7.10). Обычные соотношения логических и физических пикселов для портативных устройств следующие: 1,325x, 1,5x, 1,7x, 2x, 2,4x, 3x и даже 4x («x» здесь — это принятное соглашение, указывающее на соотношение логических и физических пикселов на устройстве). Сейчас даже для больших настольных дисплеев характерно соотношение 2x, 3x и 4x.



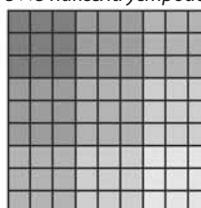
Изображение или объект =
= 3 × 3 опорных пикселя или CSS-пикселя



Соотношение логических и физических пикселов
на устройстве 1:1 (1x)
3 × 3 пикселя устройства, обозначенных сеткой



Соотношение логических и физических
пикселов на устройстве 2:1 (2x)
6 × 6 пикселя устройства



Соотношение логических и физических
пикселов на устройстве 3:1 (3x)
9 × 9 пикселя устройства

Рис. 7.10. Пиксели устройства и опорные CSS-пиксели

Допустим, имеется изображение, которое необходимо отображать при значении ширины, равном 200 пикселов, на всех мониторах. Можно создать изображение, равное 200 px в ширину (px — сокращение для пикселя), и оно будет хорошо смотреться на экранах со стандартным разрешением, но на экранах с высоким разрешением оно может оказаться немного размыто. Чтобы изображение выглядело четким на экране с соотношением логических и физических пикселов на устройстве 2x, нужно увеличить ширину этого изображения до 400 пикселов. Если же нужно, чтобы изображение выглядело четким на 3x-дисплее, его ширину следует увеличить до 600 пикселов. К сожалению, файл для больших изображений может иметь размер, который в четыре или даже больше раз превышает размер исходного файла.

Использование атрибута *srcset*

Можно избирательно направлять в браузеры большие изображения, если только они будут выводиться на больших мониторах. Для этого используется новый атрибут *srcset* известного уже нам элемента *img*. Атрибут *srcset* позволяет разработ-

АТРИБУТ *SRCSET*

Атрибут *srcset* включает список параметров изображения, выбираемых браузером.

Список состоит из двух частей: интернет-адреса (URL) изображения и x-дескриптора, который определяет целевое соотношение пикселов на устройстве. Обратите внимание, что весь список служит значением атрибута *srcset* и содержится внутри единого набора кавычек. В следующем примере показана структура значения *srcset*:

```
srcset="image-URL #x, image-URL #x"
```

Атрибут *src* все еще необходим и обычно применяется для указания значения изображения по умолчанию 1x для браузеров, которые не поддерживают атрибут *srcset*. Убедитесь, что атрибут *alt* также имеется:

```

```

Рассмотрим пример. Имеется изображение индейки, которое должно отобразиться со значением ширины, равном 200 пикселов. Для стандартного разрешения формируется изображение шириной 200 пикселов с именем *turkey-200px.jpg*. Желательно, чтобы на мониторах с высоким разрешением оно выглядело четко, поэтому добавлены еще две версии изображения: *turkey-400px.jpg* (для 2x) и *turkey-600px.jpg* (для 3x). Далее приводится разметка для добавления изображения и указания его эквивалентов с высокой четкостью при помощи x-дескрипторов:

```

```

Поскольку браузеры игнорируют имеющиеся в исходном документе символы возврата строк и пробелов, этот же код можно написать так, чтобы его легче было прочесть, как я и буду его приводить в этой главе далее:

```

```

В результате представленные варианты и сама структура становятся понятными с первого взгляда, не так ли? Браузеры, распознающие атрибут `srcset`, проверяют разрешение экрана и загружают именно то изображение, которое считают наиболее подходящим. Если браузер установлен на Mac с 2-кратным дисплеем Retina, он может загрузить изображение `image-400px.jpg`. Если соотношение пикселов устройств составляет 1,5х, 2,4х или что-либо подобное, проверяется общая среда просмотра и делается наилучший выбор. Важно знать, что, когда атрибут `srcset` применяется с элементом `img`, именно браузер делает окончательный выбор изображения.

АТРИБУТ `SRCSET` С ЭЛЕМЕНТОМ `IMG`

Если атрибут `srcset` применяется с элементом `img`, браузер имеет возможность сделать выбор наиболее подходящего изображения.

Когда применяются x-дескрипторы?

X-дескрипторы указывают браузеру, что необходимо делать выбор только на основе разрешения экрана, не учитывая размеры экрана или области просмотра. По этой причине x-дескрипторы удобнее применять для изображений, имеющих одинаковые размеры в пикселях независимо от размера экрана, — таких, как логотипы, значки социальных сетей или другие изображения фиксированной ширины.

Обычно изменение размеров изображений связано с размерами экранов, что позволяет передавать небольшие изображения на экраны портативных устройств малого размера, а большие изображения — на экраны настольных компьютеров (для этого и существуют адаптивные изображения).

Теперь, когда вы знакомы с использованием атрибута `srcset`, давайте посмотрим, как его можно применять для доставки изображений, ориентированных на экраны различных размеров. Именно это и является основной задачей атрибута `srcset`.

Изображения с изменяемой шириной (w-дескриптор)

В процессе разработки адаптивной веб-страницы, вероятнее всего, вы захотите изменять размеры изображения в зависимости от *размера окна просмотра браузера*. Этот подход известен как *выбор на основе области просмотра*. А поскольку необходимо учитывать и скорость отображения страниц, следует ограничить загрузку ненужных данных, предоставляя изображения подходящего размера.

Этой цели служат атрибуты `srcset` и `sizes`, применяемые вместе с элементом `img`. Как показано в предыдущих примерах, атрибут `srcset` предоставляет браузеру набор параметров файла изображения, но для учета ширины изображения служит *w-дескриптор* (дескриптор ширины), поддерживающий

ОБЛАСТЬ ПРОСМОТРА

Область просмотра мобильного устройства занимает весь экран. Для настольного браузера областью просмотра служит область, где отображается страница, не считая полос прокрутки и других панелей браузеров.

фактическую пиксельную ширину для каждого изображения. Применение атрибута `srcset` с `w`-дескриптором целесообразно, если обрабатываемые изображения идентичны, за исключением их размеров (другими словами, отличаются только масштабом). Рассмотрим пример использования атрибута `srcset`, который поддерживает четыре варианта изображения и с помощью `w`-дескрипторов определяет соответствующую им пиксельную ширину. Обратите внимание, что список полностью заключен в один набор кавычек:

```
srcset="strawberries-480.jpg 480w,
       strawberries-960.jpg 960w,
       strawberries-1280.jpg 1280w,
       strawberries-2400.jpg 2400w"
```

Применение атрибута `sizes`

Следует учесть, что при использовании `w`-дескрипторов также необходимо всегда применять атрибут `sizes`, который сообщает браузеру приблизительный размер

НЕОБХОДИМОСТЬ АТРИБУТА `SIZES`

Атрибут `sizes` необходим, если применяются дескрипторы ширины.

отображаемой в макете страницы. Для этого имеется серьезная причина (кроме требований спецификации), которая подробно обсуждается далее.

При загрузке в браузер HTML-документа, определяющего веб-страницу, браузер сначала просматривает весь документ и уточняет структуру его макета — его *объектную модель документа (DOM)*. После чего для получения с сервера всех необходимых изображений активизируется *предварительный загрузчик*. И, наконец, загружаются CSS и JavaScript. Таблица стилей может включать инструкции для макета и размеров изображения, но, когда браузер увидит стили, изображения уже будут загружены. Поэтому браузер должен иметь подсказку по поводу атрибута `sizes`, а именно — учесть, будет ли изображение заполнять всю ширину области просмотра или только часть этой области. Тогда предварительный загрузчик выберет из списка атрибута `srcset` нужный файл изображения.

Обратимся к самому простому сценарию: изображение служит баннером и независимо от применяемого устройства всегда должно заполнять 100% ширины области просмотра (рис. 7.11). Соответствующий код элемента `img` приводится полностью:

```

```

В приведенном примере атрибут `sizes` указывает браузеру, что, используя `vw` — единицу ширины области просмотра (наиболее распространенную единицу для атрибута `sizes`), изображение заполнит область просмотра полностью, поэтому браузер может выбрать лучшее из предложенных изображений. Например, `100vw`



Рис. 7.11. Независимо от размера изображения оно заполняет 100% ширины области просмотра

соответствует 100% ширины области просмотра, `50vw` — 50% и так далее. Также можно применять единицы `em`, `px` и несколько других единиц CSS, но часть этих величин не используются. Браузеры, не поддерживающие атрибуты `srcset` и `sizes`, выводят изображение, указанное в атрибуте `src`.

Изменение размера изображения для заполнения всей ширины браузера встречается нечасто. Скорее всего, изображение тогда является одним из компонентов адаптивного (отзывчивого) макета страницы, который изменяет размеры и выполняет их компоновку для использования доступной ширины экрана наилучшим образом. На рис. 7.12 показана боковая панель с фотографиями продуктов питания, которые занимают всю ширину экрана на небольших устройствах (слева), частично — ширину экрана на больших устройствах (в центре) и отображаются в виде трех макетов для больших окон браузера (справа).

О БРАУЗЕРАХ, НЕ ПОДДЕРЖИВАЮЩИХ АТРИБУТЫ `SRCSET` И `SIZES`

Браузеры, не поддерживающие атрибуты `srcset` и `sizes`, выводят изображение, указанное в атрибуте `src`.

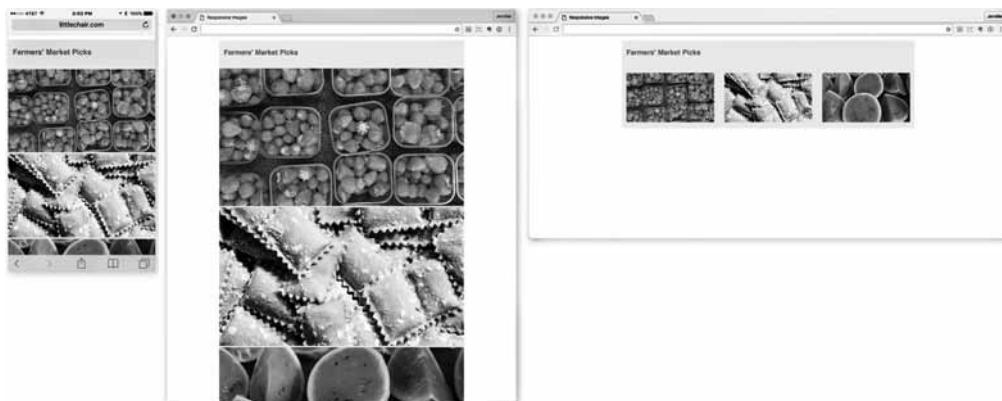


Рис. 7.12. Ширина изображения изменяется в зависимости от ширины области просмотра

В таких случаях, чтобы сообщить браузеру размер изображения для каждого макета, применяйте атрибут `sizes`. Значением атрибута `sizes` служит представленный через запятую список, где каждый пункт состоит из двух частей. В скобках сначала приводится *медиаусловие (media-feature)*, описывающее в качестве параметра `condition` ширину области. Потом указывается размер `length`, определяющий ширину, которую изображение займет в макете при выполнении медиаусловия. Вот синтаксис такой записи:

```
sizes="(media-feature: condition) length,
        (media-feature: condition) length,
        (media-feature: condition) length"
```

Я внесла в предыдущий пример некоторые медиаусловия, и далее приведен полный действительный элемент `img` для одного из фотоизображений, представленных на рис. 7.12:

```

```

Атрибут `sizes` здесь указывает браузеру следующее:

- если область просмотра имеет ширину 480 пикселов или меньше (максимальная ширина составляет 480 пикселов), изображение заполнит область просмотра на 100% ширины;
- если ширина области просмотра больше, чем 480 пикселов, но меньше 960 пикселов (максимальная ширина: 960 пикселов), изображение заполнит 70% области просмотра (этот макет имеет поля, равные 15% и для левого, и для правого полей изображения, то есть 30% области просмотра);
- если ширина области просмотра превышает 960 пикселов и не удовлетворяет ни одному из приведенных медиаусловий, размер изображения составит 240 пикселов.

Поскольку теперь браузеру известна ширина области просмотра и размер изображения для этой области, браузер может выбрать для загрузки наиболее подходящее

АТРИБУТ `SIZES` ИЛИ CSS?

Атрибут `sizes` изменяет размер изображения, только если к нему не применяется CSS. Если имеется правило CSS, задающее размер изображения, которое конфликтует со значением атрибута `sizes`, правило стиля имеет приоритет (то есть именно с учетом правила стиля переопределяется значение атрибута `sizes`).

изображение из списка, заданного атрибутом `srcset`.

Здесь были рассмотрены самые общие подходы, относящиеся к применению атрибута `sizes`, мы не коснулись вопросов, связанных с иными медиаусловиями, применением дополнительных единиц длины и даже с возможностями,

когда браузер сам рассчитывает ширину изображения. Если в проектах планируется применение изображения с учетом ширины области просмотра, то для более полного ознакомления с имеющимися возможностями желательно обратиться к спецификации.

ПРИМЕЧАНИЕ

Стратегии и инструменты, обеспечивающие формирование наборов изображений для адаптивных макетов, представлены в главе 24.

Основы принципов художественного оформления (Art Direction): элемент *picture*

`<picture>...</picture>`

Определяет количество вариантов изображения

`<source>...</source>`

Определяет альтернативные источники изображения

К этому моменту мы рассмотрели выбор изображений на основе разрешения экрана и размера области просмотра. Для обоих сценариев контент изображения не менялся — менялся только его размер.

Однако иногда изменения размера бывает недостаточно. В ряде случаев поможет, если при отображении на небольшом экране просто удалить некоторые неглавные детали изображения. Или, если изображение слишком мало, можно изменить имеющийся на нем текст, или вовсе его удалить с изображения. Может также потребоваться для разных макетов представить как панорамную (широкую), так и портретную версии изображения.

Например, представленное на рис. 7.13 полное изображение сервированного стола, а также расположенные на нем блюда хорошо смотрятся на экранах большого размера, но при просмотре этого изображения на экране смартфона весьма трудно заметить имеющиеся там удивительные по красоте детали. Поэтому желательно предоставлять пользователям альтернативные версии изображений, которые соответствуют условиями просмотра.

Подобный сценарий известен как *выбор на основе принципов художественного оформления* (*Art Direction*) — он реализуется с помощью элемента *picture*. Элемент *picture* не имеет атрибутов и представляет собой лишь оболочку для некоторого числа элементов *source* и элемента *img*. Элемент *img* является обязательным и должен фигурировать последним элементом в их наборе. Если элемент *img* пропущен, изображение вообще не отображается, потому что именно благодаря этому элементу изображение фактически помещается на странице. Приведу образец применения элемента *picture*, а затем мы рассмотрим его подробно:

```

<picture>
  <source media="(min-width: 1024px)"
    srcset="icecream-large.jpg">
  <source media="(min-width: 760px)"
    srcset="icecream-medium.jpg">
  <img alt="hand holding ice cream cone and text that reads
    Savor the Summer">
</picture>

```

В приведенном примере браузер получает сообщение о том, что если область просмотра имеет ширину 1024 пикселя или больше, то следует вывести крупную версию изображения рожка мороженого. Если область просмотра занимает по ширине больше, чем 760 пикселов (но меньше, чем 1024 — как, например, на планшете), следует вывести версию изображения среднего размера. Наконец, для областей просмотра, ширина которых не превышает 760 пикселов и, вследствие этого, области просмотра не соответствуют ни одному из медиазапросов, относящихся к предыдущим элементам источника, следует использовать уменьшенную версию изображения (рис. 7.14). Уменьшенная версия, как указано в элементе `img`, применяется для браузеров, которые не различают изображение и источник.



Это блюдо выглядит восхитительно в настольных браузерах.
(ширина 1280 px)



Детали теряются, если полное изображение уменьшить для представления на небольших устройствах. (ширина 300 px)



Может выручить обрезка с учетом сохранения лишь наиболее важной детали.
(ширина 300 px)

Рис. 7.13. Некоторые изображения становятся неразборчивыми при уменьшении их размера для отображения на экранах мобильных устройств

Каждый элемент источника содержит атрибут `media` и атрибут `srcset`. Также может присутствовать и атрибут `sizes`, хотя это и не отражено в приведенном примере. Атрибут `media` поддерживает медиазапрос для проверки текущих условий просмотра. Это напоминает медиаусловия, которые приведены в предыдущем примере `srcset`, но в рассматриваемом случае атрибут `media` формирует полнофункциональный медиазапрос CSS (подробнее о медиазапросах мы поговорим в главе 17). Атрибут `srcset` поддерживает URL для изображения, которое выводится, если имеется соответствующий медиазапрос. В приведенном примере показано только одно изображение, но также может фигурировать и список, состоящий из нескольких разделенных запятыми `x`-дескрипторов и `w`-дескрипторов.

Браузеры загружают изображение из первого `source`, которое соответствует текущим условиям, поэтому важен порядок элементов `source`. Указанный в атрибуте `srcset` URL передается атрибуту `src` в элементе `img`. Опять же, именно элемент `img` помещает изображение на страницу, поэтому не следует его пропускать. Атрибут `alt` для элемента `img` является обязательным, но этот же атрибут `alt` недопустим в элементе `source`.

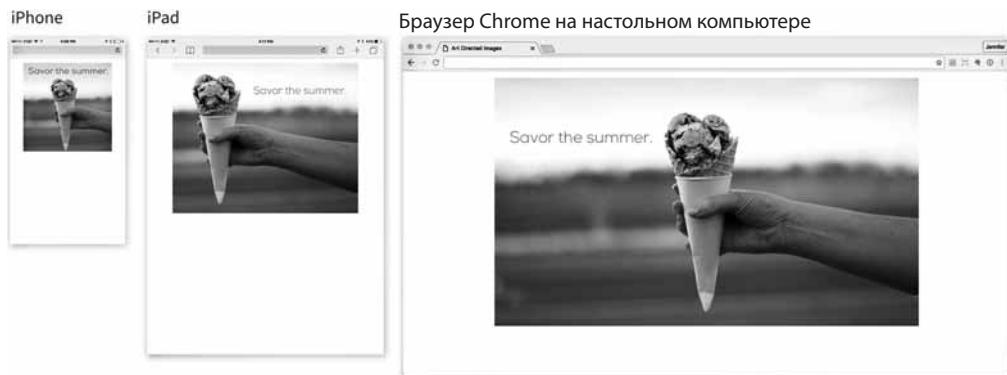


Рис. 7.14. Элемент `picture` поддерживает различные версии изображений, которые служат источниками для просмотра на экранах различных размеров

Формирование веб-страниц на основе принципов художественного оформления (Art Direction) — основное предназначение элемента `picture`, но в завершение обсуждения возможностей адаптивных изображений рассмотрим еще один вариант применения этого элемента.

Альтернативные форматы изображений: атрибут `type`

В начале 1990-х годов на веб-странице можно было размещать только GIF-изображения. Через некоторое время появилась поддержка файлов JPEG, и почти десять лет создавалась надежная поддержка браузеров для более многофункционального формата PNG. То есть требуется значительное время, чтобы новые форматы изображений стали общепринятыми. В прошлом это означало, что надо просто избегать новых форматов.

Тем не менее в целях уменьшения размеров файлов изображений были разработаны более эффективные форматы изображений — такие, как WebP, JPEG 2000 и JPEG XR, которые сжимают изображения значительно сильнее, чем аналогичные форматы JPEG и PNG. Однако некоторые браузеры их поддерживают, а некоторые — нет. Выход из ситуации заключается в том, что в наши дни можно применять элемент `picture` для отображения новых форматов изображений браузерами, которые могут их обрабатывать, и стандартного формата изображений для браузеров, которые этого не могут. И не ожидать универсальной поддержки новых форматов со стороны браузера.

ФОРМАТЫ РАСТРОВЫХ ИЗОБРАЖЕНИЙ

Форматы растровых изображений WebP, JPEG 2000 и JPEG XR будут подробно рассмотрены в главе 23.

ТИПЫ ФАЙЛОВ MIME

Для передачи информации о типе медиафайлов, передаваемых между сервером и браузером, сеть применяет стандартизированную систему. Она базируется на стандарте MIME (Multipurpose Internet Mail Extension, многоцелевые расширения почты Интернет), который изначально служил для отправки вложений по электронной почте. Любой формат файла имеет стандартизованный тип (например, `image`, `application`, `audio` или `video`), подтип, определяющий конкретный формат, а также — одно или несколько расширений файлов. В нашем примере атрибут `type` указывает параметр WebP с помощью записи `type/subtype` (`image/webp`) и применяется подходящее файловое расширение: `webp`. Другими примерами медиатипов MIME служат `image/jpeg` (расширения `jpg` и `jpeg`), `video/mp4` (расширения `mpg`, `mpe`, `mpeg`, `m1v`, `mp2`, `mp3` и `mra`), и `application/pdf` (`pdf`). Полный перечень зарегистрированных MIME-типов опубликован IANA (Internet Assigned Numbers Authority) и доступен по адресу: www.iana.org/assignments/media-types.

В следующем примере элемент `picture` определяет два альтернативных изображения в форматах WebP и JPEG XR и элемент `img`, который выводит изображение в формате JPEG в случае сбоя при отображении изображения в форматах WebP и JPEG XR:

```
<picture>
  <source type="image/webp"
srcset="pizza.webp">
  <source type="image/jxr"
srcset="pizza.jxr">
  
</picture>
```

Для выбора на основе формата изображения каждый элемент `source` имеет здесь два атрибута: атрибут `srcset`, о котором шла речь ранее, и атрибут `type` для указания типа файла (также известный как его *тип MIME* — см. врезку «*Типы файлов MIME*»). В приведенном примере первый элемент `source` указывает на изображение в формате WebP, а второй — на изображение в формате JPEG XR. Браузер выводит поддерживаемое им изображение из источника, записанного первым, поэтому желательно придерживаться возрастающего порядка — от самого маленького размера файла до самого большого.

Поддержка браузеров

Может сложиться впечатление, что при подготовке этого раздела я только и имела в виду, что поголовно все будут

использовать новые браузеры, которые поддерживают элементы `picture`, `srcset` и `sizes`, но вся проблема в том, что пользователям трудно отказаться от своих устаревших браузеров. Однако в любом случае не следует избегать применения адаптивных изображений. Весьма важно, что все способы их добавления предусматривают включение элемента `img` в качестве встроенного запасного варианта для браузеров, которые не распознают более новую разметку. В худшем случае браузер выведет изображение, указанное в элементе `img`.

Если описанных здесь мер недостаточно, попробуйте включить в веб-страницу *заполнитель изображения*. Такой заполнитель является вариантом использования сценария `polyfill`, принуждающего устаревшие браузеры поддерживать новую технологию, в рассматриваемом случае — адаптивные изображения. Этот сценарий написал Скотт Джел (Scott Jehl) из Filament Group — создатель большого числа инструментов, предназначенных для выполнения качественного дизайна и создания внешнего интерфейса. Перейдите по адресу: scottjehl.github.io/picturefill/ для загрузки сценария и ознакомления с подробным руководством о принципах его работы и методике применения.

Адаптивные изображения: подведем итоги

Хотя в этом разделе достаточно долго обсуждались особенности применения адаптивных изображений, на самом деле здесь всего лишь немного приподнята завеса над этой темой. Мы рассмотрели принципы применения элемента `img` с атрибутами `srcset` и `sizes`, что позволяет браузеру сделать выбор на основе

соотношения пикселов и размеров окна просмотра (при выполнении упражнения 7.3 вы сможете проделать все это самостоятельно). Продемонстрировано, каким образом элемент `picture` может применяться для выбора на основе принципов художественного оформления (*Art Direction*) и на основе типа изображения.

Изложение материала было проиллюстрировано небольшими и забавными примерами, но эти приемы можно комбинировать различными способами, что позволит сформировать большой массив кода, пригодный для широкого спектра изображений. Для уточнения, каким образом эти методы добавления адаптивных изображений могут объединяться при реализации более чем одного условия, рекомендуется прочесть статью Андреаса Бовенса (Andreas Bovens) «Responsive Images: Use Cases and Documented Code Snippets to Get You Started», доступную на сайте Dev.Opera (dev.opera.com/articles/responsive-images/).

Также рекомендуется состоящий из 10 частей учебник «Responsive Images 101» Джейсона Григби (Jason Grigsby) из Cloud Four. Книга содержит более детальное руководство, чем я могла вам здесь дать, приведены ссылки на другие полезные ресурсы. Начните с части 1: «Definitions» (cloudfour.com/thinks/responsive-images-101-definitions/).

ПРОВЕРКА БРАУЗЕРНОЙ ПОДДЕРЖКИ

Сайт **CanIUse.com** служит отличным инструментом для проверки поддержки браузером HTML, CSS и других веб-технологий. Введите `picture`, `srcset` или `sizes`, для того чтобы ознакомиться с особенностями их поддержки вашим браузером.

УПРАЖНЕНИЕ 7.3. ДОБАВЛЕНИЕ АДАПТИВНЫХ ИЗОБРАЖЕНИЙ

Если вы готовы реализовать некоторые из описанных здесь адаптивных технологий, загрузите последнюю версию браузера Google Chrome (google.com/chrome/) или Firefox (firefox.com), чтобы иметь твердую уверенность в том, что HTML-функции адаптивных изображений поддерживаются. Материалы упражнения доступны по адресу: learningwebdesign.com/5e/materials. Обратитесь к каталогу **responsivegallery**, в котором находится начальный HTML-файл и каталог **images**.

В этом упражнении мы создадим страницу «Black Goose Bistro Gallery» с помощью адаптивных изображений. Мы сделаем так, что вместо перехода на отдельную страницу по щелчку на миниатюре большие изображения станут появляться на странице сразу и изменять свой размер для заполнения доступного пространства. Небольшие устройства и браузеры, не поддерживающие адаптивные изображения, получат версию каждого изображения размером в 400 пикселов (рис. 7.15).

Небольшие устройства, такие, как iPhone, показывают уменьшенное за счет обрезки краев изображение шириной 400 пикселов.



В окнах просмотра размером более 480 пикселов, таких, как показанный здесь экран iPad, выводится полная версия изображения. Размеры изменяются для заполнения доступной ширины страницы между полями.



Для очень больших настольных мониторов полная версия изображения развертывается на всю доступную для нее ширину.

Браузеры, не поддерживающие адаптивное изображение, отображают его с размером в 400 пикселов, задаваемым элементом `img`.

Рис. 7.15. Страница «Black Goose Bistro Gallery» с адаптивными изображениями. Большие браузеры получают полное изображение, размеры которого могут изменяться для заполнения контента по ширине. Устройства меньшего размера воспроизводят уменьшенную версию изображения

1. С помощью текстового или HTML-редактора откройте файл `index.html`, находящийся в каталоге `responsivegallery`. В нашем случае в него был добавлен элемент `meta`, который устанавливает размер окна просмотра таким образом, чтобы он соответствовал ширине экрана устройства, что необходимо для адаптации страницы. Также внесен стиль для элементов `img`, устанавливающий максимальную ширину, равную 100% доступного пространства. Благодаря этому масштаб изображения уменьшается при уменьшении ширины экрана. Более подробно об адаптивном дизайне вы узнаете из главы 17, так что не озабочивайтесь этим сейчас. Я лишь хочу отметить здесь изменения, внесенные в предыдущее упражнение.
2. Поскольку нам надо переключаться на этой странице между горизонтальной и квадратной версиями изображения, следует применить элемент `picture`. Начните с добавления заготовки элемента `picture` в первый абзац — после строки `Our Baked Goods:` вставьте сам элемент `picture` и необходимый ему элемент `img`, который указывает на заданную по умолчанию обрезанную версию изображения (`bread-400.jpg`). Добавьте после элемента `picture` элемент разрыва строки, чтобы текст начинался на следующей строке:

```
<p>
<picture>
    
</picture>
<br>We start our day...
```

3. Это сделано для учета возможностей небольших устройств и в качестве запасного варианта для устройств, не поддерживающих адаптивные изображения. Теперь добавьте элемент `source`, который указывает браузеру на альбомную версию изображения шириной 1200 пикселов, если ширина области просмотра превышает 480 пикселов:

```
<p>
<picture>
    <source media="(min-width: 480px)"
srcset="images/bread-1200.jpg">
    
</picture>
<br>We start our day...
```

Обратите внимание, что раз в `source` указано только одно изображение, можно было бы применить простой атрибут `src`, но именно `srcset` подготавливает следующий шаг.

4. Поскольку столь большое изображение предназначено не для всех браузеров, предоставим браузеру также версию со значением ширины, равным 800 пикселов. (Полезно было бы создать большее число версий, но в этом упражнении остановимся на двух из них.) Атрибут `srcset` с помощью w-дескрипторов представляет разделенный запятыми перечень изображений с соответствующими величинами для ширины в пикселях. К исходному изображению добавлен дескриптор `1200w`, а атрибут `srcset` получил значение, равное 800 пикселов.

Наконец, примените атрибут `sizes`, тогда браузеру станет известно, что изображение займет 80% ширины области просмотра (таблица стилей добавляет 10% поля слева и справа, оставляя 80% для контента). Теперь браузер может выбрать наиболее подходящий размер:

```
<p>
<picture>
  <source media="(min-width: 480px)"
    srcset="images/bread-1200.jpg 1200w,
            images/bread-800.jpg 800w"
    sizes="80vw">
    
</picture>
<br>We start our day...
```

- Сохраните файл. На рабочем столе запустите браузер Chrome или Firefox и сократите размер окна до минимальной величины. Откройте файл `index.html` и вы увидите квадратную обрезанную версию фотографию хлеба. Медленно перетащите угол окна браузера, чтобы расширить версию изображения. Если изображение занимает по ширине больше, чем 480 пикселов, оно должно переключиться на полную версию фотографии. Если заметите небольшую метку (800), которая отображается в углу изображения, это значит, что браузер загрузил файл `bread-800.jpg`. Продолжайте расширять окно — изображение должно увеличиваться. Если заметите метку 1200, это значит, что используется файл `bread-1200.jpg`. Как только увеличенное изображение попадет в кэш браузера,

ПРОВЕРЬТЕ ПУТИ К ФАЙЛАМ И НАЛИЧИЕ НУЖНЫХ КАТАЛОГОВ

Если вы вообще не видите изображения, возможно указаны некорректные пути или на вашем компьютере отсутствует каталог `images`.

800-пиксельная версия отображаться не будет. Попробуйте снова сделать окно узким, потом широким и понаблюдайте за изменениями. Поздравляем! Теперь вы стали мастером адаптивного веб-дизайна! Переходы от узких окон к широким и составляют большую часть нашего рабочего дня.

- Добавьте на страницу два оставшихся изображения, следуя приведенному примеру. Попробуйте поэкспериментировать с различными минимальными и максимальными значениями для ширины изображений в атрибуте `media`.

КОНЕЦ — ДЕЛУ ВЕНЕЦ

На этом завершается наша работа по добавлению изображений. Речь шла о том, как размещать изображения с помощью элемента `img` и его обязательных атрибутов `src` и `alt`. Вы также узнали, что хороший альтернативный текст имеет большое значение для доступности изображений. Рассмотрено несколько способов вложения в веб-страницу SVG-изображений. Наконец, вы познакомились с новыми функциями адаптивных изображений, в том числе с атрибутами `srcset` и `sizes` для элемента

АЛЬТЕРНАТИВЫ АДАПТИВНЫМ ИЗОБРАЖЕНИЯМ

Хотя и неплохо иметь HTML-решение для вывода необходимых изображений в подходящих браузерах, представленная для рассмотрения система является весьма обременительной, поскольку включает стеки кода и характеризуется необходимостью создавать несколько изображений. Если вы работаете над сайтом, перегруженном изображениями, такой сайт может оказаться неуправляемым. Обработка изображений нуждается в автоматизации, которую лучше поручить серверу.

К счастью, имеется большое число инструментов и сервисов, как с открытым исходным кодом, так и платных, которые позволяют серверу буквально «на лету» создавать соответствующие версии изображений. Тогда загружается изображение, имеющее наибольший из доступных размеров, а сервер выполняет остальную обработку — и нет необходимости в создании и сохранении нескольких версий каждого изображения. При этом генерирование изображения относится только к изменению его размера и не имеет отношения ни к проблеме художественного оформления (*art direction problem*), ни к формированию альтернативных типов отображений. И сейчас по крайней мере уже один сервис ([Cloudinary.com](#)) использует распознавание лиц в качестве основы для обрезки изображений.

Некоторые системы управления контентом (CMS) имеют встроенные функции по изменению размера изображения. Можно также установить соответствующее парограммное обеспечение на собственный сервер. Имейте только в виду, что если для его функционирования необходим JavaScript, такой сервер будет далек от совершенства. Существует также множество сторонних решений, которые, обычно за плату, предоставляют услуги по изменению размера изображений (например, тот же [Cloudinary.com](#) и [Kraken.io](#)). Стоит обратить на них внимание при создании крупных сайтов с большим числом изображений.

Джейсон Григбси (Jason Grigsby) из Cloud Four создал электронную таблицу, содержащую сведения о программном обеспечении и услугах по изменению размеров изображения, которые служат хорошей отправной точкой для начала работы с адаптивными изображениями. Дополнительную информацию можно получить из его статьи «*Image Resizing Services*», доступной на сайте: cloudfour.com/thinks/image-resizing-services/ или по адресу tinyurl.com/pmpbyzj.

`img`, которые предназначены для представления изображений с высокой четкостью или для предоставления браузеру выбора из изображений, имеющих разные размеры, а также с элементами `picture` и `source`, предназначенными для отображения изображений с учетом принципов художественного оформления (*Art Direction*). Были рассмотрены и альтернативные форматы изображений. Теперь попробуйте ответить на несколько вопросов для проверки своих знаний.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Изображения служат важной частью веб-дизайна. Ответьте на следующие вопросы, чтобы увидеть, насколько хорошо вы усвоили ключевые понятия этой главы. Правильные ответы можно найти в *приложении 1*.

1. Какие атрибуты должны входить в каждый элемент `img`?
2. Напишите разметку для добавления изображения из файла `furry.jpg`, который находится в том же каталоге, что и текущий документ.
3. Назовите две причины включения альтернативного текста для элемента `img`.
4. В чем дело, если ваши изображения не отображаются при просмотре страницы в браузере? Имеются три возможных объяснения.
5. В чем разница между x-дескриптором и w-дескриптором?
6. В чем разница между пикселом устройства и CSS-пикселом (опорным пикселом)?
7. Сопоставьте сценарии адаптивных изображений с HTML-решениями:
 - a. ``
 - b. ``
 - c. `<picture>`
 `<source type="..." srcset=" ">`
 ``
`</picture>`
 - d. `<picture>`
 `<source media="()" srcset=" ">`
 ``
`</picture>`

_____ Вы хотите, чтобы изображение всегда заполняло всю ширину окна браузера.

_____ Вы хотите воспользоваться преимуществами сохранения файлов в формате изображения WebP.

_____ Вы хотите удалить текст с изображения при отображении его на небольших экранах.

_____ Вы хотите, чтобы изображения выглядели максимально четкими на экранах с высоким разрешением.

_____ Вы хотите показать крупным планом детали в виде изображений новостей на небольших экранах.

_____ Вы хотите, чтобы изображение уменьшилось в размере, если оно станет частью макета на большом экране.
8. Задание: скажите, о чём сообщается браузеру в следующем примере:

```
<picture>
    <source sizes="(min-width: 480px) 80vw, 100vw"
            srcset="photo-200.webp 200w
                    photo-400.webp 400w,
                    photo-800.webp 800w,
                    photo-1200.webp 1200w"
            type="image/webp">
```

```

</picture>
```

9. Что такое кэш и каким образом он влияет на производительность веб-страницы?
10. Назовите одно преимущество и один недостаток, связанные с добавлением SVG на страницу с помощью элемента `img`.
11. Назовите одно преимущество и один недостаток для встроенного SVG.
12. При каких условиях имеет смысл с помощью CSS добавлять SVG на страницу в качестве фонового изображения?
13. Что описывает этот фрагмент кода и когда может возникнуть в нем необходимость?


```
image/svg+xml
```
14. Что описывает этот фрагмент кода и где его можно обнаружить?


```
http://www.w3.org/2000/svg
```

ИСТОЧНИК ФОТОГРАФИЙ

Многие представленные в этой главе изображения взяты с удивительного по разнообразию материала фотосайта для бесплатного пользования (без роялти) [Unsplash.com](https://unsplash.com): равиoli от Davide Ragusa, бургеры от Niklas Rhöse, мороженое от Alex Jones, обеденный стол от Jay Wennington, клубника от Priscilla Fong. Из коллекции Flickr «No Rights Restrictions»: рыбное блюдо от Renata Maia, кексы от Hasma Kapouni. Остальные изображения, используемые в этой главе, являются свободно доступными.

ОБЗОР ЭЛЕМЕНТОВ: ИЗОБРАЖЕНИЯ

В табл. 7.1 приведен перечень элементов, рассмотренных при изучении разметки изображений.

Таблица 7.1. Перечень элементов, рассмотренных при изучении разметки изображений

Элемент и атрибуты	Описание
<code>img</code>	Вставляет вложенное изображение
<code>alt= "текст"</code>	Альтернативный текст
<code>src= "url-ссылка"</code>	Местоположение файла изображения
<code>srcset= "список url-ссылок с дескрипторами"</code>	Изображения, которые применяются в различных ситуациях
<code>sizes= "список медиаусловий и размеров макетов"</code>	Размеры изображений для различных макетов
<code>width= "число"</code>	Ширина графического изображения

Табл. 7.1 (окончание)

Элемент и атрибуты	Описание
<code>height = "число"</code>	Высота графического изображения
<code>usemap = "используемая карта изображения"</code>	Указывает используемую карту изображения со стороны клиента
<code>picture</code>	Контейнер, предоставляющий для включения нескольких источников для содержащегося в нем элемента <code>img</code>
<code>source</code>	<p>Поддерживает альтернативные источники для элемента <code>img</code></p> <p><code>src = "URL-ссылка"</code></p> <p><code>srcset = "URL-ссылка"</code></p> <p><code>sizes = "список размеров источников"</code></p> <p><code>media = "медиазапрос"</code></p> <p><code>type = "медиатип"</code></p>
<code>svg</code>	Добавление встроенного SVG-изображение

ГЛАВА 8

ТАБЛИЦЫ

В этой главе...

- ▶ *Использование таблиц*
- ▶ *Базовая структура таблицы*
- ▶ *Растяжение строк и столбцов*
- ▶ *Группировка строк и столбцов*
- ▶ *О доступности таблиц*

Приступая к разметке таблиц, уточним, с чем мы уже разобрались к этому моменту. Рассмотрен широкий круг вопросов: формирование базовой структуры HTML-документа, размещение текста с учетом его смысла и структуры, создание ссылок, методика вложения на страницу простых изображений.

В этой главе и двух последующих (9 и 10) рассматривается разметка специализированного контента, которая вряд ли пригодится для создания простых веб-страниц. Если вам важен внешний вид создаваемых веб-страниц, перейдите сразу к *части III* книги, где описаны возможности каскадных таблиц стилей. А к главам, посвященным созданию таблиц (*глава 8*), форм (*глава 9*) и внедрению мультимедийных элементов (*глава 10*), обращайтесь по мере надобности — тех случаях, когда возникнет необходимость в получении соответствующей информации.

Вы решили продолжить чтение книги? Вот и замечательно. В этой главе пойдет речь о таблицах. Сначала мы рассмотрим вопросы использования таблиц, а потом разберемся с применяемыми для их формирования элементами. Поскольку эта глава входит в *часть II* книги, посвященную HTML-разметке, мы сосредоточимся на разметке, которая структурирует контент в таблицы. При этом нас не слишком будет интересовать внешний вид полученных таблиц (этим мы займемся при изучении ряда глав *части III* книги, посвященным каскадным таблицам стилей).

ИСПОЛЬЗОВАНИЕ ТАБЛИЦ

HTML-таблицы предназначены для добавления на веб-страницу табличного материала — данных, которые упорядочены по строкам и столбцам. Таблицы используются для формирования расписаний, сравнения свойств продуктов, представления статистических данных или других типов информации (рис. ЦВ-8.1). Обратите

внимание, что термин «данные» не обязательно предполагает числа. Ячейка таблицы может содержать любую информацию, включая цифры, текстовые элементы, а также изображения и мультимедийные объекты.

Представление данных в строках и столбцах, свойственное визуальным браузерам, позволяет читателям мгновенно понять взаимосвязи между ячейками данных и соответствующими метками заголовков. Однако при создании таблиц имейте в виду, что некоторые читатели будут воспринимать данные на слух с помощью программы чтения с экрана или чтения по методике Брайля. Далее мы обсудим, что можно предпринять, чтобы содержимое таблиц было доступным для пользователей, которые лишены возможностей визуального восприятия.

До появления таблиц стилей именно таблицы служили единственным способом формирования макетов из нескольких столбцов, а также выполнения процедур оформления: выравнивания текста и добавления пробелов. Макеты таблиц — в частности, сложные схемы вложенных таблиц, которые когда-то были стандартной платой за веб-дизайн, — ушли в небытие, повторив судьбу птицы додо. И если вы собрались задействовать строки и столбцы лишь в целях организации представления веб-страницы, сейчас можно прибегнуть к альтернативным решениям, в которых для достижения желаемого эффекта используются возможности таблиц стилей CSS. В подходе, известном как CSS Tables (таблицы CSS), вложенные элементы `div` создают разметку, а свойства CSS Tables заставляют их выглядеть в браузере как строки и ячейки. Вы также теперь можете добиться разнообразных эффектов, которые ранее требовали применения табличной разметки, с помощью методов Flexbox и Grid Layout (см. главу 16).

Но как уже было отмечено ранее, в этой главе мы сфокусируемся на табличных элементах HTML, которые применяются для семантической разметки строк и столбцов данных согласно спецификации HTML.

МИНИМАЛЬНАЯ СТРУКТУРА ТАБЛИЦЫ

`<table>...</table>`

Контент таблицы (строки и столбцы)

`<tr>...</tr>`

Строка таблицы

`<th>...</th>`

Заголовок таблицы

`<td>...</td>`

Данные табличной ячейки

Чтобы понять, как таблица устроена, давайте сначала посмотрим на структуру следующей простой таблицы, состоящей из трех строк и трех столбцов, заполненных соответствующими данными:

Menu Item	Calories	Fat(g)
Chicken noodle soup	120	2
Caesar salad	400	6

На рис. ЦВ-8.2 представлена ее структура в соответствии с моделью HTML-таблицы. Весь контент таблицы находится в ячейках, которые имеют построчное расположение. Ячейки включают либо информацию о заголовках (заголовки столбцов — например, **Calories**), либо данные, которые могут быть любым видом контента.

Достаточно просто, не так ли? А теперь посмотрим, каким образом эта структура создается с помощью элементов (рис. ЦВ-8.3).

На рис. ЦВ-8.3 представлены элементы, идентифицирующие таблицу: **table**, строки: **tr** — для строк таблицы, ячейки: **th** — для заголовков таблицы и **td** — для данных таблицы. Как можно видеть, ячейки — это суть таблицы, поскольку именно в них содержится фактический контент. Прочие элементы служат для его представления.

Обратите внимание на отсутствие среди элементов, показанных на рис. ЦВ-8.3, элементов столбца. Это связано с тем, что количество столбцов в таблице определяется числом ячеек в каждой строке. Этот момент несколько усложняет работу с HTML-таблицами. Обработка строк вполне проста — если в таблице три строки, примените три элемента **tr**. Столбцы определяются иначе. Для представления таблицы с четырьмя столбцами необходимо, чтобы в каждой строке располагалось четыре элемента **td** или **th** (более подробно этот вопрос рассмотрен далее — в разд. «Группировка строк и столбцов»).

Исходный код разметки таблицы, представленной на рис. ЦВ-8.3, имеет следующий вид (обычно элементы **th** и **td** записывают на одном уровне, чтобы было легче находить их в исходном коде. Это не влияет на их отображение браузером):

СТАЙЛИНГ ТАБЛИЦ

Когда с помощью соответствующей разметки структура таблицы сформирована, для настройки ее внешнего вида можно добавить «слой» стиля. Используйте таблицы стилей для управления визуальным представлением таблиц. Все необходимые инструменты форматирования таблиц представлены в следующих главах книги:

- **глава 12:**
 - ◆ установки шрифтов для контента ячеек;
 - ◆ цвет текста в ячейках;
- **глава 13:**
 - ◆ фоновые цвета;
 - ◆ фоновые изображения в виде плиток;
- **глава 14:**
 - ◆ размеры таблиц (ширина и высота);
 - ◆ границы;
 - ◆ заполнение ячейки (пространство, окружающее контент ячейки);
 - ◆ поля вокруг таблицы;
- **глава 19:**
 - ◆ специальные свойства для управления границами и расстояниями между ячейками.

ПРАВДА, ЗАБАВНО?

В соответствии со спецификацией HTML5 элемент `table` может включать «в следующем порядке: необязательно, элемент `caption`, за которым следует нуль, или элементы `colgroup`, за которыми, необязательно, следует элемент `thead`, за которыми следует либо нуль, либо элементы `tbody`, либо один или несколько элементов `tr`, за которыми следует, необязательно, элемент `tfoot` (но, в целом, может быть только один дочерний элемент `tfoot`)».

Что ж, я рада, если на данном этапе вам все понятно!

```
<table>
  <tr>
    <th>Menu item</th>
    <th>Calories</th>
    <th>Fat (g)</th>
  </tr>
  <tr>
    <td>Chicken noodle soup</td>
    <td>120</td>
    <td>2</td>
  </tr>
  <tr>
    <td>Caesar salad</td>
    <td>400</td>
    <td>26</td>
  </tr>
</table>
```

Не забывайте, что собственно контент таблицы полностью содержится в ячейках, то есть в элементах `td` или `th`. В ячейку можно поместить любой контент: текст, графику или даже другую таблицу.

Начало и завершение табличного материала определяют начальный и завершающий теги `table`. Элемент `table` может включать лишь некоторое количество элементов `tr` (строк), заголовков и, необязательно, элементы группировки строк и столбцов, рассмотренные в разд. «Группировка строк и столбцов». Элемент `tr` может включать только некоторое количество элементов `td` или `th`. Другими словами, в элементе `table` не может быть текстового содержимого и элементов `tr`, которые не содержат элементов `td` или `th`.

Наконец, на рис. 8.4 показана таблица, размещенная на простой веб-странице, как она отображается в браузере по умолчанию. Я знаю, что это не столь уж красивое отображение, однако напомню, что все необходимые ее «украшательства» выполняются с помощью CSS. Стоит также отметить, что по умолчанию таблицы в браузерах всегда начинаются с новой строки.

Nutritional Information

At the Black Goose Bistro, we know you care about what you eat. We are happy to provide the nutritional information for our most popular menu items to help you make healthy choices.

Menu item	Calories	Fat (g)
Chicken noodle soup	120	2
Caesar salad	400	26

We welcome your input and suggestions for our menu. If there are any modifications you need to meet dietary restrictions, please let us know in advance and we will make every effort to accommodate you.

Рис. 8.4. Оформленное по умолчанию отображение примера таблицы в браузере

Рассмотрим исходный код другой таблицы. Можете ли вы определить количество строк и столбцов в этой таблице при отображении ее в браузере?

```
<table>
  <tr>
    <th>Burgers</th>
    <td>Organic Grass-fed Beef</td>
    <td>Black Bean Veggie</td>
  </tr>
  <tr>
    <th>Fries</th>
    <td>Hand-cut Idaho potato</td>
    <td>Seasoned sweet potato</td>
  </tr>
</table>
```

Если вы догадались, что это таблица имеет две строки и три столбца, то эта догадка верна! Два элемента `tr` создают две строки, один элемент `th` и два элемента `td` в каждой строке формируют три столбца.

ЗАГОЛОВКИ ТАБЛИЦ

Как показано на рис. 8.4, текст, помеченный как заголовок (элементы `th`), отображается иначе, чем другие ячейки таблицы (элементы `td`). В данном случае эта разница имеет значение. Заголовки таблиц предоставляют информацию о содержании контента ячеек в строке или столбце, которым предшествуют. Такой элемент может с помощью альтернативных средств просмотра обрабатываться иначе, чем элементы `td`. Например, программы чтения с экрана могут оглашать заголовок, находящийся перед каждой ячейкой данных («Menu item: Caesar salad, Calories: 400, Fat-g: 26»).

Таким образом, заголовки являются ключевым инструментом при обеспечении доступности содержимого таблицы. Не пытайтесь имитировать их, форматируя строку элементов `td` иначе, чем прочие части таблицы. И, наоборот, не пренебрегайте элементами `th` из-за их отображения по умолчанию («полужирный» шрифт, выравнивание по центру). Вместо этого пометьте заголовки семантически, а потом с помощью правил стиля внесите корректизы в представление.

Пожалуй, основа уже есть. Но, прежде чем продвигаться дальше, выполним *упражнение 8.1*.

УПРАЖНЕНИЕ 8.1. СОЗДАНИЕ ПРОСТОЙ ТАБЛИЦЫ

Попробуйте написать разметку для таблицы, показанной на рис. 8.5. Можно открыть текстовый редактор или просто записать ее на бумаге. Готовая разметка находится в папке материалов, доступной по адресу: www.learningwebdesign.com/5e/materials.

Обратите внимание, что для выделения структуры вокруг ячеек таблицы с помощью стилевого правила добавлена 1-пиксельная рамка. Если нужно, чтобы

в таблицах были границы, скопируйте этот элемент `style` в заголовок документов, которые будут создаваться для упражнений этой главы:

Album	Year
Rubber Soul	1965
Revolver	1966
Sgt. Pepper's	1967
The White Album	1968
Abbey Road	1969

Рис. 8.5. Напишите разметку для этой таблицы

```
<style>
td, th {
    border: 1px solid gray;
}
</style>
```

Обязательно закройте все элементы таблицы. Строго говоря, вы не обязаны закрывать элементы `tr`, `th` и `td`, но желательно вырабатывать у себя привычку писать аккуратный исходный код с максимальной предсказуемостью при выполнении его на всех устройствах просмотра.

РАСТЯЖЕНИЕ ЯЧЕЕК

Одной из фундаментальных особенностей структуры таблицы является возможность *растяжения ячейки* — то есть ячейка может растягиваться на несколько строк или столбцов. Растяжение позволяет создавать сложные структуры таблиц, но при этом возникает побочный эффект, который заключается в том, что такую разметку сложнее понять. Также могут возникнуть сложности для пользователей, обращающихся к программам чтения с экрана.

При формировании заголовка или диапазона ячеек данных используются атрибуты `colspan` или `rowspan`, которые рассматривается далее.

Растяжение столбцов

При *растяжении столбцов*, создаваемом с помощью атрибута `colspan` элемента `td` или `th`, ячейка растягивается вправо для охвата дополнительных столбцов (рис. 8.6). Растяжение столбцов используется для расположения заголовка над двумя столбцами (на рис. 8.6 вокруг ячеек добавлена рамка, отражающая структуру таблицы).

```
<table>
  <tr>
    <th colspan="2">Fat</th>
  </tr>
  <tr>
    <td>Saturated Fat (g)</td>
    <td>Unsaturated Fat (g)</td>
  </tr>
</table>
```

Заметим, что в первой строке (`tr`) имеется только один элемент `th`, а вторая строка содержит два элемента `td`. Элемент `th` для растягиваемого столбца не включается

Fat	
Saturated Fat (g)	Unsaturated Fat (g)

Рис. 8.6. Атрибут `colspan` растягивает ячейку вправо для охвата указанного числа столбцов

в исходный код — его представляет ячейка с атрибутом `colspan`. В каждой строке должно находиться одинаковое число ячеек или эквивалентных величин `colspan`. К примеру, если имеется два элемента `td` и величина `colspan` равна 2, то одинаковым оказывается соответствующее количество столбцов в каждой строке.

Попробуйте выполнить растяжение столбцов, выполнив [упражнение 8.2](#).

БУДЬТЕ ВНИМАТЕЛЬНЫ ПРИ РАБОТЕ СО ЗНАЧЕНИЯМИ `COLSPAN`

Если указать в атрибуте `colspan` число, превышающее число столбцов в таблице, браузеры добавят столбцы в имеющуюся таблицу, что обычно приводит к ошибкам.

УПРАЖНЕНИЕ 8.2. РАСТЯЖЕНИЕ СТОЛБЦОВ

А теперь создайте разметку для таблицы, показанной на рис. 8.7. Можно открыть текстовый редактор или записать ее на бумаге. Для отображения на рисунке структуры ячеек в код добавлены границы, но в вашей таблице их не будет, если не добавить таблицу стилей, показанную в [упражнении 8.1](#). И снова окончательная разметка находится в папке **materials**.

Некоторые наводящие советы:

- первая и третья строки указывают на тот факт, что таблица включает три столбца;
- когда ячейка перекрывается, соответствующий ей элемент `td` не отображается в таблице.

7:00pm	7:30pm	8:00pm
The Sunday Night Movie		
Perry Mason	Candid Camera	What's My Line?
Bonanza	The Wackiest Ship in the Army	

Рис. 8.7. Попрактикуйтесь с растяжением столбцов, написав разметку для этой таблицы

Растяжение строк

Созданные с помощью атрибута `rowspan` диапазоны строк функционируют так же, как интервалы столбцов, но ячейка охватывает несколько расположенных правее ее строк. В следующем примере первая ячейка таблицы занимает три строки (рис. 8.8).

```
<table>
  <tr>
```

Serving Size	Small (8oz.)
	Medium (16oz.)
	Large (24oz.)

Рис. 8.8. Атрибут `rowspan` растягивает ячейку вниз для охвата указанного числа строк

```

<th rowspan="3">Serving Size</th>
<td>Small (8oz.)</td>
</tr>
<tr>
    <td>Medium (16oz.)</td>
</tr>
<tr>
    <td>Large (24oz.)</td>
</tr>
</table>

```

И опять же обратите внимание, что в исходном коде не отображаются элементы `td` для перекрытых ячеек (первые ячейки в оставшихся строках). Выражение `rowspan="3"` подразумевает обработку ячеек для последующих двух строк, поэтому в элементах `td` нет необходимости.

Если вам понравились растянутые столбцы, вам будет интересно создать и растянутые строки, выполнив *упражнение 8.3*.

УПРАЖНЕНИЕ 8.3. РАСТЯЖЕНИЕ СТРОК

Создайте разметку для таблицы, показанной на рис. 8.9. Помните о том, что в коде таблицы не отображаются перекрывающиеся ячейки.

Некоторые наводящие советы:

- строки всегда растягиваются вниз, поэтому ячейка **oranges** входит в первую строку, даже если ее контент выровнен вертикально и по центру;
- ячейки, которые перекрываются, не отображаются в коде.

apples		pears
bananas	oranges	
lychees		pineapple

Рис. 8.9. Попрактикуйтесь с растяжением строк, написав разметку для этой таблицы

Пространство внутри ячеек и между ними

По умолчанию таблицы расширяются для охвата контента ячеек, но этого может оказаться недостаточным. В старых версиях HTML имелись атрибуты `cellpadding` и `cellspacing` для добавления пространства внутри ячеек и между ними, но эти атрибуты исключены из HTML5, поскольку относятся к устаревшей презентационной разметке. Конечно же, регулировать интервалы между ячейками таблицы удобнее с помощью таблицы стилей. Дополнительные сведения о способах указания расстояния между ячейками приведены в разд. «Стайлинг таблиц» главы 19.

ДОСТУПНОСТЬ ТАБЛИЦ

Веб-дизайнеру важно всегда учитывать, каким образом контент сайта будет восприниматься посетителями с ослабленным зрением. С помощью программы чтения с экрана особенно сложно разобраться в табличном материале, но HTML-спецификация содержит методы, позволяющие улучшить впечатление от контента страницы и сделать его более понятным.

Описание контента таблицы

```
<caption>...</caption>
```

Заголовок или описание, отображающиеся вместе с таблицей

Наиболее эффективный способ предоставить пользователям, имеющим нарушения зрения, обзор содержания таблицы — назвать ее или описать с помощью элемента `caption`. Подписи отображаются рядом с таблицей (как правило, над ней) и используются для описания контента таблицы или служат подсказками относительно ее структуры.

Если вы собираетесь использовать элемент `caption`, то он должен быть первым элементом в элементе `table`, как показано в следующем примере, где заголовок добавлен к рассмотренной нами ранее таблице:

```
<table>
  <caption>Nutritional Information</caption>
  <tr>
    <th>Menu item</th>
    <th>Calories</th>
    <th>Fat (g)</th>
  </tr>
  <!-- продолжение таблицы -->
</table>
```

Nutritional Information		
Menu item	Calories	Fat (g)
Chicken noodle soup	120	2
Caesar salad	400	26

Заголовок по умолчанию отображается над таблицей, как показано на рис. 8.10, хотя с помощью свойств таблицы стилей можно переместить его и под таблицу (`caption-side: bottom`).

Рис. 8.10. Заголовок таблицы по умолчанию отображается над таблицей

При наличии более длинных описаний можно помещать таблицу в элемент `figure` и применять для описания элемент `figcaption`. Спецификация HTML5 предлагает для описания таблиц ряд возможностей: www.w3.org/TR/html5/tabular-data.html#table-descriptions-techniques.

Связывание ячеек и заголовков

Да, заголовки представляют собой простой способ улучшения доступности к содержимому таблицы, но иногда установить соответствие между заголовками и ячейками весьма затруднительно. Например, заголовки могут быть расположены на левом или правом краю строки, а не в верхней части столбца. И хотя зрячим

пользователям с первого взгляда достаточно легко понять структуру такой таблицы, для пользователей, воспринимающих содержимое таблицы на слух, общая ее организация не будет столь ясна. В таких случаях атрибуты `scope` и `headers` позволяют авторам явно связывать заголовки с содержимым таблиц.

Атрибут `scope`

Атрибут `scope` связывает заголовок таблицы со строкой, столбцом, группой строк (так же как `tbody`) или группой столбцов, которые он определяет с помощью значений `row`, `col`, `rowgroup` или `colgroup`, соответственно. В следующем примере атрибут `scope` служит для объявления, что заголовок ячейки относится к текущей строке:

```
<tr>
  <th scope="row">Mars</th>
  <td>.95</td>
  <td>.62</td>
  <td>0</td>
</tr>
```

Эксперты по доступности рекомендуют, чтобы каждый элемент `th` включал атрибут `scope`, что позволяет четко связать ассоциированные с ним данные.

Атрибут `headers`

При обработке усложненных таблиц, где недостаточно атрибута `scope` для связывания ячеек из таблиц данных с соответствующим заголовком (например, если таблица включает несколько растянутых ячеек), атрибут `headers` используется в элементе `td`, явно связывая его со значением `id` из значения заголовка. В следующем примере контент ячейки `.38` привязан к заголовку `Diameter measured in earths`:

ЕЩЕ О НАДЕЖНОСТИ СРЕДСТВ УЛУЧШЕНИЯ ДОСТУПНОСТИ

Несмотря на наличие у таблиц расширенных функций улучшения доступности, которые использовались в спецификациях в течение многих лет, поддержка программ чтения с экрана и других вспомогательных средств весьма ненадежна. Поэтому желательно семантически размечать данные в ячейках таблицы, чтобы смысл этих данных хорошо воспринимался при последовательном прочтении источника, а именно таким образом посетители и встречаются с ними.

```
<th id="diameter">Diameter
measured in earths</th>
<!-- many other cells -->
<td headers="diameter">.38</td>
<!-- many other cells -->
```

К сожалению, поддержка возможности `id/headers` ненадежна, поэтому рекомендуется создавать таблицы так, чтобы эти функции исполнял простой атрибут `scope`.

* * *

Представленный в этом разделе материал отражает лишь малую толику того, что известно о доступности таблиц. Подробная инструкция по созданию доступных таблиц заняла бы слишком много места в книге, предназначеннной для начинающих.

Если вы желаете узнать больше, рекомендую в качестве дополнительного материала раздел «Creating Accessible Tables» в WebAIM (webaim.org/techniques/tables/data).

Впрочем, имеется еще один важный набор элементов, помогающих уточнить семантическую структуру таблицы: элементы для группировки строк и столбцов.

ГРУППИРОВКА СТРОК И СТОЛБЦОВ

Приведенные в этой главе ранее примеры таблиц весьма подробно рассмотрены, с тем чтобы дать вам разобраться в их структуре и понять методику работы с таблицами. Но таблицы зачастую весьма сложны. Обратите внимание на рис. 8.11, иллюстрирующий спецификацию CSS Writing Modes Level 3. В показанной на нем таблице имеются три группы столбцов (одна с заголовками, две другие — с двумя столбцами в каждом) и три группы строк (заголовки, данные и сноска).

Подобные варианты таблиц размечаются с помощью элементов, предназначенных для группировки строк и столбцов и предоставляющих дополнительную семантическую структуру и дополнительные средства, применяемые для стайлинга или формирования сценариев. Например, группы строк и столбцов, показанные на рис. 8.11, выделены более толстыми границами для их четкой визуализации.

'unicode-bidi' value	<i>Bidi control codes injected by 'unicode-bidi' at the start/end of 'display: inline' boxes</i>			
	'ltr'		'rtl'	
	start	end	start	end
'normal'	—	—	—	—
'embed'	LRE (U+202A)	PDF (U+202C)	RLE (U+202B)	PDF (U+202C)
'isolate'	LRI (U+2066)	PDI (U+2069)	RLI (U+2067)	PDI (U+2069)
'bidi-override'*	LRO (U+202D)	PDF (U+202C)	RLO (U+202E)	PDF (U+202C)
'isolate-override**'	FSI,LRO (U+2068,U+202D)	PDF,PDI (U+202C,U+2069)	FSI,RLO (U+2068,U+202E)	PDF,PDI (U+202C,U+2069)
'plaintext'	FSI (U+2068)	PDI (U+2069)	FSI (U+2068)	PDI (U+2069)

* The LRO/RLO+PDF pairs are also applied to the root inline box of a block container if these values of 'unicode-bidi' were specified on the block container.

Рис. 8.11. Пример таблицы с группами строк и столбцов (из спецификации CSS Writing Modes Level 3)

Элементы группировки строк

<thead>...</thead>

Группировка строк в заголовке таблицы

<tbody>...</tbody>

Группировка строк в теле таблицы

<tfoot>...</tfoot>

Группировка строк в нижнем колонтитуле таблицы

Группы строк или строки можно описать как принадлежащие верхнему или нижнему колонтитулам таблицы или телу таблицы, используя элементы `thead`, `tfoot`

и `tbody`, соответственно. Некоторые *пользовательские агенты* (другое название браузера) могут выводить строки верхнего и нижнего колонтитула в занимающих несколько страниц таблицах. При этом строки верхнего и нижнего колонтитулов выводятся на печать на каждой странице многостраничной таблицы. Разработчики также могут использовать подобные элементы для применения стилей к различным частям таблицы.

Элементы группировки строк могут включать один или несколько элементов `tr`, не включающих непосредственное текстовое содержимое. Сначала должен отображаться элемент `thead`, за ним следует любое количество элементов `tbody`, а за ними — необязательный элемент `tfoot`.

Рассмотрим разметку группы строк для таблицы, показанной на рис. 8.11 (элементы `td` и `th` скрыты для экономии места):

```
<table>
...
<thead>
<!-- заголовки в этих строках
--&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;thead&gt;
&lt;tbody&gt;
<!-- данные --&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;/tbody&gt;
&lt;tfoot&gt;
<!-- нижний колонтитул --&gt;
&lt;tr&gt;&lt;/tr&gt;
&lt;/tfoot&gt;
&lt;/table&gt;</pre>

```

ИСХОДНЫЙ КОД ТАБЛИЦЫ С РИС. 8.11

Рассмотрите исходный код таблицы, приведенной на рис. 8.11, — он доступен по адресу: www.w3.org/TR/css-writing-modes-3/#unicode-bidi (немного прокрутите вниз). Исходный код слишком длинен, чтобы выводить его на печать в тексте, но четко размечен и легок для понимания. Обратите внимание на применение всех элементов группировки строк, столбцов и показанный в предыдущем разделе атрибут `scope` для связи заголовков со строками. На этой же странице имеется несколько интересных таблиц, с которыми вам будет интересно познакомиться.

Элементы группировки столбцов

`<colgroup>...</colgroup>`

Семантически связанная группа столбцов

`<col>...</col>`

Один столбец в группе столбцов

Как известно, столбцы уже задают число ячеек (`td` или `th`) для каждой строки. Семантически группировать столбцы (и присваивать величины `id` и `class`) можно, применяя элемент `colgroup`.

Группы столбцов идентифицируются в начале таблицы, сразу после элемента `caption`, при его наличии, и они предоставляют браузеру информацию о расположении столбцов в таблице. Число столбцов, представляемых `colgroup`, указывается с помощью атрибута `span`. В следующем примере приводится группировка столбцов в начале таблицы, представленной на рис. 8.11:

```
<table>
  <caption>...</caption>
  <colgroup></colgroup>
  <colgroup span="2"></colgroup>
  <colgroup span="2"></colgroup>
  <!-- rest of table... -->
```

ЕЩЕ ОБ АТРИБУТЕ `SPAN`

Если элементы `colgroup` включают элементы `col`, в них не должен входить атрибут `span`.

И это все, что нужно сделать. Если для сценариев или стилей необходим доступ к отдельным столбцам внутри `colgroup`, идентифицируйте их с помощью элементов `col`. Код из предыдущего раздела, относящийся к группированию столбцов, может иметь и такой вид:

```
<colgroup></colgroup>
<colgroup>
  <col class="start">
  <col class="end">
</colgroup>
<colgroup>
  <col class="start">
  <col class="end">
</colgroup>
```

Обратите внимание, что элементы `colgroup` не имеют содержимого — они лишь определяют семантически релевантную структуру столбца. Пустые элементы `col` применяются как дескрипторы для сценариев или стилей, но их применение необязательно.

СОБИРАЕМ ВСЕ ВОЕДИНО

Эта глава предоставила вам вполне приличный обзор компонентов HTML-таблиц. Выполните *упражнение 8.4*, которое объединяет большую часть рассмотренного материала, — это придаст вам больше уверенности при создании таблиц.

УПРАЖНЕНИЕ 8.4. СОЗДАНИЕ СЛОЖНОЙ ТАБЛИЦЫ

Пришло время собрать воедино приобретенные вами к этому моменту навыки написания таблиц. Теперь ваша задача заключается в том, чтобы написать исходный код для таблицы, показанной на рис. 8.12.

Рассмотрим пошагово выполняемые действия.

1. Во-первых, откройте новый документ в текстовом редакторе и определите его общую структуру (элементы `DOCTYPE`, `html`, `head`, `title` и `body`). В выбранном вами каталоге сохраните этот документ под названием `table.html`.

A common header for two subheads		Header 3
Header 1	Header 2	
Thing A		data A3
Thing B		data B3
Thing C		data C3

Рис. 8.12. Создание сложной таблицы

2. Затем, в плане работы по формированию четких границ ячеек и таблицы, желательно добавить в документ несколько простых правил таблиц стилей. Не беспокойтесь, если не можете точно понять, что именно здесь происходит (хотя, интуитивно, это вполне понятно) — просто вставьте в заголовок документа элемент `style` так, как показано здесь:

```
<head>
    <title>Table Challenge</title>
    <style>
        td, th { border: 1px solid #CCC; }
        table { border: 1px solid black; }
    </style>
</head>
```

3. А теперь обратимся к формированию таблицы. Обычно я начинаю с создания структуры таблицы и добавляю в качестве заполнителей столько пустых элементов строки, сколько необходимо для завершенной таблицы. Как показано на рис. 8.12, в таблице содержатся пять строк:

```
<body>
    <table>
        <tr></tr>
        <tr></tr>
        <tr></tr>
        <tr></tr>
        <tr></tr>
    </table>
</body>
```

4. Начните с верхней строки и заполните элементы `th` и `td` слева направо, включая любые интервалы строк или столбцов, если это необходимо. А я помогу вам при заполнении первой строки.

Первая ячейка (находится в верхнем левом углу) охватывает по высоте две строки, поэтому получает атрибут `rowspan`. Здесь также присутствует элемент `th` для соответствия с оставшейся частью строки. В этой ячейке отсутствует содержимое:

```
<table>
    <tr>
        <th rowspan="2"></th>
    </tr>
```

Ячейка во втором столбце первой строки охватывает два столбца по ширине, поэтому она получает атрибут `colspan`:

```
<table>
    <tr>
        <th rowspan="2"></th>
        <th colspan="2">A common header for two
        subheads</th>
    </tr>
```

Ячейка в третьем столбце перекрыта только что добавленным атрибутом `colspan`, поэтому ее не нужно включать в разметку. Ячейка в четвертом столбце также занимает две строки:

```
<table>
  <tr>
    <th rowspan="2"></th>
    <th colspan="2">A common header for two
    subheads</th>
    <th rowspan="2">Header 3</th>
  </tr>
```

5. А теперь ваша очередь. Продолжите для оставшихся четырех строк таблицы заполнять элементы `th` и `td`. Подсказка: первая и последняя ячейки во втором ряду перекрыты. Кроме того, если текст выделен в примере полужирным шрифтом, превратите его в заголовок.
6. Для завершения контента добавьте заголовок над таблицей, используя элемент `caption`.
7. Примените атрибут `scope`, чтобы удостовериться, что заголовки **Thing A**, **Thing B** и **Thing C** связаны с соответствующими строками.
8. В заключение приайте семантическую четкость группе строк и столбцов таблицы. В таблице отсутствует `tfoot` и имеются две группы столбцов: один столбец для заголовков, остальные — для данных. Применяйте атрибут `span` (не нужно отдельно идентифицировать столбцы).
9. Сохраните свою работу и откройте файл в браузере. Таблица должна выглядеть точно так же, как представлено на рис. 8.12. Если же это не так, возвратитесь и исправьте разметку. Если у вас возникнут затруднения, обратитесь к завершенной разметке для этого упражнения, которая содержится в папке `materials`.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Ответы на следующие вопросы содержатся в *приложении 1*.

1. Из каких частей (элементов) состоит базовая HTML-таблица?
2. Какие элементы могут непосредственно включать элемент `table` (то есть потомки первого уровня)?
3. Какие элементы включают элемент `tr`?
4. Когда следует применять элемент `col` (столбец)?
5. Найдите пять ошибок в приведенной табличной разметке:

```
<caption>Primetime Television 1965</caption>
<table>
  Thursday Night
  <tr></tr>
  <th>7:30</th>
  <th>8:00</th>
  <th>8:30</th>
```

```

<tr>
  <td>Shindig</td>
  <td>Donna Reed Show</td>
  <td>Bewitched</td>
<tr>
  <colspan="2">Laredo</colspan>
  <td>Daniel Boone</td>
</tr>
</table>

```

ОБЗОР ЭЛЕМЕНТОВ: ТАБЛИЦЫ

В табл. 8.1 приведен перечень элементов, рассмотренных при изучении разметки таблиц.

Таблица 8.1. Перечень элементов, рассмотренных при изучении разметки таблиц

Элемент и атрибуты	Описание
table	Устанавливает элемент table
tr	Устанавливает строку в таблице
td	Устанавливает ячейку в строке таблицы
colspan= "число"	Число столбцов, которые охватит ячейка
rowspan= "число"	Число строк, которые охватят ячейка
headers= "название заголовка"	Связывает ячейку данных с заголовком
th	Заголовок таблицы, связанный со строкой или столбцом
abbr= "текст"	Альтернативная метка, если на заголовок ячейки ссылаются в иных контекстах
colspan= "число"	Число столбцов, которые охватят ячейка
rowspan= "число"	Число строк, которые охватят ячейка
headers= "имя заголовка"	Связывает заголовок с другим заголовком
scope= "row col rowgroup colgroup"	Связывает заголовок со строкой, группой строк, столбцом или группой столбцов
caption	Придает таблице заголовок, который отображается в браузере
colgroup	Объявляет группу столбцов
span= "число"	Число столбцов, которые охватывает группа столбцов; может не применяться, если colgroup включает элементы col
col	Объявляет столбец
span= "число"	Количество столбцов, которые охватывает столбец
tbody	Идентифицирует группу строк в теле таблицы
thead	Идентифицирует группу строк в заголовке таблицы
tfoot	Идентифицирует группу строк в колонтитуле таблице

ГЛАВА 9

ФОРМЫ

В этой главе...

- ▶ *Как работают формы?*
- ▶ *Элементы, предназначенные для добавления виджетов форм*
- ▶ *Обеспечение доступности форм*
- ▶ *Основы дизайна форм*

Весьма быстро Интернет перерос сеть страниц, предназначенных для чтения, превратившись в действенный механизм *выполнения задач*. Теперь с помощью Интернета можно совершать покупки, бронировать билеты на самолет, подписывать петиции, выполнять поиск информации, публиковать твиты... и список этих возможностей постоянно увеличивается! Подобные взаимодействия поддерживаются с помощью веб-форм.

Фактически ответом на этот переход от страниц к приложениям стала поддержка HTML5 большого числа новых элементов управления и атрибутов форм, которые облегчают их заполнение, а также упрощают задачу разработчиков по созданию этих форм. Многие задачи, которые традиционно выполнялись с помощью JavaScript, теперь реализуются исключительно на основе разметки и функций браузера. В HTML5 появился ряд новых элементов, связанных с формами, 12 новых типов ввода и большое количество новых атрибутов (все они приведены в табл. 9.2, размещенной в конце главы). Некоторые функции зависят от реализации браузера, поэтому необходимо обратить ваше внимание на элементы управления, которые могут не поддерживаться всеми браузерами.

Эта глава посвящена веб-формам и разметке, применяемой для их создания. Я также вкратце коснусь роли дизайна при создании веб-форм.

КАК РАБОТАЮТ ФОРМЫ?

Работающая форма выступает в двух ипостасях. Первая из них — отображаемая на странице форма, которая создается с помощью HTML-разметки. Такая форма состоит из полей ввода, кнопок и раскрывающихся меню (в совокупности все это называется *элементами управления формами*). Формы могут включать текст и другие элементы.

Другая ипостась веб-формы — это приложение или сценарий на сервере для обработки собранной формой информации и возврата соответствующего ответа. Такой сценарий заставляет форму *работать*. Другими словами, публикации HTML-документа с элементами формы совершенно недостаточно. Для веб-приложений и сценариев нужны навыки программирования, которые не рассматриваются в этой книге, но в приведенной далее *врезке «Как организовать работу форм?»* представлены некоторые варианты получения необходимых сценариев.

КАК ОРГАНИЗОВАТЬ РАБОТУ ФОРМ?

Если вы не являетесь программистом, не стоит озабочиваться — вам доступны несколько вариантов работы с вашими формами:

- *воспользуйтесь преимуществами хостинг-плана* — многие хостинг-планы сайтов включают доступ к сценариям, предназначенным для выполнения простых функций типа списков рассылки. Более «продвинутые» планы могут в счет ежемесячной платы за хостинг обеспечить все, что вам необходимо, для добавления на сайт полной системы корзины покупок. Вам также должны быть доступны документация или техническая поддержка, что позволит всем этим воспользоваться;
- *обратитесь к программисту* — если вам необходимо индивидуальное решение, могут потребоваться услуги программиста, обладающего навыками серверного программирования. Сообщите программисту, чего именно вы желаете достичь с помощью формы, и он предложит вам решение. Опять же, нужно убедиться, что имеется разрешение на установку сценариев на сервере в соответствии с вашим текущим планом хостинга и что сервер поддерживает выбранный язык.

От ввода данных — к ответу

Если вы собираетесь создавать и использовать веб-формы, вам нужно представлять «закулисную» организацию этого процесса. Рассмотрим пример, в котором отслеживаются этапы транзакции при использовании простой формы, подбирающей имена и адреса электронной почты для списка рассылки, но такой процесс типичен для функционирования любой формы.

1. Посетитель — пусть это будет Sally (Салли) — открывает страницу с веб-формой в окне браузера. Браузер находит элементы управления формой в разметке и отображает их с помощью соответствующих элементов управления формы на странице (рис. 9.1). В нашем случае — это два поля для ввода текста и кнопка **Submit** (Отправить).
2. Салли желает подписаться на список рассылки, поэтому вводит в поля ввода свои имя и адрес электронной почты и *отправляет* данные формы, щелкнув на кнопке **Submit**.
3. Браузер собирает введенную информацию, кодирует ее (см. далее *врезку «Немного о кодировке»*) и направляет веб-приложению, выполняющемуся на сервере.
4. Веб-приложение принимает информацию и обрабатывает (то есть выполняет запрограммированные действия). В примере имя и адрес электронной почты Салли вносятся в базу данных списка рассылки.

5. После этого веб-приложение возвращает ответ. Тип отправляемого ответа зависит от содержания и цели формы. Здесь ответ представляет собой простую веб-страницу с благодарностью за регистрацию в списке рассылки. Другие приложения могут отвечать, перезагружая страницу формы с обновленной информацией, перемещая пользователя к другой связанной странице формы или отсылая сообщение об ошибке, если форма заполнена неправильно, — причем это далеко не полный перечень возможных ответов.
6. Сервер направляет ответ веб-приложения обратно браузеру, где он и отображается. Салли видит, что форма работает, поскольку ее имя было добавлено в список рассылки.

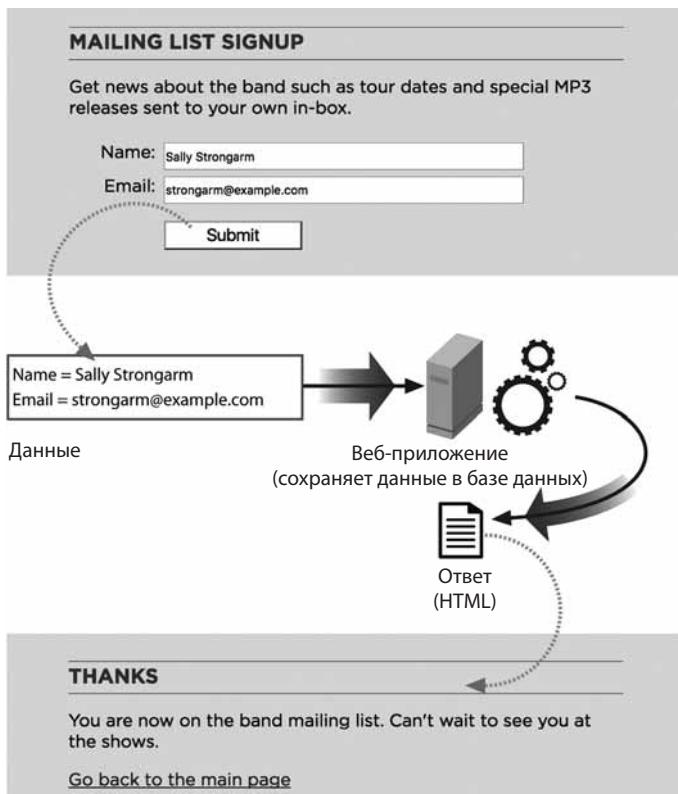


Рис. 9.1. Процедуры, выполняемые при отправке данных веб-формы

ЭЛЕМЕНТ **FORM**

<form>...</form>

Интерактивная форма

Как и следовало ожидать, формы добавляются на веб-страницы с помощью элемента `form`. Элемент `form` служит контейнером для содержимого формы, куда входят

ЭЛЕМЕНТЫ *FORM* НЕ МОГУТ ВКЛАДЫВАТЬСЯ!

Обратите внимание, что элементы *form* не могут вкладываться, а также не следует допускать их наложения. Элемент *form* должен быть закрыт еще до начала следующего элемента.

Следующий исходный код создает форму, аналогичную показанной на рис. 9.1.

```
<!DOCTYPE html>
<html>
<head>
    <title>Mailing List Signup</title>
    <meta charset=>utf-8>
</head>
<body>
    <h1>Mailing List Signup</h1>

    <form action="/mailinglist.php" method="POST">
        <fieldset>
            <legend>Join our email list</legend>
            <p>Get news about the band such as tour dates and
special MP3
releases sent to your own in-box.</p>
            <ol>
                <li><label for="firstlast">Name:</label>
                    <input type="text" name="fullname" id="firstlast"></li>
                <li><label for="email">Email:</label>
                    <input type="text" name="email" id="email"></li>
            </ol>
            <input type="submit" value="Submit">
        </fieldset>
    </form>

</body>
</html>
```

элементы управления формой, такие, как поля ввода текста и кнопки. Элемент *form* также может включать блочные элементы — например, *h1*, *p* и списки. Однако он не может содержать другой элемент *form*.

ВКЛЮЧАЙТЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ФОРМЫ В СЕМАНТИЧЕСКИЕ ЭЛЕМЕНТЫ HTML

Рекомендуется включать элементы управления формы в семантические элементы HTML — такие, как списки или элементы *div*. Упорядоченные списки, показанные в приведенном примере, весьма распространены, но достаточно часто вместе с ними используются заданные по умолчанию стили, которые нужно очищать перед оформлением подобных списков, — особенно это актуально для мобильных браузеров. Значительно улучшают доступность примененные в примере элементы *fieldset*, *legend* и *label*. Эти элементы мы подробно рассмотрим далее.

Кроме того, что элемент `form` служит контейнером для элементов управления формой, он также располагает атрибутами, которые необходимы для взаимодействия с программой обработки форм на сервере. Рассмотрим эти атрибуты внимательно.

Атрибут `action`

Атрибут `action` определяет местоположение (URL) приложения или сценария, который используется для обработки формы. Атрибут `action` в приведенном примере кода направляет данные сценарию под названием `mailinglist.php`:

```
<form action="/mailinglist.php" method="POST"> . . . </form>
```

Расширение `.php` указывает на то, что форма обрабатывается сценарием, написанным на языке сценариев PHP, но веб-формы могут обрабатываться с помощью любой из следующих технологий:

- PHP (`.php`) — служит языком сценариев с открытым исходным кодом и чаще всего применяется совместно с веб-сервером Apache. Это наиболее популярный и широко поддерживаемый вариант обработки форм;
- Microsoft ASP (Active Server Pages) — представляет среду программирования для Microsoft Internet Information Server (IIS). Расширение имени файла — `.asp`;
- Microsoft ASP.NET (Active Server Page) — сравнительно новый язык управления сценариями от Microsoft, разработанный в целях конкуренции с PHP. Расширение имени файла — `.aspx`;
- Ruby on Rails — язык программирования Ruby применяется совместно с платформой Rails. На Ruby создаются популярные веб-приложения;
- JavaServer Pages — представляет основанную на Java технологию, подобную ASP. Расширение имени файла — `.jsp`;
- Python — является популярным языком для написания сценариев для веб и серверных приложений.

Имеются и другие варианты обработки форм, файлы сценариев которых могут иметь расширения, а могут их и не иметь (как в случае с платформой Ruby on Rails). Уточните у программиста, администратора сервера или в документации сценариев правильное название и местоположение программы, которое должно быть указано в атрибуте `action`.

Иногда код обработки формы (такой, как на PHP) вложен непосредственно в HTML-файл. Тогда оставьте атрибут `action`, и форма сама разместится на странице.

УБЕДИТЕСЬ, ЧТО НУЖНЫЙ ВАМ ЯЗЫК ПОДДЕРЖИВАЕТСЯ НА ВЕБ-ХОСТИНГЕ
Если вы хотите или вам нужно работать с определенным языком обработки форм, убедитесь при покупке услуги веб-хостинга, что он там поддерживается.

Атрибут `method`

Атрибут `method` указывает, каким образом информация должна направляться серверу. Рассмотрим, как использовать этот атрибут, чтобы переслать на сервер данные, взятые из примера, приведенного на рис. 9.1.

```
fullname = Sally Strongarm  
email = strongarm@example.com
```

Когда браузер кодирует информацию для сервера, она принимает следующий вид:

```
fullname=Sally+Strongarm&email=strongarm%40example.com
```

НЕМНОГО О КОДИРОВКЕ

Данные формы кодируются так же, как и URL-адреса — пробелы и другие недопустимые символы переводятся в их шестнадцатеричные эквиваленты. Например, во введенных в форму данных каждый пробел представлен символом + или %20, а символ косой черты / (если он имеется) заменяется на %2F. Об этом вам не следует беспокоиться, поскольку обработка выполняется браузером автоматически.

Существуют два метода, применяемых для отправки закодированных данных на сервер: POST или GET, указанные с помощью атрибута method элемента form. Атрибут method не является обязательным, и по умолчанию выбирается метод GET. Далее мы рассмотрим различия между этими двумя методами. В нашем примере используется метод POST:

```
<form action="/mailinglist.php" method="POST">...</form>
```

Метод GET

С помощью метода GET данные закодированной формы прикрепляются прямо к интернет-адресу (URL), направляемому серверу. При этом от последующих данных URL отделяет знаком вопроса:

```
get http://www.bandname.com/mailnglist.php?name=Sally+Strongarm&email=strongarm%40example.com
```

Метод GET не подходит, если при отправке формы выполняется какое-либо действие — такое, как удаление каких-нибудь данных или их добавление в базу данных, поскольку, когда пользователь возвращается к форме, данные отправляются снова.

Метод POST

Если в качестве метода формы выбран POST, браузер направляет серверу отдельный запрос, содержащий некоторые специальные заголовки, за которыми и следуют данные. Теоретически только сервер видит содержание этого запроса, и, таким образом, удобно пересыпать защищенную информацию — например, домашний адрес или другие сведения личного характера. Убедитесь, что сервер поддерживает протокол HTTPS, что позволит зашифровать данные пользователя, и они будут недоступны при их передаче (протокол HTTPS был рассмотрен в главе 2).

Метод POST предпочтителен при отправке большого числа данных — таких, как длинная текстовая запись, поскольку в этом случае отсутствуют ограничения на число символов, как это имеет место для метода GET.

Метод GET подходит, если нужно разрешить пользователю добавлять в закладки результаты отправки формы (например, список результатов поиска). Поскольку в этом случае содержимое формы общедоступно, метод GET не подходит для обработ-

ки форм с личной или финансовой информацией. Кроме того, метод GET не может применяться, если форма используется для загрузки файла.

В этой главе мы будем придерживаться более распространенного метода POST.

А теперь, после ознакомления с техническими аспектами элемента `form`, обратим внимание на элементы управления формой.

ЕЩЕ О ЧУВСТВИТЕЛЬНОСТИ К РЕГИСТРУ

Методы POST и GET не являются чувствительными к регистру, но обычно по соглашению для них применяется верхний регистр. Однако в XHTML-документах значение атрибута method (post или get) должно указываться строчными буквами.

ПЕРЕМЕННЫЕ И КОНТЕНТ

В веб-формах используются различные элементы управления, которые обеспечивают возможность ввода информации или выбора между предлагаемыми вариантами. Типы элементов управления включают различные поля ввода текста, кнопки, меню и несколько элементов управления со специальными функциями. Они добавляются в документ вместе с набором элементов управления формой, которые будут рассмотрены далее — в разд. «Элементы управления формой».

Вам, как веб-дизайнеру, следует иметь представление о параметрах управления, которые сделают ваши формы более простыми и интуитивно понятными в использовании. Полезно также разобраться с механикой работы элементов управления формой.

Атрибут `name`

Каждый элемент управления формы получает один фрагмент пользовательской информации. В предыдущем примере формы поля для ввода текста получают имя посетителя и адрес его электронной почты. Говоря техническим языком, «`fullname`» (полное имя) и «`email`» (электронная почта) — это две полученные формой переменные. Введенные пользователем данные: `Sally Strongarm` и `strongarm@example.com` служат *значением* или *контентом* (содержимым) переменных.

Атрибут `name` предоставляет для того или иного элемента управления формой имя переменной. В следующем примере собранный элементом `textarea` текст определяется как переменная `comment`:

```
<textarea name="comment" rows="4" cols="45" placeholder="Leave us a comment."></textarea>
```

Если пользователь вводит в поле комментарий: `This is the best band ever!` (Это лучшая группа в истории!), серверу передается пара имя/значение (переменная/контент):

```
comment=This+is+the+best+band+ever%21
```

Все элементы управления формой должны включать атрибут `name`, что позволит приложению обработки формы выполнять сортировку информации. Вы можете, конечно, подключить атрибут `name` и для элементов кнопок `submit` и `reset`, но

АТРИБУТ `NAME`

Все элементы управления формы (кроме кнопок `Submit` и `Reset`) должны включать атрибут `name`.

это необязательно, поскольку эти элементы выполняют специальные функции (отправку или сброс формы), которые не связаны со сбором данных.

Наименование переменных

Элементы управления не могут именоваться произвольным образом. Веб-приложение по обработке данных запрограммировано на поиск определенных имен переменных. Если вы разрабатываете форму для работы с имеющимся приложением или сценарием, вам следует выяснить, какие имена переменных нужно использовать в форме, чтобы они говорили на одном языке. Допустимые имена переменных можно получить из инструкций, поставляемых с готовым сценарием на сервере, узнать у системного администратора или программиста.

Если сценарий или приложение будут созданы позднее, назовите переменные просто и наглядно и тщательно их документируйте. Кроме того, во избежание путаницы рекомендуется именовать каждую переменную уникальным образом, то есть не использовать одно и то же имя для двух переменных (однако могут быть исключения, когда это желательно). Также не следует включать в имена переменных пробелы — вместо них применяйте подчеркивание или дефис.

Итак, мы рассмотрели основы применения элемента `form` и методику именования переменных. Теперь перейдем к рассмотрению сути разметки форм: элементам управления.

ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ФОРМОЙ

А сейчас вам предстоит заняться довольно-таки интересной работой — выполнять разметку, которая добавляет на страницу элементы управления формы. В этом разделе представлены следующих элементы:

- элементы управления вводом текста;
- специализированные поля ввода текста;
- кнопки **Submit** и **Reset**;
- переключатели и флагки;
- раскрывающиеся и прокручиваемые меню;
- выбор файла и управление загрузкой;
- скрытые элементы управления;
- дата и время;
- элементы управления вводом чисел;
- элемент управления выбором цвета.

Разбираясь с этими элементами, мы на некоторое время прервемся, чтобы вы опробовали их в процессе создания формы заказа пиццы, которая показана на рис. 9.2.

Black Goose Bistro | Pizza-on-Demand

Our 12" wood-fired pizzas are available for delivery. Build your custom pizza and we'll deliver it within an hour.

— Your Information —

Name:

Address:

Telephone Number:

Email:

Delivery instructions:
No more than 400 characters long.

Design Your Dream Pizza:

— Pizza specs —

Crust (*Choose one*):

Classic white
 Multigrain
 Cheese-stuffed crust
 Gluten-free

Toppings (*Choose as many as you want*):

Red sauce
 White sauce
 Mozzarella Cheese
 Pepperoni
 Mushrooms
 Peppers
 Anchovies

Number

How many pizzas:

Рис. 9.2. Форма заказа пиццы, которая должна быть создана при выполнении упражнений этой главы

Как вы увидите, большинство элементов управления добавляются в форму с помощью элемента `input`. Функции и внешний вид элемента `input` изменяется в зависимости от значения его атрибута `type`. В HTML5.2 имеется двадцать два типа элементов управления вводом данных. Рассмотрим каждый из них по отдельности.

ПРИМЕЧАНИЕ

Атрибуты, связанные с каждым типом ввода данных, приведены в табл. 9.2, размещенной в конце главы.

Элементы управления вводом текста

Одной из наиболее распространенных задач веб-формы является ввод текстовой информации. Какой именно элемент используется для получения вводимого текста, зависит от того, необходимо вводить одну строку текста (`input`) или же несколько строк (`textarea`).

Учтите: если форма имеет поля для ввода текста, следует применять защищенный HTTPS-протокол, обеспечивающий защиту введенного пользователем контента при передаче данных на сервер (см. врезку «HTTPS — безопасный веб-протокол» для получения дополнительной информации).

Однострочное текстовое поле

`<input type="text">`

Элемент управления вводом одной строки текста

ЭЛЕМЕНТ `LABEL`

Примеры разметки, приведенные в этом разделе, включают элемент `label`, который применяется для улучшения доступности. Этот элемент подробно рассматривается в разд. «Свойства доступности форм», а пока что посмотрите на корректную разметку формы.

распознанное значение. Добавьте в форму текстовое поле ввода, вставив элемент `input` с атрибутом `type`, установленным как `text` (рис. 9.3, вверху):

```
<li><label>Favorite color: <input type="text" name="favcolor" value="Red" maxlength="50"></label></li>
```

Поле ввода текста (`input type="text"`)

City: Your Hometown

Многострочное поле для ввода текстового контента
(`input type="textarea"`)

Official contest entry:
Tell us why you love the band. Five winners will get backstage passes!

The band is totally awesome!

Многострочное поле для ввода текста с текстовым заполнителем
(`input type="textarea"`)

Official contest entry:
Tell us why you love the band. Five winners will get backstage passes!

50 words or less

Рис. 9.3. Примеры параметров управления вводом текста для веб-форм

СТИЛЬ ОТОБРАЖЕНИЯ ЭЛЕМЕНТОВ

Конкретный стиль отображения элементов управления формы зависит от операционной системы и версии браузера.

А теперь рассмотрим несколько достойных внимания атрибутов элемента `input`:

- `name` — этот атрибут применяется для указания имени переменной;
- `value` — атрибут `value` определяет заданный по умолчанию текст, который при загрузке формы отображается в поле. При сбросе формы она возвращается к этому значению. Значение атрибута `value` направляется серверу, поэтому в этом примере значение «`Red`» будет направлено вместе с формой, если пользователь не изменит его. В качестве альтернативы можно применять атрибут `placeholder`, который содержит подсказку, что именно вводить в поле, — например, `My favorite color` (Мой любимый цвет). Значение `placeholder` не направляется вместе с формой и полностью зависит от пользователя. Результаты этих действий будут показаны в следующем разделе;
- `maxlength`, `minlength` — по умолчанию пользователи могут вводить в текстовое поле неограниченное число символов, независимо от размера поля (экран будет прокручиваться вправо, если текст выходит за границы поля). Максимально допустимое количество символов можно установить с помощью атрибута `maxlength`, если оно требуется для задействованной программы обработки форм. Атрибут `minlength` указывает минимальное количество символов;
- `size` — атрибут `size` определяет длину поля ввода с использованием числа видимых символов. Однако чаще для установки размера области ввода применяется таблица стилей. По умолчанию виджет для ввода текста вмещает 20 символов.

ПОДДЕРЖКА БРАУЗЕРАМИ

Версии *Internet Explorer* до версии 11 и более ранние версии *Android* не поддерживают элемент `placeholder`.

Многострочное поле для ввода текста

`<textarea>...</textarea>`

Элемент управления вводом многострочного текста

Иногда пользователям необходимо ввести больше, чем одну строку текста. Для подобных случаев применяйте элемент `textarea`, который при отображении в браузере заменяется многострочным прокручиваемым полем ввода текста (см. рис. 9.3, в центре).

В отличие от пустого элемента `input`, в элемент `textarea` можно поместить содержимое между открывающим и закрывающим тегами. Содержимое элемента `textarea` выводится в текстовом поле, когда форма отображается в браузере. Контент также направляется серверу при отправке формы, поэтому тщательно его продумайте:

```
<p><label>Official contest entry: <br>
<em>Tell us why you love the band. Five winners will get
backstage passes!</em><br>
<textarea name="contest_entry" rows="5" cols="50">
The band is totally awesome!</textarea></label></p>
```

Атрибуты `row` и `cols` позволяют определить размер области `textarea` с помощью разметки: атрибут `rows` задает количество строк, которые отображает текстовая

область, а атрибут `cols` — ширину, представленную количеством символов (хотя для определения ширины поля чаще применяется таблица стилей CSS). Полосы прокрутки отображаются в том случае, когда пользователь вводит больше текста, чем может поместиться в выделенном пространстве.

Имеются также несколько не показанных в примере атрибутов. Атрибут `wrap` определяет, сохраняются ли при отправке формы мягкие разрывы строк (когда текст естественным образом переносится, приближаясь к краю поля). Значение `soft` (по умолчанию) не сохраняет разрывы строк. Значение `hard` сохраняет разрывы строк, если атрибут `cols` применяется для установки ширины символа поля. Атрибуты `maxlength` и `minlength` задают максимальное и минимальное число вводимых в поле символов.

Зачастую разработчики ничего не помещают между открывающим и закрывающим тегами и предоставляют подсказку с помощью атрибута `placeholder` (рис. 9.3, *внизу*). Текст заполнителя (`placeholder`), в отличие от содержимого `textarea`, не направляется серверу при отправке формы:

```
<p>Official contest entry:<br>
<em>Tell us why you love the band. Five winners will get
backstage passes!</em><br>
<textarea name="contest_entry" placeholder="50 words or less"
rows="5" cols="50"></textarea></p>
```

АТРИБУТЫ `DISABLED` И `READONLY`

Атрибуты `disabled` и `readonly` одинаково не разрешают пользователям взаимодействовать с элементом управления формой, но между ними имеются небольшие различия.

Когда элемент `form` отключен атрибутом `disabled`, этот элемент нельзя выбрать. Визуальные браузеры, по умолчанию, отображают такой элемент управления, как окрашенный серым цветом (что, конечно же, можно изменить с помощью CSS). Отключенное состояние может изменяться только с помощью сценария. Атрибут `disabled` полезен для ограничения доступа к некоторым полям формы с учетом введенных ранее данных и может применяться с любым элементом управления формы или элементом `fieldset`.

Атрибут `readonly` не разрешает пользователю изменять значение элемента управления формы (хотя его и можно выбрать). При этом разработчики могут применять сценарии для установки значений элементов управления, которые зависят от других введенных ранее в форму данных. Вводимые данные, доступные только для чтения, должны иметь определенные визуальные подсказки о своих отличиях от других вводимых данных, в противном случае они могут ввести пользователей в заблуждение, если они попытаются изменить введенные значения. Атрибут `readonly` может применяться с элементами `textarea` и с основанными на тексте элементами управления вводом (см. табл. 9.2 в конце главы).

Важное отличие рассматриваемых атрибутов друг от друга заключается в том, что поля `readonly` передаются при отправке формы, а поля `disabled` — нет.

Специализированные поля ввода текста

В дополнение к обычной однострочной текстовой записи существует ряд полей ввода, применяемых при вводе определенных типов информации, — таких, как пароли, условия поиска, адреса электронной почты, номера телефонов и интернет-адреса (URL).

Поле ввода пароля

```
<input type="password">
```

Элемент управления вводом пароля

Поле ввода пароля работает, как и поле ввода текста, за исключением скрытия вводимого текста от просмотра символами звездочки (*), маркера (•) или другого определенного браузером символа.

Важно отметить, что введенные в поле пароля символы не видны случайным наблюдателям, но форма не шифрует информацию, поэтому не следует к этому приему относиться как к действенной мере безопасности.

Далее приведен пример разметки для поля ввода пароля, а на рис. 9.4 показан результат ввода пользователем пароля в такое поле.

```
<li><label for="form-pswd">Password:</label><br>
<input type="password" name="pswd" maxlength="12"
id="form-pswd"></li>
```



Рис. 9.4. При вводе паролей браузер преобразует их на экране в символы маркера

Поиск, адрес электронной почты, номер телефона и URL-ссылка

```
<input type="search">
```

Поле поиска

```
<input type="email">
```

Адрес электронной почты

```
<input type="tel">
```

Телефонный номер

```
<input type="url">
```

Местоположение (URL-ссылка)

До появления HTML5 единственным способом ввести адрес электронной почты, телефонный номер, URL-ссылку или поисковый термин было добавление универсального поля ввода текста. В HTML5 типы ввода email, tel, url и поискового

запроса (`search`) предоставляют браузеру подсказку, какого типа информацию следует ожидать в этом поле. Указанные типы ввода используются с теми же атрибутами, что и описанный ранее общий тип ввода текста: `name`, `maxlength`, `minlength`, `size` и `value`, а также ряд других атрибутов (см. табл. 9.2 в конце главы).

Все подобные типы ввода отображаются обычно как поля ввода одностороннего текста. Но поддерживающие их браузеры могут с помощью дополнительной семантической информации реализовать особые возможности. Например, браузер Safari на iOS на основе значения типа поля ввода выводит для пользователя поддерживающую наиболее подходящую для задачи ввода клавиатуру — такую, как клавиатура с кнопкой **Search** (Поиск) для ввода поискового запроса или с кнопкой **.com**, если для типа ввода задано значение `url` (рис. 9.5). Браузеры также обычно добавляют в клавиатуру кнопку «Очистить поле» (как правило со значком в виде строчной буквы `x`). Поддерживающий типы ввода браузер может проверить введенные пользователем данные на предмет их корректности — например, убедиться, что текст, введенный как адрес электронной почты, соответствует стандартной структуре адресов электронной почты (ранее для такой проверки требовался сценарий JavaScript). Например, браузеры Opera (рис. 9.6) и Chrome отображают предупреждение, если вводимый текст не соответствует ожидаемому формату.

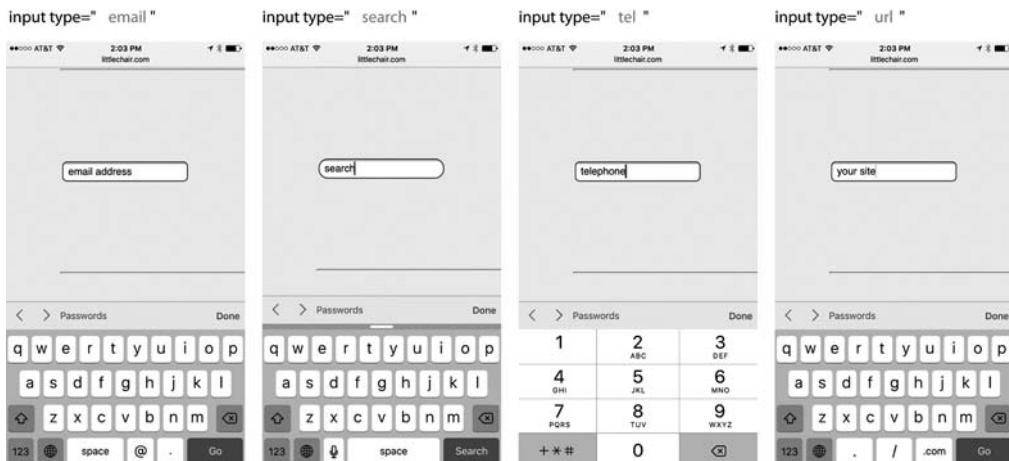


Рис. 9.5. Браузер Safari на платформе iOS на основе типа ввода отображает соответствующие пользовательские клавиатуры

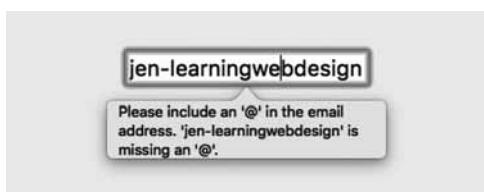


Рис. 9.6. Браузер Opera отображает предупреждение, если введенные данные не соответствуют ожидаемому формату электронной почты

ПРОВЕРЯЙТЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ФОРМЫ КОДОМ СЕРВЕРА

Значения элементов управления формы должны проверяться кодом сервера (PHP, ASP.NET и т. д.), поскольку они могут быть подвергнуты взлому или изменены. То есть, хотя проверка пользовательского ввода поддерживающими браузерами и облегчена, перед обновлением базы данных на сервере важно выполнять проверку на стороне сервера.

Впрочем, хоть современные браузеры поддерживают ввод адресов электронной почты, номеров телефонов, URL-ссылок и выполнение поиска, в способах их обработки могут присутствовать несоответствия. А устаревшие браузеры, такие, как Opera Mini и любая версия Internet Explorer до 11-й, вообще их не поддерживают, но вместо этого по умолчанию отображают поле общего текстового ввода, которое прекрасно подходит во многих ситуациях.

ЭЛЕМЕНТ УПРАВЛЕНИЯ ВВОДОМ В ВИДЕ РАСКРЫВАЮЩЕГОСЯ МЕНЮ

```
<datalist>...</datalist>
```

Раскрывающееся меню

Элемент `datalist` позволяет для любого типа текстового ввода организовать раскрывающееся меню предлагаемых значений. Пользователь получает на выбор несколько вариантов, но если не выбран ни один, пользователь может ввести собственный текст. В пределах элемента `datalist` предложенные значения отмечены как элементы `option`. Примените атрибут `list` в элементе ввода для его связывания с `id` соответствующего элемента `datalist`.

В следующем примере (рис. 9.7) элемент `datalist` предлагает несколько вариантов, позволяющих выбрать уровень образования:

```
<p>Education completed: <input type="text" list="edulevel" name="education">

<datalist id="edulevel">
  <option value="High School">
  <option value="Bachelors Degree">
  <option value="Masters Degree">
  <option value="PhD">
</datalist>
```

Поддержка браузерами списков данных пока еще не слишком распространена. Браузеры Chrome и Opera поддерживают их, но в этой поддержке имеется ошибка, из-за которой длинные списки данных невозможно прокручивать (то есть нельзя использовать прокрутку). Поэтому лучше применять эту возможность для прокрутки небольших списков. Браузеры IE11 и Edge имеют ошибки в ее реализациях, а Safari и iOS вообще ее не поддерживают. Хорошая новость состоит в том, что в случае отсутствия поддержки списков браузеры предоставляют простой текстовый ввод, что вполне приемлемо в большинстве случаев. Также для создания функционального `datalist` можно применять библиотеку Polyfill (полифилл) JavaScript.

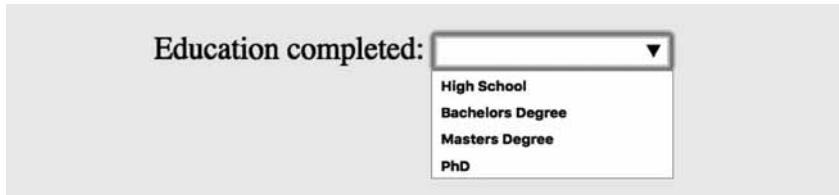


Рис. 9.7. Элемент `datalist` формирует раскрывающееся меню предлагаемых значений для поля ввода текста

Кнопки Submit и Reset

`<input type="submit">`

Отсылает данные формы серверу

`<input type="reset">`

Сбрасывает элементы управления формы до заданных по умолчанию настроек

В состав веб-форм можно добавлять несколько видов кнопок. Среди них наиболее важной является кнопка **Submit** (Отправка). В случае нажатия или щелчка на этой кнопке внесенные в форму данные немедленно направляются на сервер для обработки. Кнопка **Reset** (Сброс) возвращает элементы управления формы в состояние, в котором они находились при первоначальной загрузке формы. Другими словами, сброс формы не просто очищает все поля.

Кнопки **Submit** и **Reset** добавляются с помощью элемента `input`. Как упоминалось ранее, поскольку эти кнопки выполняют особые функции, не предусматривающие ввод данных, они являются единственными элементами управления формой, не нуждающимися в атрибуте `name`, хотя при необходимости его можно и добавить.

Кнопки **Submit** и **Reset** просты в применении — они просто размещаются в соответствующем месте формы, в большинстве случаев в самом ее конце. По умолчанию, кнопка **Submit** отображается с меткой «Submit» (Отправить) или «Submit Query» (Отправить запрос), а кнопка **Reset** — с меткой «Reset» (Сброс). Текст на кнопке можно изменить, применяя атрибут `value`, как показано в следующем примере для кнопки **Reset** (рис. 9.8).

```
<p><input type="submit"> <input type="reset" value="Start over"></p>
```

The image shows a simple web form. At the top, there are two text input fields. The first is labeled "First Name:" and the second is labeled "Last Name:". Below these fields are two buttons: a dark grey button labeled "Submit" and a light grey button labeled "Start over". The "Start over" button has a white background with black text.

Рис. 9.8. Кнопки **Submit** и **Reset**

В современном веб-дизайне кнопка **Reset** не используется в формах столь часто, как это было раньше. Это связано с тем, что сейчас для проверки правильности ввода данных форм применяется JavaScript, поэтому пользователи по мере ввода данных имеют обратную связь. При наличии продуманного дизайна и помощи все меньшее число пользователей обращаются к кнопке **Reset** по достижении конца формы. Тем не менее эта функция полезна, поэтому о ней нужно знать.

ЕЩЕ НЕСКОЛЬКО КНОПОК

Существует несколько пользовательских элементов кнопок, которые обычно упускаются при начальном ознакомлении с предметом, но для полноты изложения здесь они описаны.

Кнопки изображения

```
<input type="image">
```

Тип управления `input` позволяет заменить кнопку **Submit** изображением, выбранным пользователем. Такое изображение будет выглядеть плоским и отличающимся от трехмерной кнопки. К сожалению, этот тип кнопок не всегда доступен, поэтому обязательно предусмотрите тщательно подобранное значение атрибута `alt`.

Пользовательская кнопка ввода

```
<input type="button">
```

Если элементу `input` присвоено значение «`button`», создается кнопка, которую можно настроить с помощью JavaScript. У нее — в отличие от кнопок **Submit** и **Reset** — нет предопределенной функции.

Элемент `button`

```
<button>...</button>
```

Элемент `button` является гибким элементом, предназначенным для создания пользовательских кнопок, аналогичных тем, которые создаются с помощью элемента `input`. Контент элемента `button` (текст и/или изображения) представляет то, что отображается на кнопке.

Для получения дополнительной информации по этой теме обратитесь к статье «*When to Use the Button Element*» Крис Коуера (Chris Coyier), которая доступна на сайте: [css-tricks.com/use-buttonelement/](https://css-tricks.com/use-button-element/).

Итак, вы уже располагаете достаточным объемом знаний о разметке формы, поэтому давайте приступим к созданию формы, показанной на рис. 9.2. И начнем мы с выполнения *упражнения 9.1*, охватывающего первые шаги по созданию этой формы.

УПРАЖНЕНИЕ 9.1. ПРИСТУПАЕМ К СОЗДАНИЮ ФОРМЫ ЗАКАЗА ПИЦЦЫ

Рассмотрим следующий сценарий. Вы выполняете функции веб-дизайнера, отвечающего за создание онлайн-формы заказа пиццы для бистро «Black Goose Bistro». Владелец бистро передал вам эскиз контента формы (рис. 9.9). Имеется у вас и информация о сценарии и именах переменных, полученная от программиста.

Ваша задача заключается в том, чтобы превратить эскиз в функциональную форму. Эту задачу вам облегчит то, что простой документ с текстовым содержимым, минимальной разметкой и стилями уже создан. Этот документ — `pizza.html` — доступен в Интернете по адресу: learningwebdesign.com/5e/materials. Там же доступна и полученная форма.

Black Goose Bistro | Pizza-on-Demand

Our 12" wood-fired pizzas are available for delivery. Build your custom pizza and we'll deliver it within an hour.

Your Information

Name:

Address:

Telephone Number:

Email:

Delivery instructions:

 Limit characters and add placeholder text
 "No more than 400 characters long"

This form should be sent to
<http://blackgoosebistro.com/pizza.php>
 via the POST method.
 Name the text fields *customername*,
address, *telephone*, *email*, and
instructions, respectively.

Design Your Dream Pizza:

Pizza specs

Crust (Choose one):

() Classic white
 () Multigrain
 () Cheese-stuffed crust
 () Gluten-free

Name the controls in this section *crust*,
toppings[], and *number*, respectively.
 Note that the brackets (*[]*) after "toppings"
 are required in order for the script to
 process it correctly.

Toppings (Choose as many as you want):

[] Red sauce
 [] White sauce
 [] Mozzarella Cheese
 [] Pepperoni
 [] Mushrooms
 [] Peppers
 [] Anchovies

Make sure "red sauce" is selected when the page loads.

Number
 How many pizzas: Pull down menu for ordering up to 6 pizzas.

Bring me a pizza! Reset

Change the Submit button text.

Рис. 9.9. Эскиз формы заказа пиццы для бистро «Black Goose Bistro»

1. В текстовом редакторе откройте файл pizza.html.
2. Первое, что необходимо сделать, — полностью поместить поле вступительного абзаца в раздел формы. Программист оставил указания по поводу элементов *action* и *method*, применяемых в этой форме. Полученный элемент *form* должен иметь следующий вид (запишите его в одну строку):

```
<form action="http://www.blackgoosebistro.com/
pizza.php" method="POST">
...
</form>
```

3. В этом пункте мы работаем с разделом формы *Your information* (Ваша информация). Начнем обработку с четырех небольших элементов управления для ввода текста, которые размечены как неупорядоченный список. Далее показан первый подобный элемент, а вам необходимо вставить три остальных элемента:

```
<li>Name: <input type="text" name="customername">
</li>
```

Совет: выберите наиболее подходящий тип ввода для каждого поля ввода. Обязательно назовите элементы ввода так, как указано программистом.

4. После раздела *Delivery instructions* (Инструкции по доставке) добавьте разрыв строки и многострочное текстовое поле. Поскольку для этой формы не создается таблица стилей, применяйте разметку, чтобы создать раздел длиной в четыре строки и шириной 60 символов (на практике предпочитают применять таблицу стилей CSS, поскольку так легче управлять процессом):

```
<li>Delivery instructions:<br>
<textarea name="instructions" rows="4" cols="60"
maxlength="400" placeholder="No more than 400
characters long"></textarea></li>
```

5. Пропустим оставшуюся часть формы до появления следующих элементов управления, однако вы можете добавить кнопки отправки (*submit*) и сброса (*reset*) данных в конце, непосредственно перед тегом *</form>*. Обратите внимание, что для кнопки **Submit** изменен текст.

```
<p><input type="submit" value="Bring me a pizza!">
<input type="reset"></p>
```

6. Сохраните документ и откройте его в браузере. Контент документа должен выглядеть так, как показано на рис. 9.2. Если это не так, устранитите имеющиеся несоответствия.

Как только документ примет нужный вид, проверьте его, вводя некоторую информацию и отправляя данные формы. В результате должен быть получен ответ, подобный приведенному на рис. 9.10. Что ж, сценарий *pizza.php* работает, но пока что пиццу вам не доставят.

THANK YOU

Thank you for ordering from Black Goose Bistro. We have received the following information about your order.

Your Information

Name: Jennifer Robbins
Address: 123 Street
Telephone number: 555-1212
Email Address: jen@example.com

Delivery Instructions: Ring the middle buzzer. If nobody answers, text me.

Your pizza

Crust: white
Toppings: red sauce, mozzarella, pepperoni, mushrooms
Number: 1

This site is for educational purposes only. No pizzas will be delivered.

Рис. 9.10. Если ваша форма работает, вы увидите подобную ответную страницу. Поля описания пиццы будут вноситься позже, в последующих упражнениях, поэтому сейчас они выглядят «пустыми»

Переключатели и флажки

Флажки (checkbox) и переключатели (radio buttons) обеспечивают посетителям страницы возможность выбора одного из вариантов. Они похожи тем, что функционируют как небольшие переключатели «включить/выключить», которые могут выбираться пользователем, а также добавляются вместе с элементом `input`. Но функции, выполняемые флажками и переключателями, различаются.

Состоящий из набора переключателей элемент управления формы `radio buttons` применяется, если разрешен только один вариант из группы, — иными словами, если результаты выбора взаимно исключают друг друга, например: «Да или Нет» (Yes or No) или «Получение или доставка» (Pick-up or Delivery). Когда один переключатель «включен», остальные должны быть «выключены» — здесь применяется способ работы с переключателями, используемый на старых радиоприемниках: нажмите кнопку одного канала, остальные — отключатся.

ЭЛЕМЕНТЫ `FIELDSET` И `LABEL`

Из примеров кода переключателей, флажков и меню для максимального упрощения структуры разметки я исключила элементы `fieldset` и `label`. Далее в разд. «Функции доступности форм» вы узнаете о роли этих элементов в разметке для всех элементов формы.

Если же сгруппировать флажки (checkbox), можно будет выбрать произвольное их количество. Подобный подход удобно применять при обработке списков, когда нужно одновременно выбрать несколько вариантов.

Переключатели

```
<input type="radio">
```

Переключатель

Для добавления переключателей в форму используется элемент `input`, атрибуту `type` которого присваивается значение «`radio`». Рассмотрим синтаксис переключателя с минимальным набором функций:

```
<input type="radio" name="variable" value="value">
```

Атрибут `name` является обязательным — он играет важную роль в связывании в набор нескольких вариантов переключателя. Если нескольким переключателям назначить ввод одинакового именованного значения (в следующем примере значением служит «`age`» (возраст)), они формируют группу взаимодополняющих вариантов.

```
<p>How old are you?</p>
<ol>
  <li><input type="radio" name="age" value="under24" checked>
    under24</li>
  <li><input type="radio" name="age" value="25-34"> 25 to 34</li>
  <li><input type="radio" name="age" value="35-44"> 35 to 44</li>
  <li><input type="radio" name="age" value="over45"> 45+</li>
</ol>
```

В этом примере переключатели применяются как интерфейс для пользователей, позволяя вводить данные о своей возрастной группе. Пользователь не может

принадлежать более чем к одной возрастной группе, поэтому переключатели здесь вполне уместны. На рис. 9.11, слева показано, каким образом в окне браузера отображаются переключатели.

Radio buttons (`input type="radio"`) Checkboxes (`input type="checkbox"`)

<p>How old are you?</p> <ul style="list-style-type: none"> <input checked="" type="radio"/> under 24 <input type="radio"/> 25 to 34 <input type="radio"/> 35 to 44 <input type="radio"/> 45+ 	<p>What type of music do you listen to?</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Punk rock <input checked="" type="checkbox"/> Indie rock <input type="checkbox"/> Hip Hop <input type="checkbox"/> Rockabilly
---	---

Рис. 9.11. Переключатели (слева) применимы, если разрешен только один вариант выбора. Флажки (справа) удобнее применять, если пользователи делают выбор между любым числом вариантов: от одного до всех предложенных

Обратите внимание, что все элементы `input` имеют одинаковое значение атрибута `name`, равное «`age`», но их значения различны. Поскольку речь идет о переключателях, за один прием можно проверить только один переключатель, и, следовательно, при отправке формы на сервер для обработки направляется только одно значение.

Для уточнения, какой именно переключатель проверяется при загрузке формы, в элемент `input` добавляют атрибут `checked`. В приведенном примере флажок, соответствующий значению `under 24`, будет при загрузке страницы устанавливаться по умолчанию.

Флажки

`<input type="checkbox">`

Флажок

Для добавления флажков используется элемент `input`, атрибуту `type` которого присваивается значение `checkbox`. Как и с переключателями, для создания группы флажков каждому флажку присваивается одно и то же значение атрибута `name`. Разница, как уже отмечалось, в том, что одновременно можно устанавливать несколько флажков. При отправке формы значение каждого отмеченного флажка направляется на сервер. Рассмотрим пример группы кнопок-флажков, применяемых для описания музыкальных предпочтений (на рис. 9.11, справа показано, какой вид они имеют в браузере):

```
<p>What type of music do you listen to?</p>
<ul>
  <li><input type="checkbox" name="genre" value="punk" checked>
    Punk rock</li>
```

АТРИБУТ `CHECKED`

Может показаться, что атрибут `checked` не имеет значения, но это один из атрибутов HTML, сводимых к одному слову. Хотя на самом деле запись этого атрибута весьма избыточна:

`checked= "checked"`

Одно из правил более строгого синтаксиса XHTML гласит, что атрибуты таким образом минимизированы быть не могут.

```
<li><input type="checkbox" name="genre" value="indie" checked> Indie rock</li>
<li><input type="checkbox" name="genre" value="hiphop">
Hip Hop</li>
<li><input type="checkbox" name="genre" value="rockabilly">
Rockabilly</li>
</ul>
```

Флажки, конечно, не обязательно собираются в группы. Например, в следующем примере устанавливается один флажок, разрешая посетителям участвовать в специальных акциях. Значение этого элемента управления передается на сервер только в том случае, если пользователь установит флажок.

```
<p><input type="checkbox" name="OptIn" value="yes"> Yes, send me news and special promotions by email.</p>
```

В разметке флажков также применяют атрибут `checked`, чтобы предварительно выбрать их при загрузке формы.

Вернемся к нашим упражнениям и в *упражнении 9.2* добавим в форму заказа пиццы переключатели и флажки.

УПРАЖНЕНИЕ 9.2. ДОБАВЛЕНИЕ ПЕРЕКЛЮЧАТЕЛЕЙ И ФЛАЖКОВ

В следующем разделе формы заказа пиццы «Black Goose Bistro» для выбора вариантов пиццы применяются переключатели и флажки. Откройте документ **pizza.html** и выполните следующие действия.

1. В разделе *Design Your Dream Pizza* (Создай пиццу своей мечты) представлены перечни вариантов основы пиццы и ее начинки.

В качестве параметров *Crust* (Основы) задействованы переключатели, потому что заказанная пицца может иметь только одну основу. Перед каждым параметром установите переключатель. Следуйте этому примеру при выборе остальных вариантов основы пиццы:

```
<li><input type="radio" name="crust" value="white"> Classic white</li>
```

2. Разметьте параметры *Toppings* (Начинки), как это реализовано для параметров *Crust*, но на сей раз выбирается тип флажки. Убедитесь, что названием переменной для каждой опции служит *toppings* [] и предварительно выбран вариант *Red sauce* (красный соус) (`checked`), как указано в эскизе (см. рис. 9.9).
3. Сохраните документ и проверьте результат работы, открыв его в браузере. Убедитесь, что он имеет требуемый вид, а затем отправьте данные формы, чтобы убедиться, что она выполняет свои функции.

Меню

```
<select>...</select>
```

Элемент управления меню

```
<option>...</option>
```

Вариант, выбираемый в меню

```
<optgroup>...</optgroup>
```

Логическая группировка вариантов меню

Другим способом для представления списка вариантов является их размещение в раскрывающемся (или прокручиваемом) меню. Меню, как правило, компактнее по сравнению с группами переключателей и флажков.

Для добавления в форму раскрывающегося или прокручиваемого меню используется элемент `select`. Характер представления меню (раскрывающееся меню или прокручиваемое) зависит от указания размера, а также от того, разрешен ли выбор более одного варианта меню. Рассмотрим меню обоих типов.

Раскрывающиеся меню

Элемент `select` по умолчанию отображается как *раскрывающееся меню* (также его называют *выпадающим меню*), если размер не указан или атрибуту `size` присвоено значение 1. В выпадающих меню можно выбирать только один пункт. Рассмотрим соответствующий пример (рис. 9.12):

```
<p>What is your favorite 80s band?  
<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>
```



What is your favorite 80s band? The Cure ▾

Рис. 9.12. Раскрывающиеся меню открываются, если пользователь щелкает на стрелке или поле ввода

Можно заметить, что элемент `select` служит контейнером для определенного количества элементов `option`. Контент выбранного элемента `option` — это содержимое, передаваемое веб-приложению при отправке формы. Если по какой-либо причине

нужно отправить значение, отличное от значения, отображаемого в меню, для представления переопределющего значения применяйте атрибут `value`. Например, если кто-то из примера меню выберет **Everything But the Girl** (Все, кроме девушки), форма отправит для переменной `EightiesFave` значение «EBTG». Для остальных же вариантов контент, заключенный между тегами `option`, будет отправлен как значение.

Прокручиваемое меню

Для представления меню в виде прокручиваемого списка укажите число отображаемых строк, используя атрибут `size`. Следующий пример меню имеет те же параметры, что и предыдущий, за исключением того, что оно настроено для отображения в виде прокручиваемого списка из шести строк (рис. 9.13):

```
<p>What 80s bands did you listen to?
<select name="EightiesBands" size="6" multiple>
    <option>The Cure</option>
    <option>Cocteau Twins</option>
    <option selected>Tears for Fears</option>
    <option 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>
```

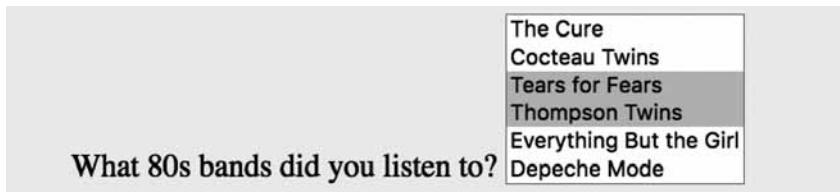


Рис. 9.13. Прокручиваемое меню с несколькими выбранными параметрами

Можно заметить здесь несколько минимизированных атрибутов. Атрибут `multiple` разрешает пользователям выбирать в меню прокрутки более одного пункта. Обратите внимание, что раскрывающиеся меню не допускают множественного выбора, но, когда браузер обнаруживает атрибут `multiple`, по умолчанию отображается небольшое прокручиваемое меню.

Используйте атрибут `selected` элемента `option`, чтобы для элемента управления меню сделать его значением, заданным по умолчанию. При загрузке формы выделяются выбранные варианты. В раскрывающихся меню может также применяться атрибут `selected`.

Группировка вариантов выбора меню

Элемент `optgroup` может применяться для создания концептуальных групп вариантов выбора. Требуемый для этого атрибут `label` предоставляет заголовок группы.

В следующем примере и на рис. 9.14 показано, каким образом группы вариантов выбора меню отображаются в современных браузерах:

```
<select name="icecream" size="7" multiple>
  <optgroup label="traditional">
    <option>vanilla</option>
    <option>chocolate</option>
  </optgroup>
  <optgroup label="fancy">
    <option>Super praline</option>
    <option>Nut surprise</option>
    <option>Candy corn</option>
  </optgroup>
</select>
```



Рис. 9.14. Группы вариантов выбора меню

При выполнении *упражнения 9.3* мы применим элемент `select`, чтобы клиенты быстро «Black Goose Bistro» могли выбрать количество заказываемых пицц.

УПРАЖНЕНИЕ 9.3. ДОБАВЛЕНИЕ МЕНЮ

Только один элемент управления требует добавить в форму заказа раскрывающееся меню — для выбора количества доставляемых пицц.

1. Вставьте элемент меню `select`, позволяющий заказывать от 1 до 6 пицц:

```
<p>How many pizzas:<br/>
<select name="pizzas">
  size="1">
  <option>1</option>
  <!-- more options here -->
</select>
</p>
```

2. Сохраните документ и просмотрите его в браузере. Также можно отправить форму и убедиться, что она работает. Вы должны получить ответную страницу **Thank You** (Спасибо) со всей информацией, введенной вами в форму.

Поздравляем! Вы создали первую рабочую веб-форму. А при выполнении *упражнения 9.4* мы добавим разметку, которая сделает форму более доступной для вспомогательных средств.

АТРИБУТ `LABEL` ЭЛЕМЕНТА `OPTGROUP`

Атрибут `label` элемента `optgroup` отличается от элемента `label`, применяемого для улучшения доступности (см. в этой главе далее).

Элемент управления выбором файлов

```
<input type="file">
```

Поле выбора файла

Веб-формы могут выполнять и другие функции, а не просто получать данные. В частности, веб-формы можно применять для получения каких-либо документов с жесткого диска компьютера пользователя. Например, типография может задействовать веб-форму для загрузки эскиза визитки. Какой-либо журнал может обращаться к форме, запрашивающей присылку цифровых фотографий для участия в фотоконкурсе.

Элемент для управления выбором файлов позволяет выбрать на жестком диске документ для отправки его адресату с использованием возможностей формы. Добавим его в форму с помощью нашего «старого знакомца» — элемента `input`, со значением атрибута `type`, равным `file`.

Приведенный далее пример разметки демонстрирует элемент управления выбором файла, применяемый при отправке фотографий (рис. 9.15):

```
<form action="/client.php" method="POST" enctype="multipart/  
form-data">  
    <label>Send a photo to be used as your online icon  
<em>(optional)</em><br>  
    <input type="file" name="photo"></label>  
</form>
```

Виджет загрузки файлов немного отличается (в зависимости от браузера и операционной системы), но, как правило, речь идет о кнопке, позволяющей получить доступ к системе организации файлов на компьютере (см. рис. 9.15).

Элемент управления выбором файла (в браузере Chrome)

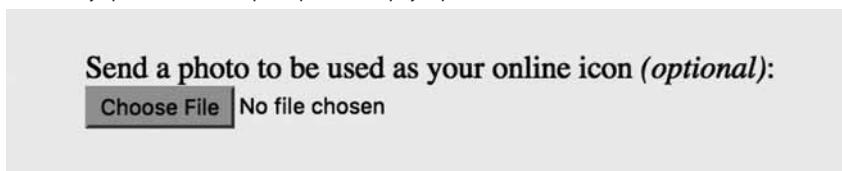


Рис. 9.15. Поле формы для выбора файла

Важно отметить, что, если форма содержит элемент ввода для выбора файла, следует задающему тип кодировки атрибуту `enctype` элемента `form` присвоить значение `multipart/form-data` и применить метод `POST`.

Элемент управления выбором файла имеет несколько атрибутов. Атрибут `accept` предоставляет браузеру информацию о том, какие типы файлов могут быть восприняты (аудио, видео, изображения или какой-либо другой формат, определяемый его медиатипом). Благодаря атрибутам `multiple` можно выбрать для загрузки нескольких файлов. Атрибут `required`, как и следует из названия, требует выбора файла.

Скрытые элементы управления

```
<input type="hidden">
```

Скрытое поле управления

Иногда нужно направить приложению для обработки форм данные, которые не вводились пользователем. При этом можно применить скрытый элемент управления формы, который при отправке формы посыпает эти данные, но остается невидимым при отображении формы в браузере.

Скрытые элементы управления добавляются с помощью элемента `input`, параметр `type` которого установлен со значением `hidden`. Единственной целью является передача на сервер при отправки формы пары `name/value` (имя/значение). В следующем примере скрытый элемент формы применяется для размещения соответствующего благодарственного документа (`thankyou.html`), который отображается после завершения транзакции:

```
<input type="hidden" name="success-link"  
value="http://www.example.com/thankyou.html">
```

Я работала с формами, в которых элемент `form` содержал десятки скрытых элементов управления, предшествующих тем частям формы, которые фактически заполнялись пользователем. Речь идет об информации, сформированной программистом приложения, системным администратором или вашим помощником при обработке форм. Если вы задействуете подобный сценарий, обязательно ознакомьтесь с прилагаемыми к нему инструкциями, чтобы понять, нужны ли какие-либо скрытые переменные формы.

ДОСТУП К СКРЫТИМ ЭЛЕМЕНТАМ УПРАВЛЕНИЯ

Пользователи могут получить доступ к скрытым элементам управления формой и управлять ими. Став профессиональным веб-разработчиком, вы научитесь защищаться от подобных опасностей.

Элементы управления датой и временем

```
<input type="date">
```

Элемент управления вводом даты

```
<input type="time">
```

Элемент управления вводом времени

```
<input type="datetime-local">
```

Элемент управления вводом даты/времени

```
<input type="month">
```

Указывает месяц в году

```
<input type="week">
```

Указывает определенную неделю в году

Если вам приходилось бронировать в Интернете отель или покупать авиабилеты, вы помните, что для выбора даты применялся небольшой виджет календаря. Скорее

всего, такой маленький календарь был создан с помощью JavaScript. В HTML5 появились шесть новых типов ввода данных, которые превращают виджеты выбора даты и времени в часть стандартных встроенных возможностей отображения браузера, точно такие же, как отображаемые браузерами флагки, всплывающие меню и другие виджеты. На момент подготовки книги средства выбора даты и времени реализованы только в нескольких браузерах (Chrome, Microsoft Edge, Opera, Vivaldi и Android), но в тех браузерах, которые не поддерживают специальные типы ввода даты и времени, они отображаются в виде простого удобного поля для ввода текста. На рис. 9.16 показаны виджеты даты и времени, отображаемые в Chrome на macOS.

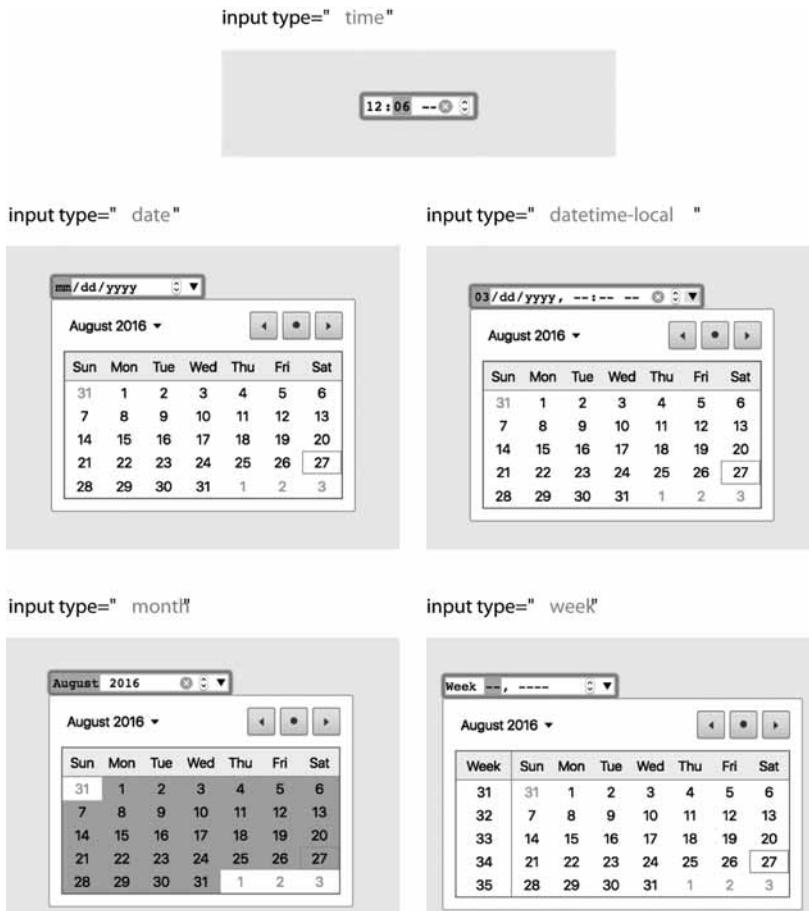


Рис. 9.16. Виджеты для выбора даты и времени (показаны в Chrome на macOS)

Новыми типами ввода даты и времени являются:

- <input type="date" name="name" value="2017-01-14">

Создание элемента управления вводом даты — например, всплывающего календаря для указания даты (год, месяц, день). Начальное значение должно быть указано в формате даты ISO: YYYY-MM-DD (ГГГГ-ММ-ДД).

- `<input type="time" name="name" value="03:13:00">`

Создание элемента управления вводом времени для ввода значения времени (часы, минуты, секунды, дробные части секунды) без указания часового пояса. Значение предоставляется в формате hh:mm:ss (чч:мм:сс).

- `<input type="datetime-local" name="name" value="2017-01-14T03:13:00">`

Создание комбинированного элемента управления вводом даты/времени без информации о часовом поясе. Значение предоставляется в формате YYYY-MM-DDThh:mm:ss (ГГГГ-ММ-ДДчч:мм:сс).

- `<input type="month" name="name" value="2017-01">`

Создание элемента управления вводом даты, который определяет конкретный месяц в году. Значение предоставляется в формате YYYY-MM (ГГГГ-ММ).

- `<input type="week" name="name" value="2017-W2">`

Создание элемента управления вводом даты для указания конкретной недели в году с использованием формата нумерации недель ISO: YYYY-W# (ГГГГ-#НЕДЕЛИ).

ОБ АТРИБУТЕ `value`

Атрибут `value` не является обязательным, но может включаться для предоставления в виджете начальной даты или времени. Здесь он включен для демонстрации формата даты и времени.

Элементы управления вводом чисел

`<input type="number">`

Элемент управления вводом чисел

`<input type="range">`

Элемент управления вводом в виде ползунка

Элементы управления вводом чисел `number` и `range` принимают числовые данные. Для элемента `number` браузер предоставляет виджет счетчика со стрелками, направленными вверх и вниз (рис. 9.17, *вверху*), с помощью которых можно выбрать определенное числовое значение (в браузерах, которые не поддерживают элемент управления вводом чисел, вместо него, как и в прочих аналогичных ситуациях, может отображаться простой элемент управления вводом текста). Элемент управления вводом `range` обычно отображается в виде ползунка (рис. 9.17, *внизу*), который позволяет выбрать значение в указанном диапазоне:

```
<label>Number of guests <input type="number" name="guests" min="1" max="6"></label>
```

```
<label>Satisfaction (0 to 10) <input type="range" name="satisfaction" min="0" max="10" step="1"></label>
```

```
input type=" number"
```

A screenshot of a web browser window showing an input field. The label "Number of guests:" is followed by an input field with a small square containing a circular arrow icon to its right, indicating a numeric slider.

```
input type=" range"
```

A screenshot of a web browser window showing an input field. The label "Satisfaction (from 0 to 10):" is followed by a horizontal slider bar with a circular handle in the middle, indicating a range input.

Рис. 9.17. Типы управления вводом чисел `number` и диапазона `range` (показаны в Chrome на macOS)

Типы управления вводом `number` и `range` используют атрибуты `min` и `max` для указания минимальных и максимальных разрешенных для ввода значений (браузер при этом может проверить, соответствуют ли данные, вводимые пользователем, этим ограничениям). Значения атрибутов `min` и `max` не являются обязательными, и можно устанавливать одно значение без установки другого. Допускаются и отрицательные значения. Если элемент выбран, его значение можно увеличивать или уменьшать с помощью цифровых клавиш на клавиатуре компьютера, а также с помощью мыши или касаний пальцем.

Атрибут `step` позволяет разработчикам указывать для вводимых чисел допустимые шаги приращения. Значение, заданное по умолчанию: 1. Значение .5 определяет следующие значения приращения: 1, 1,5, 2, 2,5 и т. д., значение 100 определяет значения: 100, 200, 300 и т. д. Также можно присвоить атрибуту `step` значение `any`, что позволит приращению принять любое значение.

Элементы `number` и `range` разрешают только рассчитанные значения шага, а не какой-либо список допустимых значений (например, 1, 2, 3, 5, 8, 13, 21). Если необходимы настраиваемые значения, для их программирования нужно применять JavaScript.

Поскольку эти элементы относительно новы, далеко не все браузеры их поддерживают. Некоторые виджеты пользовательского интерфейса имеют направленные вверх и вниз стрелки, служащие для увеличения или уменьшения числовых значений, но многие браузеры лишены этой возможности. Мобильные браузеры (iOS Safari, Android, Chrome для Android) в настоящее время не поддерживают атрибуты `min`, `max` и `step`. Internet Explorer 9 и более ранние его версии не поддерживают ввод цифр и диапазонов. Впрочем, как уже неоднократно отмечалось, браузеры, не поддерживающие новые типы ввода, в качестве хорошего запасного варианта отображают в таких ситуациях стандартное поле ввода текста.

Выбор цвета

```
<input type="color">
```

Элемент выбора цвета

Назначение элемента управления выбором цвета заключается в создании всплывающей палитры цветов для визуального выбора значения цвета — аналогично опции, применяемой в операционных системах или программах редактирования изображений. Значения представляются в виде шестнадцатеричных значений RGB (#RRGGBB). На рис. ЦВ-9.18 показана палитра цветов в Chrome на macOS (совпадает с палитрой цветов macOS). Браузеры, не поддерживающие эту возможность (в настоящее время речь идет о всех версиях IE, iOS Safari и более старых версий Android), отображают заданный по умолчанию элемент управления вводом текста.

```
<label>Your favorite color: <input type="color" name="favorite">
</label>
```

На этом мы завершим обзор элементов управления формы. Рассмотрение методики вставки элементов управления формой служит составляющей частью процесса создания форм, но любому веб-разработчику стоит затратить усилия и время, чтобы убедиться, что форма максимально доступна. К счастью, имеется также ряд новых возможностей разметки, которые могут применяться для описания структуры формы.

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ ОБ ЭЛЕМЕНТАХ ФОРМЫ

Для полноты картины рассмотрим остальные элементы формы. Они появились в HTML5 и на момент написания книги поддерживаются не всеми браузерами. В любом случае, они несколько экзотичны, и вы не сразу решитесь добавить их в свой набор HTML-инструментов. Ранее уже был рассмотрен элемент `datalist`, применяемый для предоставления предлагаемых значений для ввода текста. В HTML5 также появились следующие элементы:

Элемент `progress`

```
<progress>...</progress>
```

Указание состояния текущего процесса

Элемент `progress` предоставляет пользователям информацию о состоянии текущего процесса — например, о загрузке файла. С его помощью можно показать на индикаторе выполнения процент завершенности процесса или состояние «ожидания» в виде, например, вращающихся песочных часиков. Элемент `progress` требует для своего выполнения следующего сценария :

```
Percent downloaded: <progress max="100" id="fave">0</progress>
```

Элемент `meter`

```
<meter>...</meter>
```

Представляет измерение в диапазоне

Элемент `meter` представляет измерение, выполненное в пределах известного диапазона значений (он также известен как датчик). Имеет ряд атрибутов: атрибуты `min`

- ▶ и `max` указывают самые маленькие и самые большие значения для диапазона (по умолчанию равны 0 и 100 соответственно), значения `low` и `high` могут применяться для отображения предупреждений о нахождении на нежелательных уровнях, а величина `optimum` указывает предпочтительное значение:

```
<meter min="0" max="100" name="volume">60%</meter>
```

Элемент `output`

```
<output>...</output>
```

Вычисление выходного значения

Проще говоря, элемент `output` определяет результат вычисления с помощью сценария или программы. Следующий пример из спецификации HTML5.2 использует элемент `output` и JavaScript для отображения суммы чисел, введенных в элементы ввода `a` и `b`:

```
<form onsubmit="return false" oninput="o.value =
a.valueAsNumber +
b.valueAsNumber">
<input name=a type=number step=any>
+ <input name=b type=number step=any> =
<output name=o for="a b"></output>
</form>
```

ФУНКЦИИ ДОСТУПНОСТИ ФОРМЫ

Важно учитывать, каким образом пользователи без визуальных браузеров смогут воспринять ваши веб-формы и применять их. Элементы формы `label`, `fieldset` и `legend` улучшают степень доступности, уточняя семантические связи между компонентами формы. Результирующая разметка становится не только семантически более насыщенной, но также включает больше элементов, выступающих «якорями» для правил таблиц стилей. В результате все пользователи остаются в выигрыше!

Метки (`Labels`)

```
<label>...</label>
```

Привязывает информацию к элементам управления формы

Несмотря на то что метка «Address» (Адрес) отображается рядом с текстовым полем для ввода адреса в визуальном браузере, в исходном коде метка и поле ввода могут быть разделенными. Элемент `label` связывает описательный текст с соответствующим полем формы. Этот подход обеспечивает наличие важного контекста для пользователей, имеющих речевые браузеры. Другое преимущество, связанное с применением меток, в том, что пользователи могут щелкать или нажимать на них в любом месте, что позволит выбрать элемент управления формы или сфокусироваться на нем. Пользователи с сенсорными устройствами также оценят удобство сенсорного интерфейса.

Каждый элемент `label` связан точно с одним элементом управления формой. Имеются два способа его применения. Один способ, называемый *неявной ассоциацией*, предусматривает вложение в элемент `label` элемента управления и его описания. В следующем примере элементы `label` присваиваются отдельным флагкам и текстовым описаниям, связанным с флагками (кстати, это способ маркировки переключателей и флагков — невозможно присвоить метку всей группе):

```
<ul>
  <li><label><input type="checkbox" name="genre" value="punk"> Punk
rock</label></li>
  <li><label><input type="checkbox" name="genre" value="indie">
Indie rock</label></li>
  <li><label><input type="checkbox" name="genre" value="hiphop">
Hip Hop</label></li>
  <li><label><input type="checkbox" name="genre" value="rockabilly">
Rockabilly</label></li>
</ul>
```

Другой способ, называемый *явной ассоциацией*, сопоставляет метку со ссылкой на `id` элемента управления. Атрибут `for` сообщает, для какого элемента управления предназначена эта метка. Такой подход полезен, если элемент управления не находится в исходном коде рядом с описательным текстом. Определенное преимущество также заключается в том, что метка и элемент управления сохраняются в виде двух отдельных элементов, что может быть удобно при выравнивании их с помощью таблиц стилей:

```
<label for="form-login-
username">Login account
</label>
<input type="text" name="login"
id="form-login-username">

<label for="form-login-password">
Password</label>
<input type="password"
name="password" id="form-
login-password">
```

ИСПОЛЬЗОВАНИЕ ПРЕФИКСОВ «`FORM-`» И «`FORM-LINK-`»

Чтобы различать связанные с формой идентификаторы `id` от других идентификаторов, имеющихся на странице, подумайте о добавлении к ним префикса «`form-`», как показано в приведенных примерах.

Другой способ организации форм состоит в присвоении элементу `form` идентификатора `id` и включении его в качестве префикса в соответствующие `id` элементов управления следующим образом:

```
<form id="form-login">
<input id="form-login-user">
<input id="form-login-passwd">
```

Элементы `fieldset` и `legend`

`<fieldset>...</fieldset>`

Группы, связанные с элементами управления и метками

`<legend>...</legend>`

Присваивание заголовка набору полей

Элемент `fieldset` указывает на логическую группу элементов управления. Он также может включать элемент `legend`, который предоставляет заголовок для вложенных

полей. На рис. 9.19 показано заданное по умолчанию отображение, продемонстрированное в следующем примере (впрочем, для изменения способа отображения элементов `fieldset` и `legend` можно использовать таблицы стилей):

```
<fieldset>
    <legend>Mailing List Sign-up</legend>
    <ul>
        <li><label>Add me to your mailing list <input type="radio" name="list" value="yes" checked></label></li>
        <li><label>No thanks <input type="radio" name="list" value="no">
            </label></li>
    </ul>
</fieldset>

<fieldset>
    <legend>Customer Information</legend>
    <ul>
        <li><label>Full name: <input type="text" name="fullname">
            </label>
        </li>
        <li><label>Email: <input type="text" name="email">
            </label></li>
        <li><label>State: <input type="text" name="state">
            </label></li>
    </ul>
</fieldset>
```

The screenshot shows a web page with two nested `fieldset` elements. The outer `fieldset` has a legend "Mailing List Sign-up". Inside it are two `li` items, each containing a `label` and a `input type="radio"`. The first radio button is checked. The inner `fieldset` has a legend "Customer Information". It contains three `li` items, each with a `label` and a `input type="text"`.

Рис. 9.19. Заданное по умолчанию отображение набора полей и их заголовков

БУДЬТЕ ВЕСЬМА ОСТОРОЖНЫ, ПРИМЕНЯ СТАЙЛИНГ ДЛЯ НАБОРОВ ПОЛЕЙ И ИХ ЗАГОЛОВКОВ

Правильное отображение наборов полей и их заголовков можетискажаться при изменении стилей. Например, цвета фона в наборах полей обрабатываются в разных браузерах различным образом. Заголовки, организованные с помощью элементов `legend`, отличаются тем, что текст в них нельзя переносить. Решение состоит в размещении в них элементов `span` или `b`, что поможет управлять представлением без ущерба доступности. При стилизации этих элементов формы обязательно предварительно выполните побольше тестов.

В процессе выполнения *упражнения 9.4* завершим создание формы заказа пиццы, сделав ее более доступной с помощью заголовков (надписей) и наборов полей.

УПРАЖНЕНИЕ 9.4. НАБОРЫ ПОЛЕЙ И ЗАГОЛОВКИ

Итак, форма заказа пиццы работает, но нам следует соответствующим образом обозначить ее заголовки и создать несколько наборов полей, что сделает ее более удобной для применения на вспомогательных устройствах. Еще раз откройте документ `pizza.html` и выполните следующие действия.

Мне нравится начинать с широких «мазков» с дальнейшей их детализацией. Можно было бы начать выполнение упражнения с организации элементов управления формы в наборы полей, а затем добавить заголовки. Можно выполнить эти действия в обратном порядке, но в идеале следует просто добавить надписи и наборы полей прямо сейчас, не откладывая это на потом.

1. Пункты (поля) раздела `Your Information` (Ваша информация), находящегося в верхней части формы, определенно концептуально связаны, поэтому представим его в виде элемента `fieldset`. А также измените разметку раздела заголовка с абзаца (`p`) на `legend` для представления набора полей:

```
<fieldset>
    <legend>Your Information</legend>
    <ul>
        <li>Name: <input type="text" name="fullname">
        </li>
        ...
    </ul>
</fieldset>
```

2. Затем сгруппируйте пункты `Crust`, `Toppings` и `Number` в большом наборе полей с помощью заголовка `Pizza specs` (Спецификации пиццы). В нашем случае текст имеется — просто нужно изменить применяемый элемент `p` на `legend`:

```
<h2>Design Your Dream Pizza:</h2>
<fieldset>
    <legend>Pizza specs</legend>
    Crust...
    Toppings...
    Number...
</fieldset>
```

3. Создайте другой набор полей, предназначенный только для вариантов выбора `Crust`, снова изменяя описание в абзаце на `legend`. Аналогично выполните то же для разделов `Toppings` (Начинка) и `Number` (Количество). Получим три набора полей, содержащихся в широком наборе полей `Pizza specs` (Спецификации пиццы):

```
<fieldset>
    <legend>Crust <em>(Choose one)</em>:</legend>
    <ul>...</ul>
</fieldset>
```

По завершении сохраните документ и откройте его в браузере. Теперь он должен быть близок к окончательной форме, показанной на рис. 9.2, с учетом ожидаемых отличий в браузерах.

4. Теперь добавим несколько меток. В наборе полей Your Information явным образом свяжите метку с элементом управления вводом текста, применяя метод метки `for/id`. Заключите описание в метки тегов и добавьте `id` к входным данным. Значения `for/id` должны быть описательными и при этом совпадать. Первое значение здесь реализовано, остались реализовать остальные четыре:

```
<li><label for="form-name">Name:</label> <input type="text" name="fullname" id="form-name"></li>
```

5. Для создания переключателей и флагков заключите элемент `input` и значение метки в элемент `label`. Таким образом, кнопка будет выделена при нажатии ее пользователем в любом месте внутри элемента `label`. Здесь выполнены действия для одного варианта, остальные варианты реализуйте самостоятельно:

```
<li><label><input type="radio" name="crust" value="white"> Classic White</label></li>
```

Сохраните документ, и все! Метки не оказывают влияния на заданный по умолчанию вид формы, но вы можете быть удовлетворены — в форму добавлена семантическая ценность, а вы, возможно, когда-нибудь воспользуетесь приобретенными здесь навыками, применив к форме стили.

ВИДЖЕТЫ ФОРМЫ

Несмотря на то что для HTML доступны десятки виджетов форм, разработчики склонны «закручивать свои» виджеты форм с использованием разметки, CSS и JavaScript. Этот подход предпочтителен, если необходимо предоставить пользовательские функции или сделать необычный стайлинг формы. Например, можно создать раскрывающееся меню, используя неупорядоченный список внутри элемента `div` вместо применения стандартного элемента `select`:

```
<div class="select" role="listbox">
  <ul class="optionlist">
    <li class="option" role="option">Red</li>
    <li class="option" role="option">Yellow</li>
  </ul>
</div>
```

Для того чтобы вспомогательные средства — такие, как программы чтения с экрана, могли распознавать его как элемент формы, примените ролевой атрибут ARIA для описания предполагаемой функции `div` (список) и каждого элемента `li` (вариант в списке). Существует также множество состояний и свойств ARIA, благодаря которым формы, как стандартные, так и пользовательские, можно применять со вспомогательными средствами. Полный их список приведен на сайте, доступном по адресу: www.w3.org/WAI/PF/aria-1.1/states_and_properties.

Пользовательские виджеты форм нуждаются в сценариях и таблицах стилей CSS, выходящих за рамки этой книги, но желательно, чтобы вы познакомились с основными приемами. Учтите также, что весьма легко все испортить, устанавливая неудобное (даже для зрячих пользователей) взаимодействие пользователя с формой, так что «экспериментируйте» с осторожностью.

Статья «How to Build Custom Form Widgets», доступная в MDN Web Docs (developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/How_to_build_custom_form_widgets), содержит интересный обзор. Также можно использовать готовый пользовательский виджет из одной из доступных библиотек JavaScript — таких, как jQuery UI (jqueryui.com).

МАКЕТ И ДИЗАЙН ФОРМЫ

Я не могу поставить здесь точку, не сказав несколько слов о дизайне форм, хотя эта глава посвящена разметке форм, а не их представлению.

Формы, пригодные к применению

Неудачно спроектированная форма может испортить восприятие пользователем сайта и негативно повлиять на бизнес-цели вашей работы. Такие формы приводят к потере клиентов, поэтому очень важно выполнить все правильно: как для отображения на настольном компьютере, так и на устройствах с небольшим экраном. В этом случае путь пользователя к покупке или другому действию, выполняемому с помощью форм, будет несложным.

Хороший веб-дизайн — весьма обширная тема, о нем можно написать отдельную книгу. Действительно, эти вопросам посвящена книга: «Web Form Design» (Rosenfeld Media), написанная экспертом по веб-формам Люком Вроблевски (Luke Wroblewski), и я настоятельно рекомендую вам эту книгу. Следующая книга Люка: «Mobile First» — содержит советы по форматированию формы в мобильном контексте. Более ста статей о формах можно просмотреть на его сайте, который доступен по адресу: www.lukew.com/ff?tag=forms.

Далее приводится небольшая подборка советов из книги «Web Form Design», которые наверняка вас заинтересует, и вы поймете, что вся книга заслуживает прочтения.

- *Избегайте необязательных вопросов.* Помогите пользователям пройти через вашу форму как можно быстрее и не включайте в нее вопросы, которые не служат выполнению поставленной задачи. Дополнительные вопросы, кроме понятного замедления, настораживают пользователя, и он начинает обдумывать мотивацию задаваемых вопросов. Если имеется иной способ получения нужной вам информации (например, уточнение типа кредитной карты, который может определяться по первым четырем цифрам ее номера), применяйте альтернативные средства и не утомляйте пользователя. Если некая информация полезна, но не обязательна, вернитесь к вопросу позднее, после отправки формы и установки дружеских отношений с пользователем.
- *Учитывайте влияние расположения надписей.* Расположение надписи относительно поля ввода оказывает влияние на время, затраченное на заполнение формы. Чем меньше внимание пользователя будет рассеиваться по странице, тем быстрее форма будет заполнена. При размещении надписей над соответствующими полями создается единое выравнивание, что ускоряет просмотр формы и способствует заполнению формы, особенно если запрашивается знакомая информация (имя, адрес и т. п.). Расположенные сверху надписи также могут иметь изменяемую длину, что облегчает работу на узких устройствах с маленьким экраном. Тем не менее подобное расположение надписей удлиняет форму, поэтому, если вертикального пространства недостаточно, надписи можно располагать слева от вводимых данных. Выравнивание надписей по левому краю приводит

к замедлению при заполнении формы, но может быть и целесообразно, если вы желаете, чтобы пользователь замедлил работу, получил возможность выполнять просмотр и учитывать типы необходимой информации.

- *Тщательно выбирайте типы вводимых данных.* Как уже отмечалось в этой главе ранее, имеется весьма много типов вводимых данных, и иногда нелегко решить, какой из них применить. Например, список вариантов можно представить в виде раскрывающегося меню или нескольких вариантов для выбора с помощью флажков. Тщательно взвесьте все «за» и «против» для каждого типа элементов управления и проведите пользовательское тестирование.
- *Группируйте связанные элементы управления вводом данных.* Выполнение анализа множества полей, меню и кнопок в форме реализовать проще, если они визуально сгруппированы по связанной теме. Например, контактная информация пользователя может быть представлена в компактной группе — тогда пять или шесть полей ввода данных воспринимаются как единое целое. В этом случае порой требуется лишь небольшая подсказка — например, точная горизонтальная линейка и некоторое дополнительное пространство. Не перегружайте форму лишними деталями.
- *Уточните первичные и вторичные действия.* Основным действием, выполняемым в конце работы с формой, обычно является щелчок на кнопке отправки данных: **Buy** (Купить, **Register** (Зарегистрировать) и т. д., который свидетельствует о завершении работы с формой и готовности двигаться дальше. Обычно эта кнопка визуально доминирует и ее легко найти (желательно выполнить выравнивание ее по основной оси формы). Используя JavaScript, можно выделить кнопку **Submit** (Отправить) как неработающую, пока все необходимые данные не будут введены.

Вторичные действия, как правило, реализуют «шаг назад» — такие, как очистка или сброс формы. Если следует включить дополнительное действие, убедитесь, что оно оформлено стилями иначе и не столь важно по сравнению с основным. Также желательно рассмотреть вариант отмены действия.

Оформление форм стилями

Как показано в этой главе ранее, визуализация разметки форм по умолчанию не приводит к высокому качеству ее отображения, которое отличает сегодня большинство профессиональных веб-форм. Поэтому вы можете использовать таблицы стилей для изменения внешнего вида большинства элементов управления формы и создания ее презентабельного макета. Даже простое выравнивание и согласованность внешнего вида формы с остальной частью вашего сайта значительно улучшает впечатление, производимое на пользователя.

Имейте в виду, что виджеты форм формируются браузером и основываются на соглашениях операционной системы. Но вы можете изменять размеры, поля, шрифты, цвета, границы и фоновые эффекты для таких элементов формы, как ввод текста,

выбор меню, текстовые области, наборы полей, надписи и условные обозначения. Обязательно протестируйте форму в различных браузерах, чтобы проверить ее отображение и избежать появления неприятных сюрпризов. В главе 19 приведены некоторые конкретные методы, которые пригодятся вам при наличии опыта работы с таблицами стилей CSS. Для получения дополнительной помощи выполните поиск в Интернете по ключевой фразе: CSS для форм. В результате выполненного поиска вы сможете ознакомиться с рядом полезных учебных пособий.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Вы уже готовы ответить на вопросы, связанные с веб-формами? Далее приведены несколько вопросов, которые покажут, овладели ли вы их основами. Ответы на эти вопросы содержатся в *приложении 1*.

1. Определите, следует ли направлять каждую из приведенных форм с помощью метода GET или POST:
 - a. Форма для доступа к вашему банковскому счету в Интернете: _____
 - b. Форма для отправки на принтер рисунка футболки: _____
 - c. Форма для поиска статей в архиве: _____
 - d. Форма для сбора длинных записей эссе: _____
2. Какой элемент управления формой более всего подходит для следующих задач? Если ваш ответ «`input`», обязательно укажите также тип. Некоторые задачи могут иметь более одного правильного ответа.
 - a. Выберите из 12 знаков свой астрологический знак.
 - b. Укажите, есть ли у вас в анамнезе болезни сердца (да или нет).
 - c. Напишите обзор книги.
 - d. Выберите любимые вкусы мороженого из списка восьми пунктов.
 - e. Выберите любимые вкусы мороженого из списка, содержащего 25 пунктов.
3. Каждый из этих примеров разметки содержит ошибку. Можете ли вы заметить?
 - a. `<input name=>country value=>Your country here.>`
 - b. `<checkbox name="color" value="teal">`
 - c. `<select name="popsicle">`
`<option value="orange">`
`<option value="grape">`
`<option value="cherry">`
`</select>`
 - d. `<input type="password">`
 - e. `<textarea name="essay" width="100" height="6">Your story.`
`</textarea>`

ОБЗОР ЭЛЕМЕНТОВ: ФОРМЫ

В табл. 9.1 приведены связанные с формой элементы и атрибуты, включенные в HTML 5.2 (некоторые атрибуты в этой главе не рассмотрены). Атрибуты для каждого типа элемента управления вводом данных приведены в табл. 9.2.

Таблица 9.1. Связанные с формой элементы и атрибуты, включенные в HTML 5.2

Элемент и атрибуты	Описание
button	Универсальная кнопка ввода
autofocus	Автоматически фокусирует элемент управления формы при загрузке страницы
name= "текст"	Предоставляет для элемента управления уникальное имя переменной
disabled	Отключает input, поэтому его нельзя выбирать
type="submit reset button"	Тип настраиваемой кнопки
value= "текст"	Определяет значение, которое будет направлено на сервер
menu= "идентификатор значения"	Определяет назначенное раскрывающееся меню
form, formaction, formenctype, formmethod, formnovalidate, formtarget	Атрибуты, связанные с отправкой формы, которые применяются для кнопок Submit и Reset
datalist	Предоставляет список параметров для ввода текста
fieldset	Группы связанных элементов управления и меток
disabled	Отключает все элементы управления вводом данных в наборе полей, чтобы их нельзя было выделить, отредактировать или отправить
form= "идентификатор значения"	Связывает элемент с определенной формой
name= "текст"	Предоставляет уникальное имя переменной для элемента управления
form	Элемент формы
action= "url-ссылка"	Расположение программы обработки форм (обязательно)
method= "get post "	Метод, используемый для отправки данных формы
enctype= "тип контента"	Метод кодирования: как правило либо application/x-www-form-urlencoded (заданный по умолчанию) или multipart/form-data
accept-charset= "набор символов"	Кодировка применяемых символов

Табл. 9.1 (продолжение)

Элемент и атрибуты	Описание
autocomplete	Настройка по умолчанию для функции автозаполнения для элементов управления в форме
name="текст"	Название формы для использования в document.forms API
novalidate	Обход валидации контроля формы для этой формы
target="text _blank _self _parent _top"	Устанавливает контекст просмотра
input	Создает различные элементы управления вводом данных на основе значения type
autofocus	Указывает, что элемент управления должен быть готов для ввода при загрузке документа
type="submit reset button text password checkbox radio image file hidden email tel search url date time datetime-local month week number range color"	Тип ввода
См. таблицу 9.2 для получения полного списка атрибутов, связанных с каждым типом вводимых данных	
Disabled	Отключает вводимые данные, поэтому их нельзя выбрать, отредактировать или отправить
form="идентификатор значения формы"	Связывает элемент управления с указанной формой
label	Связывает надписи с элементами управления
for="текст"	Идентифицирует связанный элемент управления по ссылке на его id
legend	Назначает заголовок для fieldset
meter	Представляет дробное значение в известном диапазоне
high="число"	Указывает диапазон, который считается «большим» («high») для счетчика
low="число"	Указывает диапазон, который считается «маленьким» («low») для счетчика
max="число"	Определяет самое большое значение для диапазона
min="число"	Определяет самое меньшее значение для диапазона
optimum="число"	Указывает число, считающееся «оптимальным» («optimum»)
value="число"	Определяет фактическое или измеренное значение

Табл. 9.1 (продолжение)

Элемент и атрибуты	Описание
optgroup	Определяет группу параметров
disabled	Отключает группу optgroup, поэтому нельзя ее выделить
label= "текст"	Определяет надпись для группы вариантов
option	Вариант в элементе управления выбором меню
disabled	Отключает вариант, поэтому его нельзя выбрать
label= "текст"	Поддерживает альтернативную надпись для варианта
selected	Предварительно выделяет вариант
value= "текст"	Предоставляет альтернативное значение для варианта
output	Представляет результаты расчета
for= "текст"	Формирует связь между выводом и другим элементом
form= "Значение идентификатора формы"	Связывает элемент управления с указанной формой
name= "текст"	Предоставляет уникальное имя переменной для элемента управления
progress	Представляет ход выполнения задачи (может использоваться, даже если максимальное значение задачи неизвестно)
max= "число"	Определяет общее значение или окончательный размер задачи
value= "число"	Определяет, какая часть задачи выполнена
select	Раскрывающееся меню или список прокрутки
autofocus	Указывает, что элемент управления должен быть выделен и уже подготовлен для ввода при загрузке документа
Disabled	Указывает, что элемент управления не активен, можно выполнить активизацию с помощью сценария
form= "значение идентификатора формы"	Связывает элемент управления с указанной формой
multiple	Позволяет множественные выделения в списке прокрутки

Табл. 9.1 (окончание)

Элемент и атрибуты	Описание
<code>name="текст"</code>	Предоставляет для элемента управления уникальное имя переменной
<code>required</code>	Указывает, что пользовательский ввод необходим для данного элемента управления
<code>size="число"</code>	Высота (в текстовых строках) списка прокрутки
<code>textarea</code>	Многострочное поле для ввода текста
<code>autocomplete</code>	Подсказка для функции автозаполнения формы
<code>autofocus</code>	Указывает, что элемент управления должен быть выделен и подготовлен для ввода при загрузке документа
<code>cols="число"</code>	Ширина (в символах) текстовой области
<code>dirname="текст"</code>	Позволяет направлять отправленный текст
<code>Disabled</code>	Отключает элемент управления, поэтому его нельзя выделить
<code>form="значение идентификатора формы"</code>	Связывает элемент управления с указанной формой
<code>Inputmode</code>	Подсказка для выбора модальности ввода
<code>maxlength="текст"</code>	Определяет максимальное число символов, которое может ввести пользователь
<code>minlength="текст"</code>	Определяет минимальное число символов, которое может ввести пользователь
<code>name="текст"</code>	Предоставляет уникальное имя переменной для элемента управления
<code>placeholder="текст"</code>	Предоставляет краткую подсказку, помогая пользователю ввести правильные данные
<code>readonly</code>	Не допускает возможность изменения пользователем элемента управления
<code>required</code>	Указывает, что пользовательский ввод необходим для данного элемента управления
<code>rows="число"</code>	Высота текстовой области в тексте
<code>wrap="hard soft"</code>	Определяет, будут ли разрывы строк при вводе текста возвращаться в данные; значение <code>hard</code> сохраняет разрывы строк, а <code>soft</code> — нет

Таблица 9.2. Доступные атрибуты для каждого типа элемента управления вводом данных (начало)

Элементы	Атрибуты									
	submit	reset	button	text	password	Checkbox	radio	image	file	hidden
accept										
alt										
autocomplete										
autofocus	•	•	•	•	•	•	•	•	•	•
checked										
disabled	•	•	•	•	•	•	•	•	•	•
form	•	•	•	•	•	•	•	•	•	•
formaction	•								•	
formenctype	•								•	
formmethod	•								•	
formnovalidate	•								•	
formtarget	•								•	
height									•	
list				•						
max										
min										
maxlength				•	•				•	
minlength				•	•				•	
multiple									•	
name	•	•	•	•	•	•	•	•	•	•
pattern				•						
placeholder				•						
readonly				•						
required				•	•	•	•		•	
size				•	•				•	
src									•	
step										
value	•	•	•	•	•	•	•		•	•
width								•		

Табл. 9.2 (окончание)

Элементы	Атрибуты					
	email	telephone, search, url	number	range	date, time, datetime-local, month, week	color
accept						
alt						
autocomplete	•	•	•	•	•	•
autofocus	•	•	•	•	•	•
checked						
disabled	•	•	•	•	•	•
form	•	•	•	•	•	•
formaction						
formenctype						
formmethod						
formnovalidate						
formtarget						
height						
list	•	•	•	•	•	•
max			•	•	•	
min			•	•	•	
maxlength	•	•				
minlength	•	•				
multiple						
name	•	•	•	•	•	•
pattern	•	•				
placeholder	•	•				
readonly	•	•	•			
required	•	•	•		•	
size	•	•				
src						
step			•	•	•	
value	•	•	•	•	•	•
width						

ГЛАВА 10

ВСТРОЕННЫЕ МУЛЬТИМЕДИЙНЫЕ ОБЪЕКТЫ

В этой главе...

- ▶ Элемент `iframe`
- ▶ Элемент `object`
- ▶ Аудио и видеоплееры
- ▶ Элемент `canvas`

В HTML-спецификации *встроенный контент* определяется следующим образом:
контент, импортирующий в документ иной ресурс, или контент из другого словаря, который вставляется в документ.

В главе 7 были приведены примеры, относящиеся к обеим частям этого определения, поскольку изображения также представляют собой встроенный контент. Элементы `img` и `picture` указывают на внешний ресурс изображения с применением атрибутов `src` или `srcset`, а элемент `svg` встраивает файл изображения, записанный в формате SVG, непосредственно в страницу.

Естественно, что изображения являются не единственными объектами, которые можно встраивать в веб-страницу. В этой главе мы рассмотрим и другие типы встроенного контента, а также соответствующую разметку, а именно:

- окно для просмотра внешнего HTML-источника: `iframe`;
- многоцелевые встраиваемые элементы: `object` и `embed`;
- видео- и аудиоплееры: элементы `video` и `audio`;
- сценарий для рисования, который можно применять для анимации или создания игровой интерактивности: элемент `canvas`.

ОКНО В ОКНЕ: ЭЛЕМЕНТ `IFRAME`

`<iframe>...</iframe>`

Встроенное окно просмотра

Элемент `iframe` (сокращение от слов inline frame, встроенный кадр) позволяет встроить в документ отдельный HTML-документ или другой веб-ресурс. Этот

элемент известен уже много лет, но в последнее время стал одним из наиболее популярных способов обмена контентом между сайтами.

Например, при запросе вложения видео с YouTube или карты из Google Maps для копирования и вставки этого контента на страницу предоставляется код на основе `iframe`. Многие другие медиасайты следуют этому примеру, потому что это позволяет им управлять различными аспектами контента, который вы размещаете на своей странице. Встроенные фреймы (кадры, рамки) стали стандартным инструментом и для размещения рекламного контента, который отображается с помощью Flash-плеера. Учебные веб-сайты используют встроенные фреймы для размещения на своих страницы примеров кода.

В результате добавления на страницу элемента `iframe` на ней создается небольшое окно (или *вложенный контекст просмотра*, как это названо в спецификации), в котором и отображается внешний ресурс. Встроенный фрейм размещается на странице примерно так же, как и изображение, — с указанием источника (`src`) его исходного содержимого. Атрибуты `width` и `height` определяют размеры рамки этого окна..

Контент элемента `iframe` для браузеров, которые не поддерживают этот элемент, чтобы быть отображенными, должен соответствующим образом резервироваться с помощью элементов, которые этим браузером поддерживаются, хотя поддержку элемента `iframe` сейчас осуществляют практически все браузеры.

В следующем весьма простом примере исходный документ отображает во встроенном фрейме веб-страницу `glossary.html` (рис. 10.1). Показанный здесь фрейм имеет собственную полосу прокрутки, поскольку встроенный HTML-документ слишком длинный. По правде говоря, нечасто можно видеть такой вариант применения элементов `iframe` (за исключением, возможно, примеров кода), но зато так легче понять, как они работают.

```
<h1>An Inline Frame</h1>

<iframe src="glossary.html" width="400" height="250" >
    Read the <a href="glossary.html">glossary</a>.
</iframe>
```

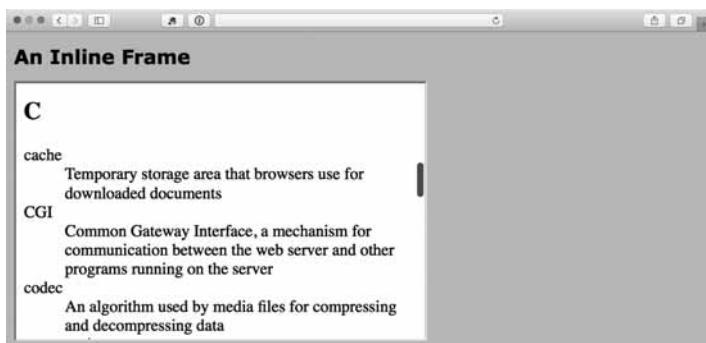


Рис. 10.1. Встроенный фрейм, добавленный с помощью элемента `iframe`, похож на вставленное в браузер окно, в котором отображается внешний HTML-документ или ресурс

В современном веб-дизайне элементы `iframe` далеко не всегда представляют собой окна. При этом у них часто вообще нет никаких признаков наличия какой бы то ни было рамки, как показано в примере с Google Maps (рис. 10.2).



Рис. 10.2. Рамка окна встроенного элемента `iframe` зачастую вовсе не отображается, как показано в этом примере с окном Google Maps

При использовании элементов `iframe` возникают некоторые проблемы с безопасностью, связанные с появлением дополнительных открытых окон, которые могут служить мишенями для хакеров. Атрибут `sandbox` накладывает ограничения на возможности контента, расположенного в таком окне, — например, запрещаются формы, всплывающие окна, сценарии и т. п.

ДОПОЛНИТЕЛЬНЫЙ МАТЕРИАЛ ПО ВОПРОСАМ БЕЗОПАСНОСТИ

Вопросы, связанные с безопасностью применения элементов `iframe`, выходят за рамки этой главы, но вам с ними было бы неплохо ознакомиться, если вы собираетесь активно использовать эти элементы. В качестве дополнительного материала рекомендую статью «From object to iframe: Other Embedding Technologies» (developer.mozilla.org/en-US/docs/Web/HTML/Multimedia_and_embedding/Other_embedding_technologies), где содержится подробный обзор вопросов, связанных с безопасностью при использовании элементов `iframe`.

Для лучшего освоения методики работы с элементом `iframe` задействуйте его для добавления на страницу любимого видео (*упражнение 10.1*).

УПРАЖНЕНИЕ 10.1. ВСТРАИВАНИЕ ВИДЕО НА СТРАНИЦУ С ПОМОЩЬЮ ЭЛЕМЕНТА `IFRAME`

Если вы хотите поэкспериментировать с `iframe`, то попробуйте поместить на страницу видео с YouTube. Начните с создания нового HTML-документа, включая основные его структурные элементы, которые были рассмотрены в главе 4. Перейдите на YouTube и, как только окажетесь на странице выбранного видео,

нажмите кнопку **Поделиться** (Share), а затем выберите вариант **Вложить** (Embed). Код `iframe` служит для копирования и вставки видео. Если щелкнуть на кнопке **Показать больше** (Show more), отобразятся дополнительные параметры конфигурации. Скопируйте код `iframe` и вставьте его в новый HTML-документ. Откройте его в браузере, и все готово!

МНОГОЦЕЛЕВОЕ СРЕДСТВО ДЛЯ ВСТРАИВАНИЯ: **OBJECT**

`<object>...</object>`

Представляет внешний ресурс

`<param>`

Параметры объекта

Созданные на заре Интернета веб-браузеры были ограничены в выборе объектов отображения, поэтому для представления мультимедийных объектов (а первые веб-браузеры были не в состоянии выполнять эту операцию) применялись *плагины*. Java-апплеты, Flash-фильмы, RealMedia (устаревший формат видео и аудио для Интернета) и другим мультимедийным объектам для воспроизведения в браузере

ПЛАГИН

Плагин — программное средство, которое предоставляет браузеру отсутствующие у него функциональные возможности.

Для встраивания на веб-страницу медиаресурсов мы сейчас используем элементы `object` и `embed`. Технологии их применения немного различаются. Элемент `object` представляет собой многоцелевое средство, предназначенное для встраивания объектов. Он применяется для размещения изображения, создания вложенного контекста просмотра (например, контента `iframe`) или встраивания ресурса, который затем обрабатывается плагином.

Элемент `embed` предназначен только для плагинов. Он используется до сих пор, но его популярность падает, а соответствующая процедура встраивания практически не применяется (см. далее врезку «Элемент `embed`»). Мультимедийные средства, такие, как Java-апплеты и Flash-фильмы, выходят из употребления, и современные браузеры для естественного отображения большого числа типов мультимедийных объектов применяют API, а не плагины. Кроме того, мобильные браузеры, а также браузер Microsoft Edge плагины вовсе не поддерживают.

А теперь рассмотрим элемент `object`. В самом минимальном варианте использования элемент `object` для указания на ресурс применяет атрибут `data`, а также атрибут `type` — для предоставления его MIME-типа. Любой контент внутри тегов элемента `object` для браузеров, которые не поддерживают этот тип встроенного ресурса, должен соответствующим образом резервироваться. В следующем примере

также требовались сторонние плагины. Представьте себе, даже для представления JPEG-изображения необходимо было применять плагин!

ЭЛЕМЕНТ `EMBED`

Элемент `embed` создан Netscape для применения с технологиями плагинов. Этот элемент всегда хорошо поддерживался, но до появления HTML5-спецификации для него не существовала формальная спецификация. При наличии большого числа других вариантов встраивания мультимедийных объектов элемент `embed` уже не столь полезен, как ранее. Этот элемент часто применяется как запасной вариант, если имеется веская причина для поддержки устаревших версий браузера.

Элемент `embed` является пустым, а для указания на внешний ресурс служит атрибут `src`:

```
<embed type="video/quicktime"  
src="movies/hekboy.mov"  
width="320" height="256">
```

Имеются дополнительные, специфичные для мультимедийного объекта атрибуты, которые устанавливают параметры, аналогичные элементу `param`, но здесь они подробно не описываются, поскольку я считаю эту тему исчерпанной.

приведен простой элемент `object`, который помещает на страницу SVG-изображение и поддерживает резервный PNG-вариант:

```
<object data="picture.svg" type="image/svg+xml">  
      
</object>
```

Для элемента `object` доступны и дополнительные атрибуты, которые различаются в зависимости от типа размещаемого мультимедийного объекта. Формат мультимедийного объекта может также требовать, чтобы элемент `object` содержал ряд элементов `param`, устанавливающих специфичные для мультимедийного объекта этого типа параметры.

В следующем примере элементы `param` определяют, запускается ли просмотр видео автоматически (no) или имеет видимые элементы управления (yes):

```
<object type="video/quicktime" data="movies/hekboy.mov" width="320"  
height="256">  
    <param name="autostart" value="false">  
    <param name="controller" value="true">  
</object>
```

ПРОЩАЙ, FLASH

Заявление Apple о том, что устройства iOS будут лишены поддержки Flash, привело к усилению внедрения HTML5, и в конечном итоге Adobe прекратила разработку своих мобильных Flash-продуктов. Затем Microsoft объявила о прекращении выпуска своего медиаплеера Silverlight, поскольку появились альтернативы со стороны HTML5. В настоящее время HTML5 еще далек от возможности восполнить обширные функциональные возможности, присущие Flash, но постепенно все меняется. Иногда еще можно заметить Flash-плееры, установленные на настольных компьютерах, но прогресс на пути от плагинов к стандартным веб-технологиям уже очевиден.

ВИДЕО И АУДИО

До недавнего времени браузеры не имели встроенных возможностей по обработке видео или звука, поэтому эти функции выполняли плагины. Но с развитием Интернета как платформы открытых стандартов и широкополосных подключений, обеспечивающих более высокие скорости загрузки, чем раньше, пришло время сделать поддержку мультимедиа частью встроенных возможностей браузеров. И это случилось, поскольку появились новые элементы `audio` и `video` и соответствующие API (см. далее врезку «API — интерфейс программирования приложений»).

API — ИНТЕРФЕЙС ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ

API (Application Programming Interface, интерфейс программирования приложений) — это стандартизованный набор команд, имен данных, свойств, действий и т. д., позволяющий приложениям взаимодействовать друг с другом. В HTML5 появился ряд API-интерфейсов, предоставляющих браузерам программируемые функции, которые ранее могли реализовываться только с помощью сторонних плагинов.

Некоторые API имеют компоненты разметки — например, при встраивании мультимедиа с помощью новых элементов HTML5 `audio` и `video` (Media Player API). Другие выполняются полностью «за кулисами» с помощью JavaScript или серверных компонентов, таких, как веб-приложения, которые иногда функционируют даже без подключения к Интернету (Offline Web Application API, API автономных веб-приложений).

Консорциум W3C работает над созданием множества API для использования с веб-приложениями, но все они находятся на разных стадиях завершения и реализации. Большинство из них имеют собственные спецификации, отличные от спецификации HTML5, но обычно они подводятся под защиту «зонтика» HTML5, включающего веб-приложения.

Список всех HTML5 API и находящихся в разработке спецификаций доступен на сайте html5-overview.net, который поддерживается Эриком Уайльдом (Erik Wilde). Введение в хорошо известные API содержится в приложении 4.

Хорошие и плохие новости

Хорошой новостью является тот факт, что элементы `audio` и `video` успешно поддерживаются в современных браузерах, включая IE 9+, Safari, Chrome, Opera и Firefox для настольных компьютеров, а также iOS Safari 4+, Android 2.3+ и Opera Mobile (однако сюда не входит Opera Mini).

Но, даже если теоретически еще можно представить себе идеальный мир, где все браузеры в совершенной гармонии поддерживают видео- и аудиовозможности, лично я сильно сомневаюсь, что это легко реализовать на практике. Хотя браузеры поддерживают разметку и JavaScript для встраивания медиаплееров, к сожалению, среди них нет согласия по поводу поддерживаемых форматов. И сейчас мы совер-

шим краткое путешествие в страну форматов мультимедийных файлов, поскольку, если вы захотите добавлять на страницы видео или звук, сначала нужно разобраться с теорией.

Как работают медиаформаты?

При подготовке аудио- или видеоконтента для доставки его через Интернет необходимо принять два решения, касающиеся форматов. Во-первых, каким образом *кодируются* носители (то есть каковы алгоритмы, применяемые для преобразования исходного кода в последовательность нулей и единиц, а также как выполняется сжатие). Используемый для кодирования метод называется *кодеком* (от слов «кодировать/декодировать»). В настоящее время существует множество кодеков. Названия некоторых из них звучат знакомо — как MP3, другие могут звучать по-новому — например: H.264, Vorbis, Theora, VP8 и AAC.

Во-вторых, нужно выбрать *формат контейнера* для мультимедийного объекта. Можно представить такой контейнер в виде ZIP-файла, который содержит в пакете уже сжатый медиаобъект и его метаданные. Обычно формат контейнера совместим с несколькими типами кодеков, и мы не будем вдаваться во все детали. Просто перейдем к делу и рассмотрим наиболее распространенные комбинации контейнеров и кодеков, доступные в Интернете. Если вы собираетесь добавлять на сайт видео или аудио, настоятельно рекоменду ознакомиться со всеми этими форматами.

ДОПОЛНИТЕЛЬНЫЙ ИСТОЧНИК ИНФОРМАЦИИ

Для подробного ознакомления с методикой встраивания аудио и видео на HTML-страницы рекомендую книгу «Beginning HTML5 Media: Make the Most of the New Video and Audio Standards for the Web» Сильвии Пфайфер (Silvia Pfeiffer) и Тома Грина (Tom Green), издательство Apress.

Знакомство с видеоформатами

Для представления видео наиболее распространены следующие форматы:

- **контейнер MPEG-4 + видеокодек H.264 + аудиокодек AAC** — эту комбинацию обычно называют MPEG-4, ее файлы имеют расширение mp4 или m4v. Кодек H.264 — это высококачественный и гибкий видеокодек, он запатентован, и для его использования требуется лицензия, приобретаемая за определенную плату. Все современные браузеры, поддерживающие воспроизведение видео средствами HTML5, имеют возможность воспроизводить файлы MPEG-4 с кодеком H.264. Более новый кодек H.265, также известный под аббревиатурой HEVC (High Efficiency Video Coding, высокоеффективное кодирование видео) находится в разработке и снижает битрейт (скорость передачи данных) вдвое, но пока еще недостаточно хорошо поддерживается;
- **контейнер WebM + видеокодек VP8 + аудиокодек Vorbis.** WebM — это контейнерный формат, преимущество которого заключается в том, что он разработан на основе открытого исходного кода и не требует авторских отчислений. Файлы этого формата имеют расширение webm. Формат был разработан специально для работы с кодеками VP8 и Vorbis;

- контейнер WebM + видеокодек VP9 + аудиокодек Opus** — видеокодек VP9 из проекта WebM поддерживает то же качество видео, что и VP8 и H.264, при половинном битрейте. Поскольку этот подход достаточно новый, поддержка его не столь хороша, но это отличный вариант для браузеров, которые его поддерживают;
- контейнер Ogg + видеокодек Theora + аудиокодек Vorbis** — обычно этот формат называют Ogg Theora, его файл имеет расширение ogv. Все его кодеки и контейнер разработаны на основе открытого исходного кода и не обременены патентами или лицензионными ограничениями, но некоторые разработчики полагают, что качество полученных изображений от этого варианта хуже. Кроме новых браузеров он поддерживается некоторыми устаревшими версиями браузеров Chrome, Firefox и Android, которые не поддерживают WebM или MP4, поэтому этот вариант доступен большему количеству пользователей.

Известная проблема заключается в том, что производители браузеров не договорились о едином формате поддержки. Некоторые браузеры разработаны на основе открытого исходного кода и поддерживают бесплатные варианты, такие, как Ogg Theora или WebM. Другие придерживаются формата H.264, несмотря на требования, связанные с необходимостью лицензионных отчислений. Ясно, что веб-разработчики должны создавать по нескольку версий видео, обеспечивая поддержку для всех браузеров. В табл. 10.1 приведены браузеры, поддерживающие различные параметры видео (см. врезку «Настройка сервера»).

НАСТРОЙКА СЕРВЕРА

В табл. 10.1 и 10.2 в столбце «Тип» для каждого формата мультимедиа определен тип MIME. Если сайт выполняется на сервере Apache, то, чтобы убедиться в том, что видео- и аудиофайлы обслуживаются правильно, может потребоваться добавить в файл сервера .htaccess описание соответствующих типов. В следующем примере добавляются тип/подтип MP4 и расширения:

```
AddType video/mp4 mp4 m4v
```

Таблица 10.1. Поддержка видео в настольных и мобильных браузерах
(по состоянию на 2017-й год)

Формат	Тип	IE	MS Edge	Chrome	Firefox	Safari	Opera	Android	iOS Safari
MP4 (H.264)	video/mp4 mp4 m4v	9.0+	12+	4+	Да*	3.2+	25+	4.4+	3.2+
WebM (VP8)	video/webm webm webmvp	–	–	6+	4.0+	–	15+	2.3+	–
WebM (VP9)	video/webm webm webmvp	–	14+	29+	28+	–	16+	4.4+	–
Ogg Theora	video/ogg ogv	–	–	3.0+	3.5+	–	13+	2.3+	–

* Версии Firefox варьируются в зависимости от операционных систем.

ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ: HLS (ПОТОКОВОЕ HTTP-ВИДЕО)

Если на достаточно серьезном уровне обрабатывать видео в Интернете, следует ознакомиться с форматом *HLS* (HTTP Streaming Video, потоковое HTTP-видео) — потоковым форматом, который может оперативно изменять битрейт. Отличной исходной точкой для начала знакомства с этим форматом является статья в Википедии: <https://ru.wikipedia.org/wiki/HLS>.

Знакомство с аудиоформатами

Следующая ситуация выглядит до боли знакомой: имеется выбор из нескольких форматов, но отсутствует поддерживаемый всеми браузерами формат (см. табл. 10.2):

- **MP3** — формат MP3 (сокращение от MPEG-1 Audio Layer 3) совмещает в себе кодек и контейнер, его файлы имеют расширение mp3. Этот формат повсеместно применяется для распространения музыкальных файлов;
- **WAV** — формат WAV (расширение имени файла wav) также является и кодеком, и контейнером. Этот формат не сжимается, поэтому подходит только для коротких клипов типа звуковых эффектов;
- **контейнер Ogg + аудиокодек Vorbis** — обычно называется Ogg Vorbis, его файлы имеют расширение ogg или oga;
- **контейнер MPEG 4 + аудиокодек AAC** — формат MPEG4 audio (расширение m4a) менее распространен по сравнению с MP3;
- **контейнер WebM + аудиокодек Vorbis** — формат WebM (webm) также может включать только аудио;
- **контейнер WebM + аудиокодек Opus.** Opus — новый, более эффективный аудиокодек, который можно применять с WebM.

Таблица 10.2. Поддержка аудио текущими браузерами (по состоянию на 2017-й год)

Формат	Тип	IE	MS Edge	Chrome	Firefox	Opera	Safari	iOS Safari	Android
MP3	audio/mpeg mp3	9.0+	12+	3.0+	22+	15+	4+	4.1	2.3+
WAV	audio/wav or audio/wave	—	12+	8.0+	3.5+	11.5+	4+	3.2+	2.3+
Ogg Vorbis	audio/ogg ogg oga	—	—	4.0+	3.5+	11.5+	—	—	2.3+
MPEG-4/AAC	audio/mp4 m4a	11.0+	12+	12.0+	—	15+	4+	4.1+	3.0+
WebM/Vorbis	audio/webm webm	—	—	6.0+	4.0+	11.5+	—	—	2.3-3+
WebM/Opus	audio/webm webm	—	14+	33+	15+	20+	—	—	—

ИНСТРУМЕНТЫ ДЛЯ КОДИРОВАНИЯ ВИДЕО И АУДИО

Имеется много инструментов для обработки и кодирования видео- и аудиофайлов, описание которых потребует отдельной книги. Поэтому мы остановимся на инструментах, которые являются бесплатными и вполне работоспособными.

Конвертация видео

- Handbrake (handbrake.fr) — популярный инструмент с открытым исходным кодом, применяемый для конвертации в MPEG4 с помощью H.264, H.265, VP8 и Theora. Доступен для Windows, macOS и Linux;
- Firefogg (firefogg.org) — расширение для Firefox, служащее для преобразования видео в форматы WebM (VP8 и VP9) и Ogg Theora. Установите кросс-платформенное расширение Firefogg для Firefox, посетите сайт Firefogg и с помощью онлайн-интерфейса конвертируйте видео;
- FFmpeg (ffmpeg.org) — инструмент командной строки с открытым исходным кодом, применяемый для конвертирования практически любого видеоформата. Если вы не хотите работать с командной строкой, воспользуйтесь одним из нескольких программных пакетов (как платных, так и бесплатных), которые предлагают пользовательский интерфейс для FFmpeg, что делает его удобным для пользователя;
- Freemake (freemake.com) — бесплатный инструмент для конвертации видео и аудио для Windows, который поддерживает более 500 мультимедийных форматов.

Конвертация аудио

- Audio Converter (online-audio-converter.com) — это один из бесплатных аудио- и видеоинструментов от 123Apps.com, который конвертирует файлы в форматы MP3, WAV, OGG и другие;
- Media.io (media.io) — бесплатный веб-сервис, который конвертирует аудио в форматы MP3, WAV и OGG;
- MediaHuman Audio Converter (www.mediahuman.com/audio-converter/) — является бесплатным для Mac и Windows и выполняет конвертацию во все упомянутые в этой главе и другие аудиоформаты. Имеет простой интерфейс перетаскивания, но без значительных излишеств;
- Max (sbooth.org/Max/) — аудиоконвертер с открытым исходным кодом (только для пользователей Mac);
- Audacity (www.audacityteam.org) — это кросс-платформенное программное обеспечение с открытым исходным кодом, предназначенное для записи аудио на нескольких дорожках и последующего редактирования. Может импортировать и экспортить файлы, находящиеся во многих упомянутых в этой главе форматах.

Добавление на страницу видеоплеера

`<video>...</video>`

Добавляет на страницу видеоплеер

Пришло время создать HTML-разметку для добавления видео на веб-страницу (глава посвящена, как известно, HTML). И начнем мы с примера, когда разработка среды ведется с учетом возможностей того браузера, который будет применяться

пользователем. При этом можно предоставить только один видеоформат, применяя атрибут `src` элемента `video` (как это выполняется для элемента `img`). На рис. 10.3 показано, как в браузере Chrome демонстрируется фильм, воспроизводимый заданным по умолчанию плеером этого браузера.



Рис. 10.3. Демонстрация видеофрагмента, встроенного с помощью элемента `video` (отображается в Chrome на Mac)

Вот пример простого элемента `video`, который вставляет на веб-страницу видео, воспроизводимое в окне плеера:

```
<video src="highlight_reel.mp4" width="640" height="480"  
poster="highlight_still.jpg" controls autoplay>  
    Your browser does not support HTML5 video. Get the <a  
    href="highlight_reel.mp4">MP4 video</a>  
</video>
```

Браузеры, не поддерживающие видео, отображают любой находящийся в элементе `video` контент. В приведенном примере предоставляется ссылка на фильм, который посетитель может загрузить и воспроизвести в другом плеере.

В примере также имеются следующие атрибуты:

- `width="пиксельный размер"`, `height="пиксельный размер"` — они определяют размер области, которую занимает на экране встроенный медиаплеер. Как правило, лучше устанавливать размеры, точно соответствующие размерам окна видео в пикселях. При этом размер окна видео изменяется в соответствии с установленными здесь размерами;
- `poster="url-ссылка на изображение"` — указывает местоположение изображения, отображаемого на месте видео перед его воспроизведением;

- `controls` — добавление атрибута `controls` заставляет браузер отображать встроенные элементы управления мультимедийными объектами. Как правило, это кнопка воспроизведения/паузы, движок, который разрешает перемещаться в какую-либо позицию в окне видео, а также регуляторы громкости. Можно создать собственный пользовательский интерфейс плеера с помощью CSS и JavaScript, если между браузерами необходима большая согласованность;
- `autoplay` — приводит к автоматическому воспроизведению видео после загрузки достаточной для воспроизведения без остановки части мультимедийного файла. В общем-то, следует избегать применения атрибута `autoplay`, чтобы пользователь сам решал, когда начать воспроизводить видео. Атрибут `autoplay` не поддерживается в iOS Safari и некоторых других мобильных браузерах, что защищает пользователей от ненужных загрузок данных.

Кроме того, элемент `video` может включать атрибут `loop` — для повторного воспроизведения видео после его завершения (до бесконечности), `muted` — для воспроизведения видеодорожки без звука и `preload` — для предложения браузеру извлекать видеоданные сразу после загрузки страницы (`preload="auto"`) или подождать, пока пользователь нажмет кнопку воспроизведения (`preload="none"`). Параметр `preload="metadata"` позволяет загружать информацию о мультимедийном файле, но не сам мультимедийный файл. Устройство может решать, как обрабатывать настройку `auto`, например, браузер на смартфоне может запретить предварительную загрузку медиа, даже если установлен режим `auto`.

Поддержка параметров формата видео

Помните о рассмотренном в главе 7 подходе, когда несколько форматов изображений представляются элементом `picture` и применяется ряд элементов `source?` О да, элемент `picture` является «родственником» элемента `video`!

Как вы уже видели, нелегко найти единый формат видео, который воспримут все браузеры (хотя формат MPEG4/H.264 подходит больше всего). Кроме того, доступны новые эффективные форматы видео, такие, как VP9 и H.265, но они не поддерживаются устаревшими браузерами. Что ж, применяя элементы `source`, позволим браузерам выбрать, что именно им подходит.

В разметке ряд элементов `source` внутри элемента `video` указывают на каждый видеофайл. Браузеры просматривают список, пока не найдут файл, который поддерживают, и загрузят именно эту версию. В следующем примере для поддержки браузерами видеоклип представлен в расширенном формате WebM /VP9, а также в форматах MP4 и Ogg Theora для поддержки другими браузерами. В этом случае охвачены практически все браузеры, поддерживающие видео HTML5 (см. врезку «Запасной вариант для Flash-видео»).

```
<video id="video" controls poster="img/poster.jpg">
  <source src="clip.webm" type="video/webm">
  <source src="clip.mp4" type="video/mp4">
  <source src="clip.ogg" type="video/ogg">
  <a href="clip.mp4">Download the MP4 of the clip.</a>
</video>
```

ЗАПАСНОЙ ВАРИАНТ ДЛЯ FLASH-ВИДЕО

Устаревшие браузеры, особенно Internet Explorer версии 8 и более ранние, не поддерживают элемент `video`. Если отказ от IE8 приводит к значительному улучшению статистических показателей вашего сайта, можно выбрать запасной вариант, заключающийся в воспроизведении Flash-фильма. В упомянутой ранее статье «Creating a Cross-Browser Video Player» содержится подробное объяснение подобной методики. Я рекомендую еще одну статью: «Video for Everybody» Крока Камена (Kroc Camen) (camendesign.com/code/video_for_everybody). Материал немного устарел, но будет полезен и вполне соответствует современному уровню представлений о браузерной поддержке.

Пользовательские видеоплееры

Одной из сильных сторон элемента `video` и API-интерфейса Media Player является то, что система допускает множество настроек. С помощью таблиц стилей CSS можно изменить внешний вид кнопок управления, а при помощи JavaScript — управлять функциональностью. Поскольку описание всего этого выходит за рамки этой главы, я рекомендую вам ознакомиться со статьей Эрика Шеперда (Eric Shepherd), Криса Миллса (Chris Mills) и Иана Девлина (Ian Devlin) «Creating a Cross-Browser Video Player», вы также можете обратиться на сайт по адресу: developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/cross_browser_video_player, где содержится отличный тематический обзор.

Многих читателей может заинтересовать создание видеоплеера, имеющего привлекательный внешний вид и обеспечивающего высокую производительность, в том числе и поддержку потоковых видеоформатов. Реализовать такой видеоплеер совсем несложно, добавив в документ одну-две строки кода JavaScript, а затем применив элемент `video`. Список полезных функций видеоплеера plug-and-play имеется на ресурсе VideoSWS (videosws.praegnanz.de/).

Добавление на страницу аудиоплеера

`<audio>...</audio>`

Добавляет на страницу аудиоплеер

Если вы рассмотрели пример по разметке видео, то уже представляете себе, как добавить на страницу аудио. Элемент `audio` имеет те же атрибуты, что и элемент `video`, за исключением атрибутов `width`, `height` и `poster` (поскольку здесь нет отображаемого окна). Как и в случае с элементом `video`, используя элемент `source`, можно предоставить набор аудиоформатов, что и показано в следующем примере. А на рис. 10.4 приведен вид аудиоплеера при его отображении в браузере.

`<p>Play "Percussion Gun" by White Rabbits</p>`

```
<audio id="whiterabbits" controls preload="auto">
  <source src="percussiongun.mp3" type="audio/mp3">
```

```
<source src="percussiongun.ogg" type="audio/ogg">
<source src="percussiongun.webm" type="audio/webm">
<p>Download "Percussion Gun":</p>
<ul>
  <li><a href="percussiongun.mp3">MP3</a></li>
  <li><a href="percussiongun.ogg">Ogg Vorbis</a></li>
</ul>
</audio>
```



Рис. 10.4. Аудиоплеер, отображенный в браузере Firefox

Если у вас имеется только один аудиофайл, можно использовать для него атрибут `src`. Желая над кем-нибудь подшутить, вы можете встроить на страницу аудио, настроить его на автоматическое воспроизведение, а затем зациклить следующим образом, не предоставляя элементов управления для его останова:

```
<audio src="jetfighter.mp3" autoplay loop></audio>
```

Но вы ведь не поступите так, верно? Не так ли?! Конечно!

Добавление текстовых дорожек

`<track>...</track>`

Добавление к встроенному медиаобъекту синхронизированного текста

Элемент `track` позволяет добавлять текст, синхронизированный с временной шкалой дорожки видео или аудио. Вот некоторые варианты его применения:

- субтитры на альтернативных языках;
- подписи для слабослышащих;
- описания происходящего в видео — для слабовидящих;
- названия разделов, позволяющие перемещаться по медиаобъекту;
- метаданные, которые не отображаются, но могут применяться в сценариях.

Понятно, что добавление текстовых дорожек способствует доступности мульти-медийного объекта, а также предоставляет дополнительный бонус, улучшая поисковую оптимизацию сайта. Обеспечивается при этом и поддержка глубокой связи с определенным местом в пределах временной шкалы мультимедийного объекта. На рис. 10.5 показано, как такие дорожки отображаются в браузере, поддерживающем элемент `track`.

Применяйте элемент `track` в составе элемента `video` или `audio`, который желаете сопроводить подписями. Элемент `track` должен отображаться после всех элемен-

тов `source`, если таковые имеются, и может содержать следующие атрибуты:

- `src` — указывает текстовый файл;
- `kind` — указывает тип предоставляемой текстовой аннотации (`subtitles`, `captions`, `descriptions`, `chapters` или `metadata`). Если атрибут `kind` задан как `subtitle`, также следует с помощью стандартизированного двухбуквенного языкового тега IANA указать язык, добавив атрибут `srclang`;
- `label` — предоставляет для дорожки имя, которое можно применить в интерфейсе для выбора конкретной дорожки;
- `default` — помечает конкретную дорожку как заданную по умолчанию. Этот атрибут может применяться только на одной дорожке в элементе мультимедийного объекта.



Рис. 10.5. Видео с подписями

ДВУХБУКВЕННЫЕ ЯЗЫКОВЫЕ КОДЫ

Полный список двухбуквенных языковых кодов опубликован на сайте: www.iana.org/assignments/language-subtag-registry/language-subtag-registry.

Следующий код предоставляет для фильма варианты субтитров на английском и французском языках:

```
<video width="640" height="320" controls>
  <source src="japanese_movie.mp4" type="video/mp4">
  <source src="japanese_movie.webm" type="video/webm">
  <track src="english_subtitles.vtt"
        kind="subtitles"
        srclang="en"
        label="English subtitles"
        default>
  <track src="french.vtt"
        kind="subtitles"
        srclang="fr"
        label="Sous-titres en français">
</video>
```

Формат WebVTT

В только что приведенном примере дорожка указывает на файл с расширением `vtt`. Это текстовый файл в формате WebVTT (Web Video Text Tracks, текстовые дорожки видео в Интернете), который включает список подсказок и имеет следующий вид:

WEBVTT

```
00:00:01.345 --> 00:00:03.456
Welcome to Artifact [applause]
```

00:00:06.289 --> 00:00:09.066

There is a lot of new mobile technology to discuss.

00:00:06.289 --> 00:00:13.049

We're glad you could all join us at the Alamo Drafthouse.

Подсказки отделены друг от друга пустыми строками. Каждая подсказка имеет время начала и окончания в формате *hours:minutes:seconds:milliseconds* (часы:минуты:секунды:миллисекунды), при этом время начала и время окончания разделены «стрелкой» (→). Текст подсказки (субтитры, подпись, описание, раздел или метаданные) находится в строке ниже. Для каждой подсказки в строке над записью времени может быть указан идентификатор (ID).

Нетрудно догадаться, что текстовые дорожки в формате WebVTT — это нечто большее, чем просто текст. Чтобы разобраться в этом более детально, обратите внимание на следующие ресурсы:

- «Adding Captions and Subtitles to HTML5 Video» в MDN Web Docs (developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Adding_captions_and_subtitles_to_HTML5_video);

ДРУГИЕ СИНХРОНИЗИРОВАННЫЕ ТЕКСТОВЫЕ ФОРМАТЫ

Другие синхронизированные текстовые форматы включают субтитры SRT (зменены на WebVTT) и TML/DFXP, поддерживаемые W3C и Internet Explorer, однако они не рекомендуются в спецификации HTML5 для использования с элементом `track`.

- учебник «Subtitle tutorial on Miracle Tutorials» (www.miracletutorials.com/how-to-create/captionssubtitles-for-video-and-audio-in-webvtt-srt-dfxp-format/);
- спецификация WebVTT для W3C доступна по адресу: www.w3.org/TR/webvtt1/.

Если вы хотите поэкспериментировать с элементом `video`, выполните *упражнение 10.2*.

УПРАЖНЕНИЕ 10.2. ВСТРАИВАНИЕ ВИДЕОПЛЕЕРА

При выполнении этого упражнения видео добавляется на страницу с помощью элемента `video`. В материалах к главе 10 содержится небольшой фильм в форматах MPEG-4, OGG/Theora и WebM, посвященный тестированию аэродинамической трубы.

1. Создайте новый документ с соответствующей разметкой HTML5 или же возьмите документ, который уже использовали в *упражнении 10.1*.
2. Начните с добавления элемента `video` с атрибутом `src`, указывающим на файл `windtunnel.mp4`, поскольку видео в формате MP4 лучше поддерживается браузерами. Убедитесь, что указаны значения ширины (320 пикселов) и высоты (262 пикселя), а также атрибут `controls`, позволяющий воспроизводить и приостанавливать видео. Включите в элемент `video` резервный вариант: либо сообщение, либо ссылку на видео:

```
<video src="windtunnel.mp4" width="320"
       height="262" controls>
    Sorry, your browser doesn't support HTML5 video.
</video>
```

3. Сохраните и просмотрите документ в браузере. Если появится сообщение о сбое, ваш браузер устарел и не поддерживает элемент `video`. Если вы видите элементы управления, но не само видео, формат MP4 не поддерживается вашим браузером, поэтому попробуйте еще раз, выбрав видео другого формата.
4. Элемент `video` прост в использовании, поэтому вы уже достаточно хорошо подготовлены для работы с ним, однако желательно немного поэкспериментировать, чтобы увидеть и сравнить результаты. Далее приводится несколько вариантов для экспериментов:
 - измените размеры окна видеоплеера с помощью атрибутов `width` и `height`;
 - добавьте атрибут `autoplay`;
 - удалите атрибут `controls` (элемент управления) и посмотрите с точки зрения пользователя, какой окно воспроизведения примет вид;
 - перепишите элемент `video`, применяя элементы `source` для каждого из трех имеющихся форматов видео.

ХОЛСТ

Еще одним интересным вариантом типа «Посмотри, Ма, никаких плагинов!» в HTML5 является элемент `canvas` и связанный с ним Canvas API. Элемент `canvas` (холст) создает на веб-странице область для рисования с помощью набора функций JavaScript, служащих для формирования линий, фигур, заливок, текста, анимации и т. д. Этот элемент можно применять и для отображения иллюстрации, но наибольший потенциал элемента `canvas` связан с тем, что все отображаемое им генерируется с помощью сценариев (радуя весь мир веб-дизайна). Элемент `canvas` динамический, с его помощью можно рисовать «на лету», откликаясь на ввод пользователя. Поэтому он является отличной платформой для создания анимации, игр и даже целых приложений — и всего этого можно добиться, используя встроенное поведение браузера и не прибегая к фирменным плагинам типа Flash.

Стоит отметить, что область рисования холста основана на использовании раstra, то есть состоит из матрицы пикселов. В этом существенное отличие рисования на холсте от другого стандарта рисования — SVG, в котором используются векторные фигуры и контуры, определенные с помощью точек и математических соотношений.

Хорошая новость заключается в том, что все современные браузеры поддерживают элемент `canvas` — за исключением Internet Explorer 8 и более ранних версий. И уже давно известно, что программное обеспечение Adobe Animate (замена Flash Pro) экспортируется в формат `canvas`.

На рис. 10.6 показано несколько примеров использования элемента `canvas`: для создания игр, программ для рисования, для планетоидной анимации и в качестве инструмента изучения структуры молекул. Большое число подобных примеров

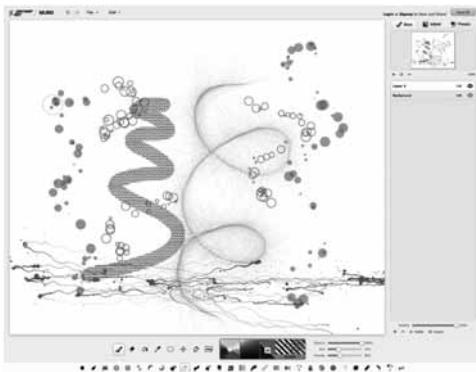
БИБЛИОТЕКА JAVASCRIPT FLASHCANVAS

Если имеется веская причина для поддержки IE8, воспользуйтесь библиотекой JavaScript FlashCanvas (flashcanvas.net), которая с помощью средств API-рисования Flash добавит поддержку элемента `canvas`.

содержится на странице EnvatoTuts+ (code.tutsplus.com/articles/21-ridiculously-impressive-html5-canvas-experiments--net-14210), в блоге Дэвида Уолша (David Walsh) (davidwalsh.name/canvas-demos) и доступно на других интернет-ресурсах.



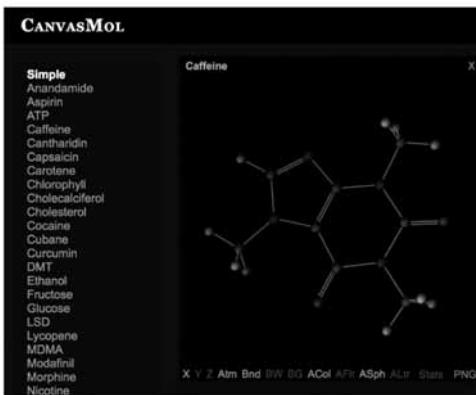
mahjong.frvr.com/



muro.deviantart.com/



www.effectgames.com/demos/canvascycle/



alteredqualia.com/canvasmol/

Рис. 10.6. Несколько примеров использования элемента canvas для создания игр, анимации и приложений

Полное освоение элемента canvas — это гораздо больше, чем мы можем сделать здесь и сейчас, особенно без какого бы то ни было опыта использования JavaScript за плечами, поэтому далее я расскажу вам о том, что такое рисовать с помощью JavaScript. Это даст вам возможность понять, как это все работает, а также увидеть насколько сложны некоторые из показанных вам примеров.

Элемент *canvas*

<canvas>...</canvas>

Добавление двумерной области для динамического рисования

Пространство холста (canvas) добавляется на страницу с помощью элемента canvas, а с помощью атрибутов width и height задаются его размеры. И это все, что нужно

для разметки. Для браузеров, не поддерживающих элемент canvas, внутри тегов можно предоставить определенный резервный контент (сообщение, изображение или еще что-либо):

```
<canvas width="600" height="400" id="my_first_canvas">
Your browser does not support HTML5 canvas. Try using Chrome,
Firefox, Safari or MS Edge.
</canvas>
```

Разметка очищает пространство, где будут создаваться рисованные объекты. На само пространство рисования можно повлиять с помощью CSS (например, добавить рамку или цвет фона), но все содержимое холста создается с помощью сценариев и не может быть выбрано для стайлинга с помощью CSS.

Рисование с помощью функций JavaScript

Библиотека API Canvas включает функции для создания фигур: такие, как `strokeRect()` — для рисования прямоугольного контура и `beginPath()` — для начала рисования линий. Некоторые функции служат для перемещения объектов — например: `rotate()` и `scale()`. Также имеются атрибуты для применения стилей — например: `lineWidth`, `font`, `strokeStyle` и `fillStyle`.

Следующий пример придумал Сандерс Кляйнфельд (Sanders Kleinfeld) для своей книги «HTML5 for Publishers» (издательство O'Reilly). С его любезного разрешения мы обратимся к этому примеру и здесь. На рис. 10.7 показан простой смайлик, который будет создан с помощью Canvas API.

Далее приведен сценарий для этого примера. Не беспокойтесь, что вы не знаете JavaScript. Просмотрите сценарий и обратите внимание на сопровождающие его комментарии. А после сценария мы рассмотрим некоторые примененные в нем функции. Я уверена, что вы отлично во всем разберетесь.

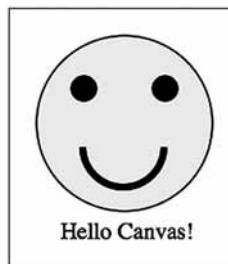


Рис. 10.7. Готовое решение для примера «Hello Canvas» (оригинал находится на сайте: examples.oreilly.com/0636920022473/my_first_canvas/my_first_canvas.html)

```
<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 рамку вокруг холста
```

```

    // рисование смайлика

my_canvas.beginPath();
my_canvas.arc(100, 100, 75, (Math.PI/180)*0, (Math.PI/180)*360, false);

    // размеры круга
my_canvas.strokeStyle = "black"; // окрашивание черным контура
                                    // круга
my_canvas.lineWidth = 3;          // контур шириной в 3 пикселя
my_canvas.fillStyle = "yellow";   // заливка круга желтым
my_canvas.stroke();              // рисование круга
my_canvas.fill();                // заливка круга
my_canvas.closePath();

    // теперь нарисуем левый глаз
my_canvas.fillStyle = "black"; // выбор черного цвета для заливки
my_canvas.beginPath();
my_canvas.arc(65, 70, 10, (Math.PI/180)*0, (Math.PI/180)*360,
false);

    // размеры круга
my_canvas.stroke();           // рисование круга
my_canvas.fill();             // заливка круга
my_canvas.closePath();

    // теперь нарисуем 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();           // рисование круга
my_canvas.fill();             // заливка круга
my_canvas.closePath();

    // рисование улыбки
my_canvas.lineWidth = 6;        // выбор 6-пиксельного контура
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();

    // сообщение от смайлика!
my_canvas.fillStyle = "black"; // выбор черной текстовой заливки
my_canvas.font = '20px sans'; // 20-пиксельный шрифт без засечек
my_canvas.fillText ("Hello Canvas!", 45, 200); // запись текста
}
</script>

```

А теперь немного информации о применяемых в примере с Canvas API функциях:

- `strokeRect(x1, y1, x2, y2)` — рисует прямоугольный контур от точки (x1, y1) до точки (x2, y2). По умолчанию рисование начинается в левом верхнем углу холста, где находится точка (0, 0), а координаты x и y изменяются вправо и вниз;

- `beginPath()` — начинает рисование линий;
- `closePath()` — завершает рисование линий, начатое с помощью функции `beginPath()`;
- `arc(x, y, arc_radius, angle_radians_beg, angle_radians_end)` — рисует дугу (`arc`), где (`x, y`) — центр окружности, `arc_radius` — длина радиуса окружности, а `angle_radians_beg` и `_end` указывают начальный и конечный углы дуги;
- `stroke()` — рисует линию, определенную заданным путем. Если не активизировать эту опцию, линия не отобразится на холсте;
- `fill()` — заполняет путь, указанный с помощью функций `beginPath()` и `endPath()`;
- `fillText(your_text, x1, y1)` — добавляет текст к холсту, начиная с указанной координаты (`x, y`).

Кроме того, для указания цветов и стилей применялись следующие атрибуты :

- `lineWidth` — ширина границы контура;
- `strokeStyle` — цвет этой границы;
- `fillStyle` — цвет заливки (внутренней части) фигуры, созданной с помощью контура;
- `font` — шрифт и размер текста.

Конечно, Canvas API содержит гораздо больше функций и атрибутов, чем здесь использовано. Полный их список содержится в W3C HTML5 Canvas 2D Context спецификации, доступной по адресу: www.w3.org/TR/2dcontext. Поиском в Интернете вы найдете множество учебных пособий по работе с холстом. Кроме того, я могу порекомендовать следующие ресурсы:

- книга Стива Фултона (Steve Fulton) и Джеффа Фултона (Jeff Fulton) «HTML5 Canvas», шестое издание, издательство O'Reilly;
- если вы любите смотреть видео, ознакомьтесь с уроком Дэвида Гири (David Geary) «HTML5 Canvas for Developers», размещенным по адресу: shop.oreilly.com/product/0636920030751.do.

КОНТРОЛЬНЫЕ ВОПРОСЫ

В этой главе рассмотрены всевозможные способы размещения объектов на веб-страницах. Показано, каким образом можно применять элементы: `iframe` — для формирования «окна в окне» при отображении внешних веб-ресурсов и `object` — для ресурсов, нуждающихся в плагинах, как пользоваться видео- и аудиоплеерами и как создать двумерный сценарий `canvas` для рисования. А теперь посмотрим, достаточно ли вы были внимательны при рассмотрении этих вопросов. Как всегда, ответы на вопросы находятся в *приложении 1*.

1. Что такое «вложенный контекст просмотра» и как его создать?
2. Почему атрибут `sandbox` применяется с `iframe`?
3. Укажите несколько случаев, когда необходимо знать MIME-тип медиафайла.
4. Идентифицируйте каждое из следующих названий как формат контейнера, видеокодек или аудиокодек:
 - a. Ogg _____
 - b. H.264 _____
 - c. VP8 _____
 - d. Vorbis _____
 - e. WebM _____
 - f. Theora _____
 - g. Opus _____
 - h. MPEG-4 _____
5. Что реализует атрибут `poster`?
6. Чем является файл `vtt`?
7. Укажите не менее двух различий между SVG и Canvas.
8. Приведите две функции Canvas API, которые можно применить при рисовании прямоугольника, и залейте его красным. Можно не писать сценарий полностью.

ОБЗОР ЭЛЕМЕНТОВ: ВСТРОЕННЫЕ МУЛЬТИМЕДИЙНЫЕ ОБЪЕКТЫ

В табл. 10.3 приведены элементы, применяемые для встраивания в веб-страницы различных типов мультимедийных файлов.

Таблица 10.3. Элементы, применяемые для встраивания в веб-страницы различных типов мультимедийных файлов

Элемент и атрибуты	Описание
<code>audio</code>	Вставляет на страницу аудиоплеер
<code>src="URL-ссылка"</code>	Адрес ресурса
<code>crossorigin="anonymous use-credentials"</code>	Указывает, как элемент обрабатывает запросы от других источников (серверов)
<code>preload="auto none metadata"</code>	Указывает, сколько мультимедийных ресурсов должно при загрузке страницы помещаться в буфер
<code>autoplay</code>	Указывает, что мультимедийный объект может воспроизвести, как только загружена страница
<code>Loop</code>	Указывает, что мультимедийный файл должен начать воспроизведение автоматически после достижения конца воспроизведения
<code>muted</code>	Отключает вывод звука

Табл. 10.3. (продолжение)

Элемент и атрибуты	Описание
controls	Указывает, что браузер должен отображать набор элементов управления воспроизведением для мультимедийного файла
canvas	Представляет двумерную область, которая может использоваться для визуализации динамической растровой графики
height	Высота области холста
width	Ширина области холста
embed	Встраивает мультимедийный объект, который требует плагин для воспроизведения на странице. Некоторые типы носителей требуют пользовательских атрибутов, не перечисленных далее
src= "URL-ссылка"	Адрес медиаресурса
type= "тип мультимедиа"	Тип мультимедийного объекта (MIME)
width= "число"	Горизонтальное измерение видеоплеера в пикселях
height= "число"	Вертикальное измерение видеоплеера в пикселях
iframe	Создает вложенный контекст просмотра для отображения на странице HTML ресурсов.
src= "URL-ссылка"	Адрес HTML-ресурса
srcdoc= "исходный код HTML"	Исходный код HTML документа для отображения во встроенном фрейме
name= "текст"	Присваивает встроенному фрейму имя, на которое формируются затем целевые ссылки
sandbox= "allow-forms allow-pointer-lock allow-popups allow-same-origin allow-scripts allow-top-navigation"	Правила безопасности для вложенного контента
allowfullscreen	Указывает, что объектам во встроенном фрейме разрешено использовать requestFullScreen()
width= "число"	Горизонтальное измерение окна видеоплеера в пикселях
height= "число"	Вертикальное измерение окна видеоплеера в пикселях
object	Универсальный элемент для встраивания внешнего ресурса
data= "URI-ссылка"	Адрес ресурса
type= "тип мультимедиа"	Мультимедийный тип (MIME) ресурса
Typemustmatch	Указывает, что ресурс должен применяться, только если совпадают значение типа атрибута и типа контента ресурса
name= "текст"	Имя объекта, на который ссылаются скрипты
form= "ID формы"	Связывает объект с элементом form

Табл. 10.3. (окончание)

Элемент и атрибуты	Описание
<code>width="число"</code>	Горизонтальное измерение окна видеоплеера в пикселях
<code>height="число"</code>	Вертикальное измерение окна видеоплеера в пикселях
<code>param name="текст" value="текст"</code>	Предоставляет параметр в элементе <code>object</code> Определяет имя параметра Определяет значение параметра
<code>source src="текст" type="тип мультимедиа"</code>	Позволяет авторам указывать несколько версий медиафайла (используется с видео и аудио) Адрес ресурса Тип мультимедийного (MIME) ресурса
<code>track kind="subtitles captions descriptions chapters metadata" src="текст" srclang="тег корректного языка" label="текст" default</code>	Определяет внешний ресурс (текст или аудио), который синхронизирован с мультимедийным файлом, улучшая доступность, навигацию или SEO Тип текстовой дорожки Адрес внешнего ресурса Язык текстовой дорожки Название дорожки, которое может отображаться браузером по умолчанию Указывает, что дорожка должна использоваться по умолчанию, если она не переопределяет пользовательские настройки
<code>video src="URL-ссылка" crossorigin="anonymous use-credentials" poster="URL-ссылка" preload="auto none metadata" autoplay loop muted controls width="число" height="число"</code>	Вставляет на страницу видеопроигрыватель Адрес ресурса Каким образом элемент обрабатывает запросы, поступающие от других источников (серверов) Расположение файла изображения, который отображается в качестве заполнителя до начала воспроизведения видео Подсказки: сколько потребуется буферизации мультимедийного ресурса Указывает, что мультимедийный объект может воспроизводиться, как только страница загружена Указывает, что мультимедийный файл должен начать воспроизведение автоматически после достижения конца Отключает вывод звука Указывает, что браузер должен отображать набор элементов управления процессом воспроизведения мультимедийного файла Определяет горизонтальное измерение окна видеоплеера в пикселях Определяет вертикальное измерение окна видеоплеера в пикселях

ЧАСТЬ III

ПРАВИЛА CSS ДЛЯ ПРЕДСТАВЛЕНИЯ HTML-ДОКУМЕНТОВ

ГЛАВА 11

ВВЕДЕНИЕ В КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ

В этой главе...

- ▶ *Преимущества и мощь CSS*
- ▶ *Формирование структуры документа с помощью HTML-разметки*
- ▶ *Запись стилевых правил*
- ▶ *Присоединение стилей к HTML-документу*
- ▶ *Важные понятия: наследование, специфичность, каскадность, порядок правил и блочная модель*

Каскадные таблицы стилей (CSS) уже неоднократно упоминались в этой книге ранее, и теперь, наконец, пришло время задействовать их и придать страницам столь необходимый им стиль. Каскадные таблицы стилей — это стандарт W3C, предназначенный для придания документам, написанным на HTML, а, фактически, на любом XML-языке, соответствующего *представления*. Понятие «представление» охватывает также и способ доставки документа пользователю — независимо от того, отображается ли документ на мониторе компьютера, экране мобильного телефона, печатается на бумаге или произносится вслух программой чтения с экрана. С помощью таблиц стилей, создающих представление, HTML может справиться с задачей определения структуры и значения документа.

Каскадные таблицы стилей представляют собой отдельный язык, обладающий собственным синтаксисом. В этой главе рассматриваются терминология CSS и основные их понятия, помогающие ориентироваться в последующих главах, где речь идет о том, каким образом можно изменять стили текста и шрифтов, добавлять цвета и фоны и даже формировать базовый макет страницы. К концу *третьей части* книги вы получите прочную основу для дальнейшего самостоятельного тематического чтения и практики.

ПРЕИМУЩЕСТВА CSS

Полагаю, вас не нужно убеждать, что таблицы стилей — весьма полезный объект для изучения и применения, однако краткое изложение преимуществ, связанных с применением таблиц стилей, не помешает. Итак, к преимуществам CSS относятся:

- **возможность точной настройки шрифтов и макета страниц** — используя CSS, можно достичь высокой точности печати. Существует набор свойств, специально предназначенных для оформления и вывода печатной страницы (впочем, они не входят в круг вопросов, рассматриваемых в этой книге);
- **уменьшение объема работы** — при использовании CSS можно изменять внешний вид сайта путем модификации всего лишь одной таблицы стилей. Таким образом поддерживается согласованность форматирования всего сайта;
- **улучшение доступности сайтов** — если все проблемы, связанные с представлением, решаются с помощью CSS, можно более осознанно размещать контент, улучшая его доступность для невизуальных средств и мобильных устройств.

Если вдуматься, то обращение к таблицам стилей несет исключительно одни положительные моменты. Конечно, возникают определенные неприятные ситуации из-за наличия у отдельных браузеров несовместимости или ряда ограничений, но можно их либо избежать, либо устраниить, если известна их природа.

МОЩЬ CSS

Сейчас мы не станем обсуждать незначительные визуальные улучшения типа изменения цвета заголовков или добавления в текст абзацных отступов. При умелом использовании возможностей CSS в вашем распоряжении окажется надежный и универсальный инструмент для дизайна. Обширные возможности применения CSS для дизайна стали мне более понятны, когда я познакомилась с разнообразием и богатством дизайнерских разработок, выполненных с помощью CSS, на сайте CSS Zen Garden (www.csszengarden.com).

Уже тогда (в далеком 2003-м году), когда разработчики еще не решались отказаться от макетирования на основе таблиц в пользу CSS, сайт CSS Zen Garden Дэвида Ши (David Shea) продемонстрировал поразительные возможности, которые открываются при использовании лишь одних таблиц стилей. Дэвид опубликовал HTML-документ и пригласил дизайнеров добавить собственные таблицы стилей, которые привнесли бы в него визуальный дизайн. На рис 11.1 показаны лишь некоторые варианты из наиболее привлекательных разновидностей дизайна. Причем, во всех проектах за основу взят единственный исходный HTML-документ.

Дело не только в том, что эти варианты не содержат единственный элемент `img` (все изображения размещены на заднем плане). Просто посмотрите, насколько разнится внешний вид каждой страницы и с каким вкусом они выполнены. Этого удалось добиться с помощью таблиц стилей. Вот вам еще одно доказательство того, что таблицы CSS отделены от HTML, а представление отделено от структуры.



CSS Zen Dragen
by Matthew Buchanan



By the Pier
by Peter Ong Kelmscott



Organica Creativa
by Eduardo Cesario



ShaolinYokobue
by Javier Cabrera

Рис. 11.1. Страницы с сайта CSS Zen Garden используют один и тот же исходный HTML-документ, а различиям в дизайне они обязаны исключительно средствам CSS (иллюстрация публикуется с разрешения сайта CSS Zen Garden и авторов-дизайнеров)

Сайт CSS Zen Garden более не обновляется и теперь уже стал просто историческим документом, свидетельствующим о поворотном моменте в принятии веб-стандартов. Несмотря на давность представленных на нем разработок, это хороший универсальный урок для демонстрации возможностей CSS.

Конечно, для создания CSS-макетов, подобных показанным на рис. 11.1, требуется большая практика. Серьезную подмогу в их разработке также окажут профессиональные навыки графического дизайна (к сожалению, для такого материала нет места в этой книге). Я продемонстрировала вам все это заранее, поскольку хочу, чтобы вы знали о безграничных возможностях CSS-дизайна, а также и потому, что примеры, приведенные в этой книге для начинающих, как правило, просты и понятны. Учитесь на них, но не забывайте о вершинах мастерства.

КАК РАБОТАЮТ ТАБЛИЦЫ СТИЛЕЙ?

На самом деле все просто, как 1-2-3!

1. Начните с документа, который был размечен в HTML.
2. Создайте правила стиля, учитывающие желательный вид нужных элементов.

3. Прикрепите к документу правила стиля. Браузер, отображая этот документ, учтет ваши правила представления элементов (если пользователь не применил какие-либо обязательные стили, но об этом речь пойдет позже).

Все это хорошо, но, конечно, имеются и нюансы. Рассмотрим подробнее каждый из этих шагов.

Шаг 1. Разметка документа

О разметке контента мы уже достаточно много говорили в предыдущих главах. Например, важно выбирать элементы, точно описывающие содержание контента. Речь также шла о том, что разметка создает структуру документа, иногда называемую *структурным уровнем (слоем)*, к которому можно применить *уровень (слой) представления*.

В этой и последующих главах вы увидите, что понимание структуры вашего документа и взаимосвязей между его элементами является центральным компонентом вашей работы в качестве автора таблицы стилей.

В упражнениях, которые вы будете выполнять при чтении этой главы, показано, как можно быстро и легко изменять внешний вид документа с помощью таблиц стилей. Чтобы облегчить вашу задачу, я подготовила небольшой HTML-документ, с которым вы сможете экспериментировать. При выполнении *упражнения 11.1* вы с ним познакомитесь.

УПРАЖНЕНИЕ 11.1. ПЕРВОЕ ЗНАКОМСТВО

В процессе выполнения упражнений из этой главы в небольшой фрагмент документа будут добавлены несколько простых стилей. С сайта книги по адресу:

learningwebdesign.com/5e/materials/ загрузите документ **cooking.html** и связанное с ним изображение **salads.jpg**.

Откройте документ в браузере и посмотрите, какой он имеет вид по умолчанию (должен выглядеть примерно так, как показано на рис. 11.2). Вы также можете открыть документ в текстовом редакторе и подготовиться к выполнению следующих двух упражнений.



Рис. 11.2. Представление документа в браузере при отсутствии инструкций из таблицы стилей. Мы не ставим цель добиться превосходного качества, а всего лишь хотим ознакомиться с принципами работы таблицы стилей

Шаг 2. Создание правил таблицы стилей

Таблица стилей состоит из одной или нескольких инструкций по оформлению стиля (называемых *правилами стиля*), которые описывают методику отображения элемента или группы элементов. Изучение CSS начинается со знакомства с синтаксисом этих правил. Как вы увидите, он достаточно прост. Каждое правило *выбирает* элемент и *объявляет*, как он должен выглядеть.

Следующий пример включает два правила. Первое — перекрашивает все элементы уровня `h1`, имеющиеся в документе, в зеленый цвет, а второе — указывает, что абзацы должны оформляться крупным шрифтом типа `sans-serif` (без засечек). Шрифты «без засечек» не имеют небольших засечек (serif) на концах штрихов и выглядят, как правило, более гладкими и современными:

```
h1 { color: green; }
p { font-size: large; font-family: sans-serif; }
```

В CSS-терминологии двумя основными разделами правила являются *селектор*, идентифицирующий элемент или элементы, на которые оказывается воздействие, и *объявление*, содержащее инструкции по визуализации. *Объявление*, в свою очередь, состоит из *свойства* (например, `color`) и его *значения* (`green`), которые разделены двоеточием и пробелом. Одно или несколько объявлений заключаются в фигурные скобки, как показано на рис. 11.3.

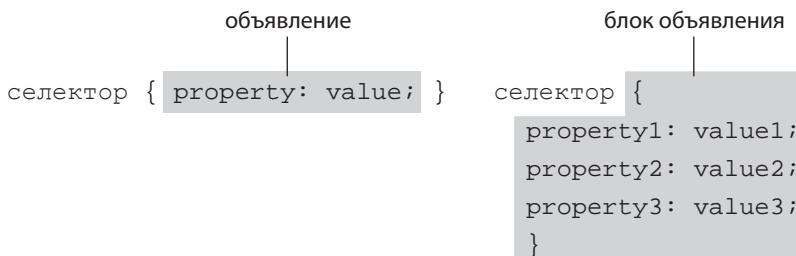


Рис. 11.3. Разделы стилевого правила

Селекторы

В предыдущем небольшом примере создания таблицы стилей элементы `h1` и `p` играют роль селекторов. Такой селектор называется *селектором типа элемента* и является основным применяемым типом селектора. Свойства, заданные для каждого правила, применяются к каждому элементу `h1` и `p` документа, соответственно.

Другим типом селектора является ID-селектор, который выбирает элемент на основе значения его атрибута `id`. Такой селектор обозначается символом `#`. Например, селектор `#recipe` указывает на элемент с `id="recipe"`.

В последующих главах вы познакомитесь с более сложными селекторами, применяемыми для указания элементов, включая способы выбора групп элементов и элементов, которые имеют определенный контекст. Более подробно об этом рассказывается во врезке «Селекторы, рассматриваемые в этой книге».

СЕЛЕКТОРЫ, РАССМАТРИВАЕМЫЕ В ЭТОЙ КНИГЕ

Вместо того чтобы привести здесь исчерпывающее изложение материала, относящегося к селекторам, я буду вводить вас в курс дела постепенно. Вы просто обратите внимание на перечень глав, в которых рассматриваются селекторы:

- **глава 11:**

- ◆ селекторы типа элемента;
- ◆ сгруппированные селекторы;

- **глава 12:**

- ◆ селекторы ID и класса;
- ◆ селекторы потомков (*child selectors*), селекторы *next-sibling* (ближайших братьев и сестер) и *following-sibling* (последующих братьев и сестер);
- ◆ универсальные селекторы;

- **глава 13:**

- ◆ селекторы псевдоклассов;
- ◆ селекторы псевдоэлементов;
- ◆ селекторы атрибутов.

ЗАКАНЧИВАЙТЕ ОБЪЯВЛЕНИЕ ТОЧКОЙ С ЗАПЯТОЙ

Технически после последнего объявления в блоке не требуется указывать точку с запятой, но рекомендуется всегда заканчивать объявление точкой с запятой. Это значительно упростит добавление объявлений к правилу.

Обратите внимание, что ничего не изменилось: используется один набор фигурных скобок, точки с запятой стоят после каждого объявления и так далее. Единственное различие — во вставке символов возврата строк и некоторых пробелов для выравнивания.

Свойства

В основе таблиц стилей лежит набор стандартных свойств, применяемых к выбранным элементам. Полная CSS-спецификация определяет десятки свойств, определяющих буквально все: от отступов текста до порядка чтения в слух заголовков таблиц. В этой книге рассматриваются наиболее распространенные и лучше всего поддерживаемые свойства, которые можно начать применять сразу.

Овладение селекторами, выбор наилучшего типа селектора и стратегия его применения — важный шаг в освоении CSS.

Объявления

Объявление состоит из пары «свойство/значение». В одном правиле может быть несколько объявлений; например, показанное ранее в примере кода правило для элемента *p* имеет два свойства: *font-size* и *font-family*. Каждое объявление должно завершаться точкой с запятой, которая отделяет его от следующего объявления. Если пропустить точку с запятой, само объявление и последующее за ним объявление будут проигнорированы. Фигурные скобки и содержащиеся в них объявления часто называют *блоком объявлений* (см. рис. 11.3).

Поскольку CSS игнорирует пробелы и символы возврата строки в блоке объявления, авторы обычно пишут в блоке каждое объявление на отдельной строке, как показано в следующем примере. Это облегчает поиск примененных к селектору свойств и указывает, когда именно завершается правило стиля:

```
p {
    font-size: large;
    font-family: sans-serif;
}
```

Значения

Значения зависят от свойств. Одни свойства предоставляют линейный размер, другие — цветовое значение, а третьи — заданный список ключевых слов. Применяя свойство, важно знать, какие оно принимает значения, однако во многих случаях вам поможет простой здравый смысл. Инструменты разработки, такие, как Dreamweaver или Visual Studio, подсказывают подходящие значения. Итак, прежде чем продолжить, почему бы не попрактиковаться в самостоятельном создании правил стиля при выполнении *упражнения 11.2*?

УПРАЖНЕНИЕ 11.2. ВАША ПЕРВАЯ ТАБЛИЦА СТИЛЕЙ

Откройте файл `cooking.html` в текстовом редакторе. В разделе `head` документа находится уже установленный элемент `style`, подготовленный для ввода в него правил. Элемент `style` как раз и служит для встраивания таблицы стилей в HTML-документ. Для начала добавим небольшую таблицу стилей, рассмотренную нами ранее. Для этого введите в документ следующие правила в том виде, в котором они здесь представлены:

```
<style>
  h1 {
    color: green;
  }
  p {
    font-size: large;
    font-family: sans-serif;
  }
</style>
```

Сохраните файл и посмотрите на него в браузере. Наверняка вы заметите некоторые изменения (если браузер исходно использует шрифт без засечек, можно будет заметить только изменение размера шрифта). Если изменения незаметны, вернитесь и убедитесь, что не пропущены открывающая и закрывающая фигурные скобки и точки с запятой. Эти символы легко пропустить, из-за чего таблица стилей не сработает.

Отредактируем таблицу стилей и заметим при этом, насколько легче писать правила, наблюдая последствия изменений. Далее приведено несколько вариантов экспериментов.

ВАЖНО!

Помните, что нужно сохранять документ после каждого изменения, чтобы изменения отображались в браузере при перезагрузке документа.

1. Сделайте элемент `h1` серым (`gray`) и посмотрите на него в браузере. Окрасьте его в синий цвет (`blue`), наконец, сделайте его оранжевым (`orange`) — полный список доступных названий цветов содержится в главе 3.
2. Добавьте новое правило, которое окрашивает элементы `h2` в оранжевый цвет.
3. С помощью следующего объявления добавьте 100-пиксельное левое поле к элементам абзаца (`p`):

```
margin-left: 100px;
```

Помните, что новое объявление можно добавить к имеющемуся правилу для элементов `p`.

4. Добавьте 100-пиксельное левое поле к заголовкам `h2`.
5. С помощью следующего объявления добавьте оранжевую 1-пиксельную границу к нижней части элемента `h1`:

```
border-bottom: 1px solid orange;
```

6. Переместите изображение к правому краю и заставьте текст обтекать его с помощью свойства `float`. Показанное в этом правиле сокращенное свойство `margin` добавляет нулевые пиксели пространства вверху и внизу изображения, а также 12 пикселов пространства слева и справа от изображения (значения отображаются таким образом, как это описано в главе 4):

```
img {  
    float: right;  
    margin: 0 12px;  
}
```

После выполнения этих действий документ примет вид, близкий к показанному на рис. ЦВ-11.4.

Шаг 3. Прикрепление стилей к документу

В предыдущем упражнении таблица стилей была вложена в документ с применением элемента `style`. Этот способ является только одним из трех вариантов применения стиля к HTML-документу. Вскоре вы сможете опробовать каждый из них, однако лучше заранее ознакомиться с методиками и терминологией.

- **Внешние таблицы стилей.** Внешняя таблица стилей — отдельный текстовый документ, содержащий несколько правил стилей. Файл документа должен иметь расширение `css`. Этот документ связывается (с помощью элемента `link`) или импортируется (с помощью правила `@import` из таблицы стилей) в один или несколько HTML-документов. Таким образом, все файлы сайта могут ориентироваться на одну таблицу стилей. Этот метод — универсальный и наиболее предпочтительный вариант для прикрепления таблиц стилей к контенту (мы подробно рассмотрим внешние таблицы стилей и начнем применять их в упражнениях из главы 13).
- **Вложенные таблицы стилей.** Этот тип таблицы стилей применен нами в упражнении 11.2. Он помещается в документ с помощью элемента `style`, а его правила применяются только к этому документу. Элемент `style` должен помещаться в заголовке документа. В следующем примере элемент `style` содержит комментарий (см. врезку «Комментарии в таблицах стилей»):

```
<head>  
    <title>Required document title here</title>  
    <style>  
        /* style rules go here */  
    </style>  
</head>
```

- **Встроенные стили.** Можно применять свойства и значения к одному элементу, используя атрибут `style` в самом элементе, как это показано в следующем примере:

```
<h1 style="color: red">Introduction</h1>
```

Для добавления нескольких свойств разделите их точкой с запятой — например, так:

```
<h1 style="color: red; margin-top: 2em">Introduction</h1>
```

Встроенные стили применяются только к конкретному элементу, у которого они отображаются. Следует избегать встроенных стилей, за исключением случаев, если абсолютно необходимо переопределить стили, берущиеся из вложенной или внешней таблицы стилей. Проблема встроенных стилей в том, что они вклинивают в структурную разметку информацию о представлении. Они затрудняют внесение изменений, поскольку для этого каждый их атрибут `style` придется искать в источнике.

При выполнении *упражнения 11.3* вы сможете создать встроенный стиль и посмотреть, как он функционирует. Но далее встроенные стили в книге не рассматриваются по указанным здесь причинам, поэтому используйте свой шанс прямо сейчас.

УПРАЖНЕНИЕ 11.3. ПРИМЕНЕНИЕ ВСТРОЕННОГО СТИЛЯ

Откройте документ `cooking.html` в том состоянии, как вы оставили его в последний раз при выполнении *упражнения 11.2*. Если упражнение выполнено до конца, в документе присутствует правило, которое окрашивает элементы `h2` в оранжевый цвет.

Создайте встроенный стиль, который назначит для второго элемента `h2` серый цвет. Вставьте его после открывающего тега `h2`, воспользовавшись атрибутом `style`, как показано здесь:

```
<h2 style="color: gray">The  
Main Course</h2>
```

Обратите внимание, что это должен быть серый цвет «gray» — через «а» (а не серый цвет «grey» через «е»), поскольку именно так этот цвет определяется в спецификации. Сохраните файл и откройте его в браузере. Теперь второй заголовок окрасится в серый цвет, поскольку оранжевый цвет, установленный во встроенной таблице стилей, будет переопределен встроенным стилем. При этом другой заголовок `h2` остается неизменным.

КОММЕНТАРИИ В ТАБЛИЦАХ СТИЛЕЙ

Иногда в таблице стилей полезно оставлять комментарии. В CSS имеется собственный показанный здесь синтаксис комментариев:

```
/* Здесь находится комментарий */
```

- ▶ Контент `/ * и * /` игнорируется при разборе таблицы стилей, поэтому можно оставлять комментарии в любом месте таблицы стилей, даже внутри правила :

```
body {
    font-size: small;
    /* Это нужно изменить позже */
}
```

Одно из применений комментариев — выделение разделов таблицы стилей для облегчения дальнейшего поиска, например:

```
/* СТИЛИ КОЛОНТИТУЛА */
```

CSS-комментарии полезны для временного сокрытия объявлений стилей в процессе разработки. Экспериментируя с несколькими стилями, можно быстро отключать стили, заключая их в символы `/ * и * /`, проверяя в браузере дизайн, а затем удалять символы комментария, чтобы стиль отобразился снова. Это намного быстрее, чем выполнять повторный ввод кода.

ВАЖНЫЕ ПОНЯТИЯ

Существует ряд важных понятий, которые нужно усвоить, чтобы в дальнейшем вам было комфортно использовать каскадные таблицы стилей. Я собираюсь ознакомить вас с этими понятиями сейчас, чтобы нам не пришлось отвлекаться на изучение теоретических вопросов, когда мы начнем заниматься непосредственно свойствами стиля. К каждому из этих понятий мы вновь вернемся и рассмотрим его более подробно в последующих главах.

Наследование

У вас глаза такого же цвета, что и у ваших родителей? Унаследовали ли вы цвет их волос? Точно так же, как родители передают детям какие-либо свои особенности, HTML-элементы, к которым применены стилевые правила, передают определенные свойства стиля содержащимся в них элементам. При выполнении *упражнения 11.1* в процессе стайлинга элементов `p` крупным шрифтом без засечек текст во втором абзаце, помеченный элементом `em`, увеличился и тоже стал отображаться без засечек, хотя для него не было создано специального правила (рис. 11.5). Дело в том, что элемент `em` *унаследовал* стили от абзаца, в котором он находится.

Фрагмент текста без стилей

I've been doing a lot of **talking** about cooking

Фрагмент текста с примененными стилями

I've been doing a lot of **talking** about cooking

Текст под элементом `em` крупный без засечек, хотя для него отсутствуют собственные правила стиля. Этот элемент наследует стили от содержащего его абзаца.

Рис. 11.5. Элемент `em` наследует примененные к абзацу стили

Наследование предоставляет механизм стайлинга элементов, которые не имеют собственных явных правил стиля.

Структура документа

Именно на этом этапе ощущается важность понимания структуры документа. Как отмечалось ранее, HTML-документы характеризуются четкой структурой, или иерархией. Например, в примере, который мы уже рассматривали ранее (файл `cooking.html`), имеется корневой элемент `html`, содержащий элементы `head` и `body`, а элемент `body` содержит элементы заголовка и абзаца. Несколько абзацев, в свою очередь, содержат встроенные элементы — такие, как изображения (`img`) и выделенный текст (`em`). Структуру документа можно представить себе как перевернутое дерево, разветвляющееся от корня (рис. 11.6).

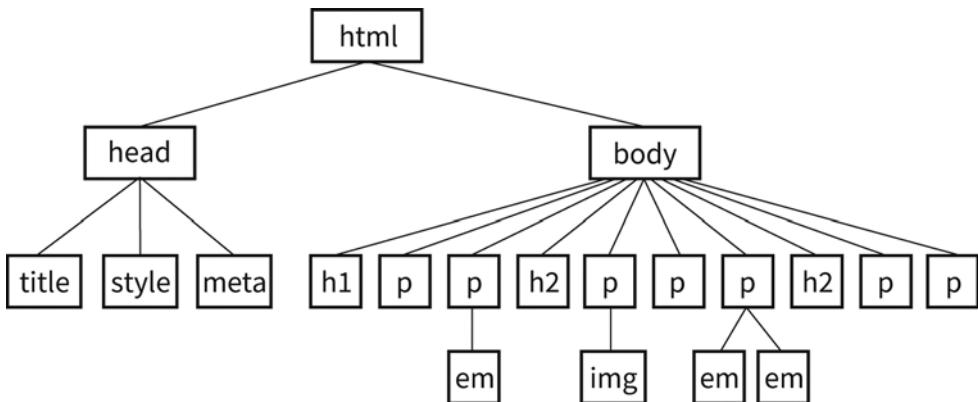


Рис. 11.6. Древовидная структура для документа `cooking.html`

Предки и потомки

Дерево документа становится родословным деревом, когда речь заходит о связях между элементами. Все элементы, содержащиеся в том или ином элементе, называются его *потомками*. Так, элементы `h1`, `h2`, `p`, `em` и `img` в представленном на рис. 11.6 документе являются потомками элемента `body`.

Элемент, непосредственно содержащийся в другом элементе (без промежуточных иерархических уровней), называется *дочерним элементом* данного элемента. И наоборот, содержащий его элемент является *родителем*. Например, элемент `em` является дочерним элементом для элемента `p`, а элемент `p` является его родителем.

Все элементы, расположенные в иерархии выше определенного элемента, являются его *предками*. Два элемента с одним и тем же родителем будут *братьями* и/или *сестрами*. Углубляясь далее и вести речь о «тетушках» или «двоюродных братьях» мы не станем, так что аналогия на этом заканчивается. Все это может показаться чересчур академичным, но пригодится при написании CSS-селекторов.

О передаче свойств

Когда вы пишете правило стиля, связанного со шрифтом, используя элемент `p` в качестве селектора, это правило применяется ко всем абзацам в документе, а также ко всем текстовым элементам, которые в них входят. Ранее было показано (см. рис. 1.5), что элемент `em` наследует свойства стиля, примененные к его родителю, — элементу `p`. На рис. 11.7 показано, как это выглядит с точки зрения структурной схемы документа. Обратите внимание, что элемент `img` из схемы исключен, поскольку свойства шрифта не применяются к изображениям.

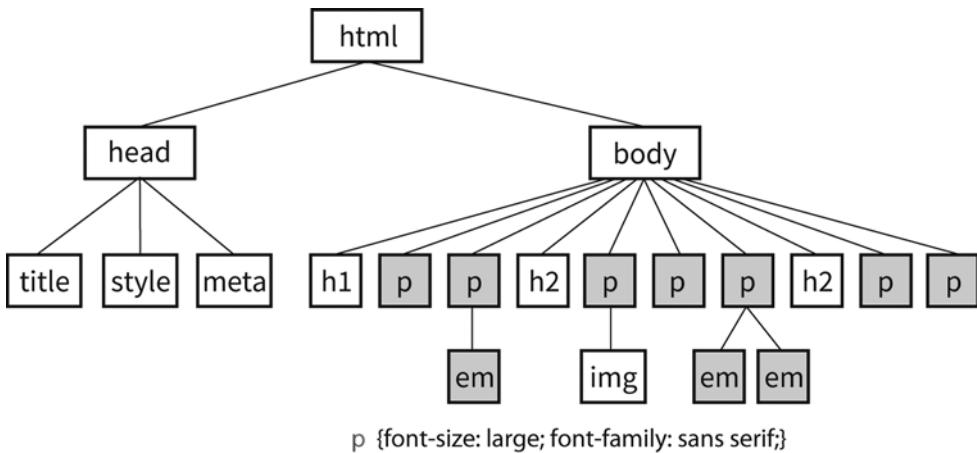


Рис. 11.7. Наследование дочерними элементами некоторых свойств, примененных к элементу `p`

Обратите внимание, что наследуются «некоторые» свойства. Важно отметить, что какие-то свойства таблицы стилей наследуются, а какие-то — нет. Как правило, свойства, связанные со стайлингом текста: размер шрифта, цвет, тип и т. д., — наследуются, то есть передаются. Такие свойства, как границы, поля, фон и т. п., влияющие на вид области, окружающей элемент, как правило, не передаются. И в этом — если вдуматься — есть глубокий смысл. Например, если вы заключите какой-либо абзац в рамку, это не значит, что заключенным в рамку окажется каждый входящий в абзац строчный элемент (такой, как `em`, `strong` или `a`).

Свойство наследования можно использовать в своих интересах при создании правил таблиц стилей. Например, если необходимо, чтобы все текстовые элементы в документе имели синий цвет, не обязательно создавать отдельные правила стиля с заданием синего цвета для каждого элемента в документе. Лучше написать единственное правило стиля, которое применяет свойство `color` к элементу `body`, и пусть все содержащиеся в `body` элементы наследуют этот стиль (рис. 11.8).

Любое примененное к конкретному элементу свойство, переопределяет унаследованные этим свойством значения. Возвращаясь к рассматриваемому примеру, отметим, что указание по поводу того, что элемент `em` должен быть оранжевым, отменяет унаследованную установку на синий цвет.

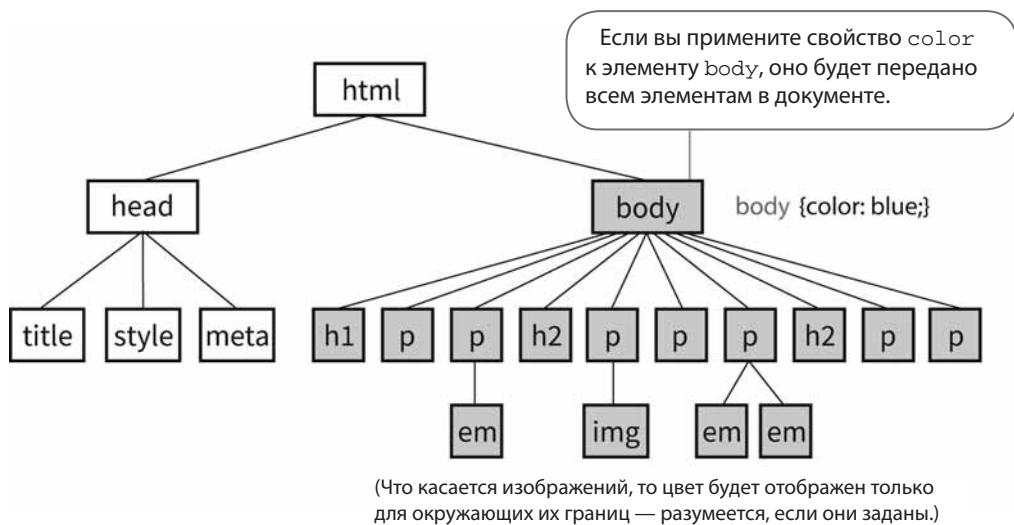


Рис. 11.8. Все элементы в документе наследуют некоторые свойства, примененные к элементу `body`

УТОЧНЯЙТЕ, НАСЛЕДУЕТСЯ ЛИ ИСПОЛЬЗУЕМОЕ ВАМИ СВОЙСТВО

Разбираясь с новым для вас свойством, полезно уточнить, наследуется ли оно. Наследование отмечено для каждого рассматриваемого в книге свойства. В большинстве случаев наследование соответствует ожиданиям веб-разработчиков.

Конфликтующие стили: каскад

Вы никогда не задумывались, почему таблицы стилей называют «каскадными»? Дело в том, что они позволяют применять несколько таблиц стилей к одному документу, и эта их возможность чревата конфликтами. Например, каким образом должен реагировать браузер, если в импортированной таблице стилей документа указано, что элементы `h1` должны иметь красный цвет, а во встроенной таблице стилей имеется правило, которое окрашивает элементы `h1` фиолетовым цветом? Два правила стиля с селекторами `h1` имеют одинаковый вес, не так ли?

КАСКАД

Понятие «каскад» относится к той ситуации, когда за право управлять стилем элементов на странице соперничают несколько источников информации о стиле.

КАСКАД

Понятие «какад» относится к той ситуации, когда за право управлять стилем элементов на странице соперничают несколько источников информации о стиле.

Разработчики, создавшие спецификацию таблиц стилей, предвидели эту проблему и разработали иерархическую систему, которая различным источникам информации о стиле присваивает различные «веса». Каскад относится к ситуации, когда несколько источников информации о стиле соперничают за право управлять элементами на странице: информация о стиле передается при этом сверху вниз («каскадом»), пока не произойдет переопределение стиля элемента с помощью правила стиля с большим весом. Вес же формируется на основе *приоритета* источника для правила стиля, *специфичности* селектора и *порядка следования правил*.

Приоритет

Если какая-либо информация о стиле не применяется к веб-странице, эта страница отображается в соответствии с внутренней таблицей стилей браузера. Мы назовем такую ситуацию отображением, заданным по умолчанию, а консорциум W3C называет это *таблицей стилей агента пользователя*. Отдельным пользователям позволено применять собственные стили (*таблицу стилей пользователя*, также называемую *таблицей стилей читателя*), которая в браузере переопределяет стили по умолчанию. Однако, если автор веб-страницы прикрепил к ней *таблицу стилей автора*, она переопределит как стили пользователя, так и агента пользователя. Во *врезке «Иерархия правил стилей»* представлен обзор порядка следования каскадов: от наивысшего приоритета до самого низкого.

ИЕРАРХИЯ ПРАВИЛ СТИЛЕЙ

Информация о стиле может исходить из разных источников, имеющих приоритет от наивысшего до самого низкого. Другими словами, элементы, находящиеся в следующем списке выше, переопределяют элементы, расположенные в списке ниже:

- любое правило стиля, помеченное как `!important` читателем (пользователем);
- таблицы стилей, составленные автором веб-страницы;
- таблицы стилей, сформированные читателем (пользователем);
- заданные по умолчанию правила стилей браузера («таблица стилей пользовательского агента»).

Единственное исключение: если пользователь определил стиль как `«important»` (важный), то такой стиль переопределит все конкурирующие стили (см. *врезку «Присваивание степени важности»*). В этом случае пользователь сохранит настройки, учитывающие, например, укрупненный размер шрифта, предусмотренный для слабовидящих пользователей.

ПРИСВАИВАНИЕ СТЕПЕНИ ВАЖНОСТИ

Если требуется, чтобы правило не переопределялось последующим конфликтующим правилом, добавьте индикатор `!important` сразу после значения свойства для этого правила, но перед точкой с запятой. Например, чтобы текст абзаца гарантированно окрашивался в синий цвет, примените следующее правило:

```
p {color: blue !important;}
```

И если даже браузер позже обнаружит в документе встроенный стиль (который переопределяет таблицу стилей для всего документа) — например, такой:

```
<p style="color: red">
```

этот абзац по-прежнему останется в синем цвете, поскольку правило, содержащее индикатор `!important` в таблице стилей автора, не может переопределиться другими стилями.

Единственный способ для отмены правила, имеющего индикатор `!important`, состоит в указании индикатора `!important` для конфликтующего правила в таблице стилей

- читателя (пользователя). Это гарантирует, что не будут отменены специальные запросы читателя — такие, как крупный шрифт или высококонтрастный текст для слабовидящих.

И далее, если таблица стилей читателя включает такое правило:

```
p {color: black;}
```

текст останется окрашенным в синий цвет, поскольку все авторские стили (даже те, которые не отмечены как `!important`) имеют приоритет над стилями читателя. Однако, если конфликтующий стиль читателя помечен как `!important` — например:

```
p {color: black !important;}
```

абзацы окрасятся в черный цвет и не будут переопределяться с помощью авторского стиля.

Имейте в виду, что индикатор `!important` не является «палочкой-выручалочкой» на все случаи. Специалисты в области веб-дизайна советуют применять его осторожно, только в случае крайней необходимости, и, конечно же, учитывать сложную ситуацию, связанную с присутствием каскадов и наследования.

Специфичность

Конфликты могут возникать и в том случае, когда элемент получает инструкции стиля из более чем одного правила. Например, имеется правило, которое применяется ко всем абзацам, и имеется другое правило, применяемое для абзаца с идентификатором «`intro`». Какое же правило использовать для такого абзаца?

Если в таблице стилей два правила вступают в конфликт, победитель определяется по типу селектора. Чем более специфичен селектор, тем больший вес получает он при переопределении конфликтующих объявлений. В приведенном примере селектор, включающий идентификатор (ID) `#intro`, более специфичен, чем селектор общего элемента (например, `p`), поэтому к абзацу «`intro`» применяется назначенное именно ему правило, переопределяющее для него правила, установленные для всех абзацев.

ПОБЕДИТЕЛЯ ОПРЕДЕЛЯЕТ ТИП СЕЛЕКТОРА

Когда два правила в одной таблице стилей конфликтуют, победитель определяется по типу селектора.

Сейчас еще рано подробно рассматривать понятие «специфичность», поскольку к этому моменту рассмотрены только два типа селекторов. Так что пока воспринимайте термин *специфичность* как понятие, согласно которому некоторые селекторы имеют больший «вес» и, следовательно, имеют приоритет над другими селекторами. Подробно мы рассмотрим специфичность в главе 12, когда вы будете знакомы с большим числом типов селекторов.

Порядок следования правил

После выполнения сортировки по приоритету всех источников таблиц стилей и даже после того как все связанные и импортированные таблицы стилей уже оказались на своих местах, в правилах с равными весами также могут возникнуть конфликты.

ПОСЛЕДНИЙ ВЫИГРЫВАЕТ!

Каскад следует правилу: «последний выигрывает». Какое бы правило ни появилось последним, оно и будет решающим.

Тогда учитывается порядок появления правил. Каскад следует правилу: «последний выигрывает». Какое бы правило ни появилось последним, оно и будет иметь решающее значение.

То есть в пределах таблицы стилей при наличии конфликтов среди правил стиля с одинаковыми весами выигрывает тот, который оказывается последним в списке. Рассмотрим, например, следующие три правила:

```
<style>
  p { color: red; }
  p { color: blue; }
  p { color: green; }
</style>
```

В этом случае текст абзаца окрасится в зеленый цвет, поскольку последнее правило в таблице стилей, ближе всех находящееся к контенту документа, переопределяет предыдущие правила. Если рассмотреть процесс подробнее, абзацу сначала присваивается первый цвет — красный, затем назначается второй — синий и, наконец, задается третий цвет — зеленый, который и применяется. Аналогично все происходит, если конфликтующие стили встречаются в одном стеке объявлений:

```
<style>
  p { color: red;
      color: blue;
      color: green; }

</style>
```

В результате цвет абзаца будет зеленым, поскольку последнее объявление переопределяет два предыдущих. Манипулируя с составными свойствами, можно случайно переопределить в правиле предыдущие объявления, поэтому важно помнить об этом обстоятельстве. Но приведенный пример весьма прост. А что случится, если к документу будут применены правила таблиц стилей из разных источников?

Рассмотрим HTML-документ со встроенной таблицей стилей (добавлена с помощью элемента `style`), которая начинается с правила `@import` для импорта внешнего файла `*.css`. В этом HTML-документе также имеется несколько встроенных атрибутов `style`, применяемых к конкретным элементам `h1`.

Таблица стилей (external.css):

```
...
h1 { color: red }
...
```

HTML-документ:

```
<!DOCTYPE html>
<html>
  <head>
```

```
<title>...</title>
<style>
@import url(external.css); /* set to red first */
h1 { color: purple;} /* overridden by purple */
</style>
</head>
<body>
<h1 style="color: blue">Heading</h1> /* blue comes last and wins */
...
</body>
</html>
```

Когда браузер анализирует файл, он сначала попадает в импортированную таблицу стилей, которая устанавливает для элементов `h1` красный цвет. Затем браузер обнаруживает правило с равным весом во встроенной таблице стилей, которое определяет импортированное правило, поэтому элементы `h1` становятся фиолетовыми (`purple`). Затем браузер замечает правило стиля прямо в элементе `h1`, которое устанавливает цвет элемента как синий. Поскольку такое правило встретилось последним, оно и «побеждает», и этот элемент `h1` окрасится в синий цвет. Именно этот эффект наблюдался в *упражнении 11.3*. Обратите внимание, что другие элементы `h1` в этом документе, не имеющие правил встроенного стиля, окрасятся фиолетовым цветом, поскольку последний заданный цвет для всех `h1`, применяется ко всему документу.

ПРИМЕНЕНИЕ ПОРЯДКА СЛЕДОВАНИЯ ПРАВИЛ ДЛЯ ЗАПАСНЫХ ВАРИАНТОВ

Многие свойства CSS уже давно проверены и поддерживаются всеми браузерами, однако все время появляются новые полезные свойства, которые браузерам еще предстоит реализовать. Обычно новую функцию в начале поддерживают один или два браузера, а прочие отстают или вообще ее не поддерживают. Кроме того, все еще находятся в употреблении и некоторые устаревшие браузеры, что также следует учитывать веб-разработчикам.

К счастью, имеется несколько способов поддержки запасных (резервных) вариантов (альтернативных стилей), применяющих лучшие из поддерживаемых свойств) для браузеров, которые новые свойства не поддерживают. Самый простой метод основан на особенности встроенного в браузер поведения, которое состоит в игнорировании любого объявления, которое браузер не воспринимает, и срабатывании вместо подобного объявления установленного по умолчанию порядка правил.

В рассматриваемом далее примере я добавила в элемент декоративное изображение границы, применив свойство `border-image`, и предоставила запасную сплошную границу с помощью испытанного свойства `border`. Поддерживающие новое свойство браузеры применяют для изображения последнее правило в стеке. Не поддерживающие его браузеры отобразят сплошную границу, но не выдадут сообщение об ошибке, обнаружив свойство `border-image`, которое для них непонятно, а просто проигнорируют такое свойство. В этих браузерах граница отображается как запасной вариант в виде сплошной красной линии, и это нормально, а пользователи с поддерживающими браузерами увидят декоративную границу, как и предполагалось:

```
▶ h1 {
    /* fallback first */
    border: 25px solid #eee
    /* newer technique */
    border-image: url(fancyframe.png) 55 fill / 55px / 25px;
}
```

Вы заметите, что в этой книге я всегда следую такому методу предоставления запасных вариантов, помещая новые свойства последними.

Блочная модель

Поскольку речь идет о важных понятиях CSS, уместно представить краеугольный камень системы визуального форматирования CSS: *блочную модель* (box model). Самый простой вариант представления о блочной модели состоит в следующем: браузеры видят каждый элемент на странице — как блочные (block element), так и строчные — как заключенные в маленькие прямоугольные блоки. К этим блокам можно применять свойства — такие, как границы, поля, отступы и фоны, и даже перемещать их по странице¹.

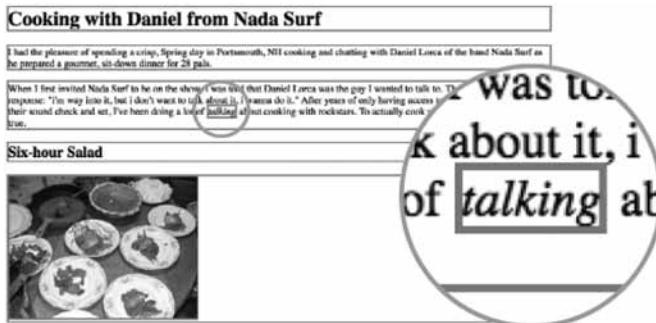
Блочная модель будет подробно рассмотрена в главе 14, но общее представление о ней можно составить, обсуждая текст и фоны в последующих двух главах.

Для того чтобы увидеть элементы примерно так, как их видит браузер, я создала правила стиля, добавляющие границы вокруг каждого элемента из рассматриваемого в этой главе примера:

```
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; }
```

На рис. 11.9 показано, что у меня получилось в результате применения этих правил стиля. Границы демонстрируют форму каждого блока блочных элементов страницы. Вокруг строчных элементов (*em* и *img*) границы также имеются. Если обратить внимание на заголовки, можно заметить, что блоки, окружающие блочные элементы, расширяются, заполняя доступную им ширину окна браузера, что и соответствует природе блочных элементов в обычном потоке документа. а вот строчные блоки охватывают только символы или изображения, которые они содержат.

¹ Следует различать понятия «блочная модель» (от англ. box model) и «блочный элемент» (от англ. block element). Как видите, в английском языке понятие «блок» можно передать как словом «box» (бокс, коробка, ящик), так и словом «block» (блок, кубик). К сожалению, в русском языке приходится оба понятия передавать словом «блок», что может вносить некоторую путаницу. Остается надеяться, что наши читателям поймут разницу между этими понятиями по контексту (Прим. ред.).



Daniel prepared a salad of arugula, smoked tomatoes, tomato jam, and grilled avocado (it's as good as it sounds!). I jokingly called it "6-hour Salad" because that's how long he worked on it. The fresh tomatoes were slowly smoked over woodchips in the grill, and when they were softened, Daniel separated out the seeds which he reduced into a smoky jam. The tomatoes were cut into strips to put on the salad. As the day ended, the avocado finally went on the grill after dark. I was on flashlight duty while Daniel checked for the perfect grill marks.

I wrote up a streamlined adaptation of his recipe that requires ~~much~~ less time and serves 6 people instead of ~~gobs~~ that amount.

The Main Course

In addition to the smoky grilled salad, Daniel served tarragon cornish hen with a cognac cream sauce loaded with chives and grapes, and wild rice with grilled ramps (wild garlicy leeks). Dinner was served close to midnight, but it was a party so nobody cared.

We left that night (technically, early the next morning) with full bellies, new cooking tips, and nearly 5 hours of footage. I'm considering renaming the show "Cooking with Nada Surf".

Рис. 11.9. Приведенные в этом разделе правила стилей, будучи примененными ко всем элементам, выявляют блоки, охватывающие как блочные, так и строчные элементы

КРАТКАЯ ИСТОРИЯ CSS

Первая версия таблицы стилей CSS (**рекомендация CSS уровня 1**, так называемая CSS1) выпущена в 1996-м году, и в нее были включены свойства для добавления шрифта, цвета и инструкций по формированию интервалов между элементами страницы. К сожалению, отсутствие поддержки со стороны браузеров несколько лет мешало широкому распространению CSS.

Таблица стилей CSS уровня 2 (CSS2), выпущенная в 1998-м году, содержала дополнительные свойства по позиционированию, которые позволили применять CSS для макетирования страниц. Также были добавлены стили для других типов носителей (например, для печати и портативных устройств) и более сложные методы выбора элементов. **Таблицы стилей CSS уровня 2, версия 1** (CSS2.1) содержали некоторые незначительные изменения, внесенные в CSS2, и получили статус рекомендации в 2011-м году.

Таблицы стилей уровня 3 (CSS3) отличаются от предыдущих версий разделением на отдельные модули, содержащие такие функции, как анимация, многоколоночные макеты или границы. Некоторые модули были стандартизированы, другие остались экспериментальными. Это дало возможность разработчикам браузеров приступать к реализации (и использованию!) ряда функций, не ожидая подготовки всей спецификации.

В настоящее время каждый модуль CSS выделен в отдельный уровень, и модули имеют собственные номера уровней. Нет больше гигантских, включающих в себя все возможные свойства, версий CSS. Вновь добавленные модули — такие, как модуль разметки сетки, начинаются с уровня 1. Модули, имеющие определенную предысторию, достигли уровня 4.

А какое разнообразие индивидуальных спецификаций находится в разработке! Для обзора спецификаций и их различных состояний посетите текущую рабочую страницу CSS W3C, доступную по адресу: www.w3.org/Style/CSS/current-work.

Сгруппированные селекторы

А вот и прекрасная возможность показать вам удобный прием применения правил стиля. Если нужно применить одно и то же свойство стиля к ряду элементов, селекторы можно сгруппировать в одно правило, разделяя их запятыми. Следующее

ВОПРОС НА ЗАСЫПКУ

Можете ли вы догадаться, почему я просто не добавила свойство `border` к элементу `body`, чтобы его наследовали все элементы в сгруппированном селекторе?

Ответ:

Можно, но это будет не очень эффективно. Каждый элемент получит свой собственный `border`.

правило оказывает тот же эффект, что и пять правил, приведенных в предыдущем разделе. Группирование селекторов позволяет эффективно вносить изменения в сайты и уменьшает размер файла:

```
h1, h2, p, em, img { border: 1px solid blue; }
```

Теперь в вашем наборе инструментов имеются два типа селекторов: простой селектор простого элемента и сгруппированные селекторы.

ЕДИНИЦЫ ИЗМЕРЕНИЯ CSS

В этой главе закладываются основы для дальнейшего изучения таблиц стилей, поэтому сейчас желательно ознакомиться с единицами измерения, используемыми в них. Эти единицы применяются для установки размера шрифта, ширины и высоты элементов, полей, отступов и т. д. Полный их список представлен во врезке «Единицы измерения CSS».

Некоторые из них вам знакомы (например, дюймы и миллиметры), но имеются единицы измерения, которые нуждаются в дополнительных объяснениях: абсолютные единицы, `rem`, `em` и `vw/vh`. Представление об эффективном применении элементов CSS служит одним из основных навыков при работе с таблицами стилей.

ЕДИНИЦЫ ИЗМЕРЕНИЯ CSS

В CSS3 представлены различные единицы измерения. Они делятся на две большие категории: *абсолютные* и *относительные*.

Абсолютные единицы

Абсолютные единицы имеют предопределенные значения или эквиваленты из реального мира. За исключением пикселов (`px`), они не подходят для веб-страниц, которые отображаются на экранах. Итак:

- `px` — пиксель, определенный в CSS3 как равный $1/96$ дюйма;
- `in` — дюймы;
- `mm` — миллиметры;
- `cm` — сантиметры;

- ▶ • **q** — $\frac{1}{4}$ миллиметра;
- **pt** — пункты ($\frac{1}{72}$ дюйма). Пункт — это единица, обычно применяемая в полиграфическом дизайне;
- **pc** — пики (1 пика = 12 пунктов или $\frac{1}{6}$ дюйма). Пика — это единица, обычно применяемая в полиграфическом дизайне.

Относительные единицы

Относительные единицы измерения основаны на сравнении с размером другого объекта — такого, как заданный по умолчанию размер текста или размер родительского элемента:

- **em** — единица измерения, равная текущему размеру шрифта;
- **ex** — x-высота, приблизительно равная высоте строчной буквы «х» в шрифте;
- **rem** — root em, равный размеру корневого элемента `em` HTML;
- **ch** — нулевая ширина, равная ширине нуля (0) в текущем шрифте и размере;
- **vw** — единица ширины окна просмотра, равная $\frac{1}{100}$ ширины текущего окна просмотра (окна браузера);
- **vh** — единица высоты окна просмотра, равная $\frac{1}{100}$ текущей высоты окна просмотра;
- **vmin** — минимальная единица просмотра, равная значению `vw` или `vh`, в зависимости от того, что из них меньше;
- **vmax** — максимальная единица просмотра, равная значению `vw` или `vh`, в зависимости от того, что из них больше.

ПРИМЕЧАНИЯ

- Хотя трудно полагать, что это «единица измерения», однако проценты являются еще одним распространенной единицей измерения для элементов веб-страницы. Проценты рассчитываются относительно другого значения — такого, как значение свойства, примененного к текущему элементу, к родительскому элементу или предку. Спецификация всегда указывает, относительно чего именно рассчитывается процентное значение для свойства. Процентные значения для макетов страниц обеспечивают пропорциональность элементов страницы.
- Дочерние элементы не наследуют относительные значения своего родителя, а применяют, скорее, результирующее вычисленное значение.
- IE9 поддерживает `vm` вместо `vmin`. IE и Edge (все версии по состоянию на 2017-й год) не поддерживают `vmax`.

Абсолютные единицы измерения

Абсолютные единицы имеют предопределенные значения или эквиваленты из реального мира. Они всегда задают один и тот же размер, независимо от контекста, в котором используются.

Наиболее популярная абсолютная единица для веб-дизайна — пиксель, который CSS3 определяет как $\frac{1}{96}$ дюйма. Пиксели чувствуют себя на пиксельном экране как дома и обеспечивают на веб-странице точное задание размеров текста и элементов. Некоторое время пиксели широко использовались. Однако затем стало ясно, что они слишком «жесткие» для страниц, которые должны адаптироваться к различным

размерам экрана и пользовательским предпочтениям. Относительные единицы — такие, как `rem`, `em` и `%` — больше соответствуют изменчивому поведению страниц в этих средах.

Использовать пиксели имеет смысл, пожалуй, только для задания размеров границ окон, а вот все прочие абсолютные единицы — такие, как `pt`, `pc`, `in`, `mm` и `cm` — мало пригодны для задания экранных размеров элементов, хотя они могут быть полезны для таблиц стилей правил печати. Это немного сужает ваш выбор.

Тем не менее пиксели занимают свою нишу в веб-дизайне для элементов, которые должны иметь одинаковый размер независимо от контекста. Ширина границ изображений, размеры которых заданы в пикселях, также определяется в пикселях.

Относительные единицы измерения

Как было отмечено ранее, для задания большинства размеров в Интернете применяются, как правило, относительные единицы, при этом доступно несколько их вариантов: `rem`, `em` и `vw/vh`.

Единица измерения `rem`

Относительная единица измерения, называемая `rem` (сокращение от `root em`) появилась в таблицах стилей CSS3. Она основана на размере шрифта корневого элемента HTML. В современных браузерах размер корневого шрифта по умолчанию составляет 16 пикселов — следовательно, `rem` эквивалентен единице измерения в 16 пикселов (если явно не введено другое значение). Элемент размером в 10 `rem` имеет размер, равный 160 пикселов.

Для большинства случаев в правилах стиля можно применять единицы измерения `rem` как абсолютные единицы измерения, но поскольку они все-таки относительны, при изменении размера основного шрифта то же происходит и с единицей измерения `rem`. Когда для облегчения чтения с расстояния пользователь изменяет базовый размер шрифта до значения, равного 24 пикселя, или при отображении страницы на устройстве, размер шрифта которого по умолчанию составляет 24 пикселя, элемент размером 10 `rem` занимает 240 пикселов. Все это кажется достаточно сложным, но эта особенность очень важна, особенно в случаях, когда требуется, чтобы элемент макета расширялся при увеличении размера текста. Тогда страница отображается пропорционально размеру шрифта, что помогает поддерживать оптимальную длину строки.

ЗАПАСНЫЕ ВАРИАНТЫ ДЛЯ УСТАРЕВШИХ БРАУЗЕРОВ IE

Недостаток единиц `rem` заключается в том, что браузер IE8 и более ранние его версии их не поддерживают, и необходимо предоставить запасное объявление с эквивалентным измерением в пикселях. Впрочем, существуют инструменты разработки, которые автоматически конвертируют все `rem`-единицы в пиксели, — речь о них пойдет в главе 20.

Единица измерения em

Единица *em* — это относительная единица измерения, которая для традиционной типографии исходит из ширины заглавной буквы M (отсюда и название «ем»). В спецификации CSS единица измерения *em* вычисляется как расстояние между базовыми линиями, когда шрифт установлен без дополнительного пространства между строками (такой шрифт называют *лидингом*). Для текста, размер шрифта которого равен 16 пикселов, единица *em* и равна 16 пикселов, для 12-пиксельного текста *em* равна 12 пикселов и так далее (рис. 11.10).

EM ИЛИ EM?

*Не следует путать единицу измерения *em* с HTML-элементом *em*, используемым для обозначения выделенного текста. Это совершенно разные объекты.*

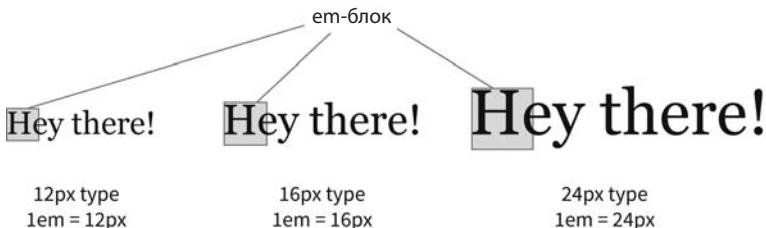


Рис. 11.10. Единица *em* основана на размере текста

Когда размер *em* для текстового элемента браузером вычислен, его можно использовать для всех других измерений — таких, как отступы, поля, ширина элемента на странице и т. д. Базовые измерения текста помогают сохранить пропорции при изменении его размеров.

Сложность при работе с единицами измерения *em* заключается в необходимости помнить о том, что эти единицы измерения всегда соответствуют текущему размеру шрифта элемента. Согласно примеру из книги Эрика Мейера (Eric Meyer) и Эстель Вейль (Estelle Weyl) «CSS: The Definitive Guide» (издательство O'Reilly) — при выборе для левого поля элементов *h1*, *h2* и *p*-значения, равного *2em*, эти элементы не будут корректно выстраиваться в столбец, поскольку единицы измерения *em* основаны на соответствующих размерах шрифта элементов (рис. 11.11).

```
h1, h2, p { margin-left: 2em; }
```

This screenshot shows a web page with the following CSS rule applied to the *h1*, *h2*, and *p* elements:

```
h1, h2, p { margin-left: 2em; }
```

The page contains two headings and one paragraph:

- This is a 24pt Heading**: This heading has a *font-size* of 24 pixels. Its *margin-left* is 2em, which is equivalent to 48 pixels (24px * 2).
- A Heading in 20pt**: This heading has a *font-size* of 20 pixels. Its *margin-left* is 2em, which is equivalent to 40 pixels (20px * 2).
- Text**: This paragraph has a *font-size* of 16 pixels. Its *margin-left* is 2em, which is equivalent to 32 pixels (16px * 2). The text overlaps with the second heading because its margin is larger than its font size.

Рис. 11.11. Единицы измерения *em* всегда соответствуют размеру шрифта элемента. Единица измерения *em* для одного элемента может не совпадать с аналогичной величиной для другого элемента

Размеры области просмотра, выраженные в процентах (vw/vh)

Единицы для ширины области просмотра (*vw*) и высоты области просмотра (*vh*) являются относительными для размеров области просмотра (окна браузера). Единица измерения *vw* равна $\frac{1}{100}$ ширины области просмотра, а единица измерения *vh* равна $\frac{1}{100}$ ее высоты. Единицы на основе области просмотра позволяют изображениям и текстовым элементам занимать всю ширину или высоту области просмотра:

```
header {  
    width: 100vw;  
    height: 100vh; }
```

Следующий код также присваивает единице измерения процентное соотношение, определяющее размер окна в 50% от возможного:

```
img {  
    width: 50vw;  
    height: 50vh; }
```

Относительными являются и единицы измерения *vmin* (равные значению *vw* или *vh* — в зависимости от того, что является меньшим) и *vmax* (равные значению *vw* или *vh* — в зависимости от того, что является большим).

Материал этого раздела должен сформировать у вас представление о единицах измерений, которые применяются в таблицах стилей. Я рекомендую вам также ознакомиться с документом «CSS Values and Units Module» (www.w3.org/TR/css3-values/) для углубления познаний и упрощения понимания значений свойств, представленных

ПОДДЕРЖКА БРАУЗЕРАМИ

*Браузер IE9 поддерживает единицы *vmt* вместо *vmin*. Браузеры IE and Edge (все версии вплоть до версии от 2017-го года) не поддерживают *vmax*.*

в моей книге. В дополнение к единицам измерений этот документ включает текстовые значения (такие, как ключевые слова, текстовые строки и URL-адреса), числа и процентные значения, цвета и многое другое.

ИНСТРУМЕНТЫ ВАШЕГО БРАУЗЕРА ДЛЯ ВЕБ-ДИЗАЙНА

Вследствие свойства каскадности к одному элементу страницы можно применять стили из нескольких источников. Это усложняет отладку страницы, если стили не отображаются так, как вы планируете. К счастью, каждый серьезный браузер располагает инструментами разработчика, которые помогут вам в работе.

Предположим, вы открываете в браузере Chrome простой документ `cooking.html`, над которым в этой главе уже работали, а затем в меню выбираете команды: **View | Developer | Developer Tools** (Вид | Разработчик | Инструменты разработчика). В нижней части документа откроется панель инструментов разработчика (рис. 11.12). Впрочем, можно создать для этой панели отдельное окно, щелкнув на значке окна в ее левом верхнем углу.



Рис. 11.12. Браузер Chrome с открытой панелью **Developer Tools**

На вкладке **Elements** (Элементы), показанной слева, отображается исходный HTML-код документа. Контент изначально скрыт, поэтому более четко видна структура документа, но при нажатии на имеющиеся слева от каждой строки стрелки ее раздел открывается полностью. При щелчке на элементе в исходном коде (например, на показанном на рис. 11.12 втором элементе `p`), этот элемент также выделяется и в окне браузера.

На вкладке **Styles** (Стили), показанной справа, отображаются все стили, которые применены к выбранному элементу. В приведенном примере в документе показаны свойства `font-size`, `font-family` и `margin-left` элемента `style`. При наличии внешних CSS-документов они также указываются в списке. Отображается в нем и **User Agent Style Sheet** (таблица стилей агента пользователя), которая служит стилем браузера, выбранным по умолчанию. В нашем случае таблица стилей браузера добавляет пространство вокруг абзаца. Браузер Chrome также поддерживает для выбранного элемента визуальное представление блочной модели, которая показывает применяемые размеры контента, отступы, границы и поля. Это отличный инструмент для устранения в макетах непредвиденных пустот.

И действительно круто, что при редактировании правил стиля на панели инструментов изменения отражаются в браузере в режиме просмотра страницы и в реальном времени! Стоит выбрать элемент `h1` и изменить окраску с оранжевого цвета на зеленый, как этот элемент станет зеленым и в окне. Это отличный способ поэкспериментировать или устраниить неполадки в дизайне, причем реальные изменения в документ при этом не вносятся. Это лишь предварительный просмотр, поэтому вам потом придется продублировать изменения в исходном коде.

Таким образом, можно просматривать любую страницу в Сети, экспериментировать с подключением и отключением стилей и даже добавлять собственные стили. Ваши действия не окажут влияния на реальный сайт — это занятие больше подходит для образовательных и развлекательных целей.

Инспекторы элементов и стилей — это только вершина айсберга из всего, что могут делать инструменты разработчика браузера. Можно также настраивать и отлаживать код JavaScript, проверять уровень производительности, просматривать документ в различных имитационных устройствах и выполнять другие действия. Приятно отметить, что все основные браузеры теперь имеют встроенные инструменты с похожими функциями. Для веб-разработчика они являются лучшими друзьями:

- Chrome DevTools: Вид | Разработчик | Инструменты разработчика (developer.chrome.com/devtools);
- Firefox: Инструменты | Веб-разработка (developer.mozilla.org/en-US/docs/Tools);
- Microsoft Edge: открывается с помощью клавиши <F12> (developer.microsoft.com/en-us/microsoft-edge/platform/documentation/f12-devtools-guide/);
- Safari: Разработчик | Показать веб-инспектор (developer.apple.com/safari/tools/);
- Opera: Вид | Инструменты разработчика | Opera Dragonfly (www.opera.com/dragonfly/);
- Internet Explorer 9+: открывается с помощью клавиши <F12> ([msdn.microsoft.com/en-us/library/gg589512\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/gg589512(v=vs.85).aspx)).

ВПЕРЕД — ВМЕСТЕ С CSS

В этой главе рассмотрены основные понятия, связанные с каскадными таблицами стилей, включая синтаксис правил, способы применения стилей к документу, а также основные понятия наследования, каскада (включая приоритет, специфичность и порядок правил) и блочную модель. Таблицы стилей больше не представляют для вас загадочный объект, а далее мы будем опираться на созданный здесь фундамент, добавляя в свой арсенал свойства и селекторы и расширяя представленные здесь концепции.

Таблицы стилей CSS — это обширная тема, выходящая далеко за рамки книги. Книжные магазины и Интернет просто переполнены информацией о таблицах стилей для пользователей различных уровней квалификации. Я подобрала для вас список ресурсов, которые считаю наиболее полезными в процессе обучения. Также предоставляется список популярных инструментов, оказывающих неоценимую помощь при написании таблиц стилей.

Книги

Написано весьма много хороших книг, посвященных CSS, и здесь предлагаются те из них, которые я могу порекомендовать, предварительно ознакомившись с их содержанием:

- «CSS: The Definitive Guide», 4-е издание, авторы: Эрик А. Мейер (Eric A. Meyer) и Эстель Вейль (Estelle Weyl) (издательство О'Рейли);
- «CSS Cookbook», автор Кристофер Шмитт (Christopher Schmitt) (издательство О'Рейли).

Ресурсы в Интернете

Приведенные здесь сайты являются хорошей отправной точкой для изучения таблиц стилей:

- CSS-Tricks (css-tricks.com) — этот блог ведет гуру в области CSS Крис Койер (Chris Coyier). Крис любит CSS и с энтузиазмом делится на этом сайте своими исследованиями и разработками;
- Консорциум World Wide Web (www.w3.org/TR/CSS/) — консорциум World Wide Web курирует разработку веб-технологий, в том числе и CSS. Эта страница представляет собой как бы «снимок» CSS-спецификации. Смотрите также информацию на сайте: www.w3.org/Style/CSS/current-work;
- MDN Web Docs (developer.mozilla.org) — страницы CSS в MDN включают подробные справочные страницы, пошаговые руководства и демонстрации и служат отличным центром для исследования веб-технологий;
- A List Apart (www.alistapart.com/topics/code/css/) — в этом онлайн-журнале представлены лучшие идеи и статьи о новинках опирающегося на стандарты веб-дизайна. Журнал основан в 1998-м году Джейффи Зельдманом (Jeffrey Zeldman) и Брайаном Платцем (Brian Platz).

КОНТРОЛЬНЫЕ ВОПРОСЫ

Далее предлагаются несколько вопросов, которые помогут вам проверить свои знания по основам CSS. Ответы на эти вопросы приведены в *приложении 1*.

1. Определите различные части этого правила стиля:

```
blockquote { line-height: 1.5; }
```

селектор: _____ значение: _____

свойство: _____ объявление: _____

2. Какого цвета будут абзацы после применения к документу следующей встроенной таблицы стилей? Почему?

```
<style type="text/css">
  p { color: purple; }
  p { color: green; }
  p { color: gray; }
</style>
```

3. Перепишите каждый из этих CSS-примеров. Некоторые из них совершенно неверны, а некоторые могут быть написаны значительно эффективнее.

a.

```
p {font-family: sans-serif;}
  p {font-size: 1em;}
  p {line-height: 1.2em;}
```

```
b. blockquote {  
    font-size: 1em  
    line-height: 150%  
    color: gray }  
  
c. body  
    {background-color: black; }  
    {color: #666; }  
    {margin-left: 12em; }  
    {margin-right: 12em; }  
  
d. p {color: white; }  
    blockquote {color: white; }  
    li {color: white; }  
  
e. <strong style="red">Act now!</strong>
```

4. Обведите все элементы, которые должны отображаться красным, когда к документу со структурой, показанной на рис. 11.13, применено следующее правило стиля:

```
div#intro { color: red; }
```

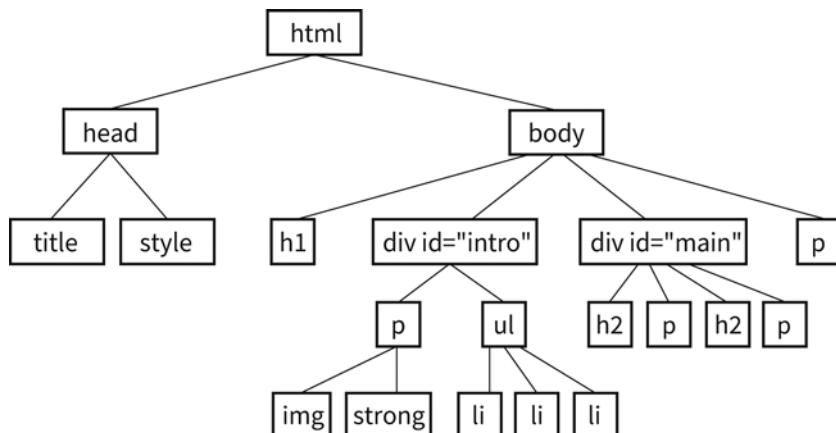


Рис. 11.13. Структура документа для вопроса 4

ГЛАВА 12

ФОРМАТИРОВАНИЕ ТЕКСТА

В этой главе...

- ▶ Свойства шрифтов
- ▶ Веб-шрифты
- ▶ Расширенная CSS3-тиографика
- ▶ Настройки для текстовых строк
- ▶ Текстовые эффекты
- ▶ Селекторы потомков, селекторы ID и класса
- ▶ Обзор специфиности
- ▶ Список стилей

Наверное, получив предварительно отформатированный текст, не мешало бы привести его в божеский вид? И это не так уж и сложно. К концу этой главы вы ознакомитесь с более чем сорока дополнительными CSS-свойствами, которые служат для управления внешним видом текста. Попутно вы узнаете, как использовать более мощные селекторы для выделения элементов в определенном контексте и с определенным идентификатором (`id`) или именем (`class`).

Сама природа Интернета усложняет назначение форматирования, а иногда и приводит к разочарованию, особенно, если у вас уже имеется опыт подготовки текстов для печати или форматирования текста в текстовых редакторах. Невозможно точно определить, окажется ли доступным назначенный вами шрифт в браузерах ваших пользователей, а также задать для этого шрифта подходящий размер. Но не все так плохо, и по мере продвижения вперед мы рассмотрим лучшие практические подходы, обеспечивающие решение этих проблем.

В процессе работы над этой главой мы создадим на базе меню сайта «Black Goose Bistro», размеченном нами в главе 5, более продвинутый вариант онлайн-меню этого сайта. Я рекомендую вам выполнять приведенные в этой главе упражнения, чтобы глубже понять, как работают рассматриваемые здесь свойства. На рис. ЦВ-12.1 показано наше меню до начала его обработки (*сзади*) и по ее завершении (*впереди*). Это далеко не шедевр, поскольку здесь только немного задействованы возможности применения CSS, но текст, по крайней мере, уже имеет характерную индивидуальность.

ОСНОВНЫЕ СВОЙСТВА ШРИФТОВ

Когда я разрабатываю текстовый документ (для печати или для размещения в Сети), первое, что я делаю, это задаю шрифт. В CSS шрифты указываются с использованием набора связанных с гарнитурой шрифта свойств, а также его размера, жирности, стиля и специальных символов. Существуют также сокращенные свойства, которые позволяют в одном правиле указывать несколько атрибутов шрифта.

ОБРАТИМСЯ К СПИСКАМ СВОЙСТВ

Каждый список свойств CSS в книге снабжен дополнительной информацией об особенностях этих свойств и методике их применения. Списки свойств содержат следующие позиции:

- **Значения** — приведены допустимые для свойства значения. Предопределенные значения ключевых слов отображаются с помощью соответствующего шрифта (например: `small`, `italic` или `small-caps`) и должны вводиться, как показано.
- **По умолчанию** — значение, которое применяется для свойства, заданного по умолчанию (его начальное значение), то есть применяемое в случае, когда не указано другое значение. Обратите внимание, что заданные по умолчанию значения таблицы стилей браузера могут отличаться от заданных по умолчанию, но определенных в CSS значений.
- **Применение** — некоторые свойства применяются только к элементам определенных типов.
- **Наследование** — указывает, передается ли свойство потомкам элемента.

Ключевые слова CSS

Все свойства CSS воспринимают три ключевых слова в формате CSS: `initial`, `inherit` и `unset`. Поскольку они являются общими для всех свойств, они не указываются среди значений в отдельных списках свойств:

- ключевое слово `initial` явно устанавливает для свойства (начальное) значение, заданное по умолчанию;
- ключевое слово `inherit` позволяет явно заставить элемент наследовать свойство стиля от его родителя. Это может пригодиться при переопределении примененных к этому элементу других стилей и гарантирует, что элемент всегда соответствует своему родителю;
- и, наконец, ключевое слово `unset` устраниет объявленные значения, встречающиеся ранее в каскаде, и присваивает свойству либо значение `inherit`, либо значение `initial` — в зависимости от того, наследуется свойство или нет.

Указание названия шрифта

Выбор гарнитуры шрифта, или *семейства шрифтов* (`font family`), как это называется в CSS, служит хорошей отправной точкой для начала работы. Так что со свойства `font-family` и его значений мы и начнем:

`font-family`

- **Значения:** один или несколько шрифтов или названия общих семейств шрифтов, разделенные запятыми.

- **По умолчанию** — зависит от браузера.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Применяйте свойство `font-family` для указания названия шрифта или списка шрифтов (называемого также *стеком шрифтов*), как показано в следующих примерах:

```
body { font-family: Arial; }
var { font-family: Courier,
      monospace; }
p { font-family: "Duru Sans",
      Verdana, sans-serif; }
```

Существуют также некоторые важные синтаксические требования:

- все названия шрифтов, за исключением общих семейств шрифтов, должны быть написаны с заглавной буквы. Например, пишите `Arial`, а не `arial`;
- используйте запятые для разделения нескольких названий шрифтов, как показано во втором и третьем примерах;
- обратите внимание, что имена шрифтов, содержащие пробелы (например, `Duru Sans` в третьем примере), должны заключаться в кавычки.

Может возникнуть вопрос: «Зачем указывать более одного шрифта?». Это — хороший вопрос, и он подводит нас к задаче определения веб-шрифтов.

Ограничения, налагаемые на шрифты

Браузеры способны отображать только те шрифты, к которым у них есть доступ. Речь при этом идет о шрифтах, уже установленных на жестком диске компьютера пользователя. Однако в 2010-м году для браузеров была расширена поддержка встроенных веб-шрифтов на основе использования правила CSS `@font-face`, поэтому разработчики получили возможность размещать на веб-страницах и другие шрифты (см. врезку «*Знакомство с веб-шрифтами*»).

Но вернемся к нашему правилу `font-family`. Даже если указать в правиле стиля, что должен применяться шрифт `Futura`, но браузер не сможет его найти (например, этот шрифт не установлен на компьютере пользователя или не удается загрузить

ЕЩЕ О СВОЙСТВАХ ШРИФТОВ

Свойства шрифтов, относящиеся к CSS2.1 и полностью поддерживаемые:

- `font-family`
- `font-size`
- `font-weight`
- `font-style`
- `font-variant`
- `font`

Модуль шрифтов CSS уровня 3 добавляет следующие свойства для более сложной обработки шрифтов. Учтите, что на момент подготовки книги эти свойства поддерживались не всеми браузерами:

- `font-stretch`
- `font-variant-ligatures`
- `font-variant-position`
- `font-variant-caps`
- `font-variant-numeric`
- `font-variant-alternates`
- `font-variant-east-asian`
- `font-size-adjust`
- `font-kerning`
- `font-feature-settings`
- `font-language-override`

соответствующий веб-шрифт), то вместо него будет применен заданный по умолчанию шрифт браузера.

К счастью, CSS поддерживает список резервных шрифтов (стек шрифтов, о котором речь шла ранее), к которому браузер обращается в том случае, если недоступен заданный шрифт. Если первый из указанных шрифтов не найден, браузер пытается найти следующий, и далее по списку, пока не найдет доступный шрифт. В третьем правиле `font-family`, показанном в предыдущем примере кода, если браузер не находит шрифт `Duru Sans`, будет применяться шрифт `Verdana`, а если `Verdana` недоступен, его заменит какой-либо другой шрифт без засечек.

ЗНАКОМСТВО С ВЕБ-ШРИФТАМИ

Возможность предоставлять свой собственный шрифт для использования на веб-странице доступна, начиная с 1998-го года, но реально не осуществлялась из-за того, что браузеры не были к этому готовы. К счастью, это положение изменилось, и теперь веб-шрифты вполне доступны. Благодаря этому Интернет выглядит все краше!

О веб-шрифтах можно много рассказать, и эта врезка служит лишь введением, которое включает следующие темы.

Форматы веб-шрифтов

Изначально существовали два основных препятствия для использования шрифтов на веб-страницах. Первая проблема заключается в том, что разные браузеры поддерживают различные форматы шрифтов. Большинство шрифтов имеют формат OpenType (OTF) или TrueType (TTF), но устаревшие версии Internet Explorer допускают только собственный встроенный шрифт (Embedded Open Type, EOT).

Однако уже существует новый стандарт упаковки шрифтов, применяемых на веб-страницах, причем этот формат внедряют сейчас все поставщики браузеров, даже IE. Новый формат WOFF/WOFF2 (Web Open Font Format версий 1 и 2) представляет собой контейнер для упаковки файлов шрифтов при их веб-доставке. Так что теперь браузер IE9 поддерживает WOFF и, в целом, это как раз то, что нам нужно. Однако до сих пор по-прежнему рекомендуется предоставлять для веб-страниц один и тот же шрифт в нескольких различных форматах (мы вернемся к этому вопросу позднее).

Другая проблема с предоставлением шрифтов для веб-страниц состоит в том, что занимающиеся разработкой шрифтов компании (назовем их *тиографами*) обеспокоены (это вежливый способ сказать «в шоке») тем, что их шрифты при публикации на серверах станут доступны для загрузки. Разработка шрифтов стоит дорого. И большинство шрифтов распространяются под защитой лицензий, которые оговаривают их весьма специфическое использование на ограниченном числе компьютеров и «бесплатную загрузку» обычно не предусматривают.

Таким образом, предоставляя ссылку на веб-шрифт, необходимо иметь в виду, что шрифт этот можно использовать на законных основаниях и что его способны поддерживать все браузеры.

Имеются два основных подхода к предоставлению шрифтов: самостоятельное размещение или обращение к сервису, выполняющему эти функции. Рассмотрим оба варианта.



► Размещение имеющегося у вас шрифта

Если вы выберете этот вариант, найдите нужный шрифт, разместите его на сервере с использованием всех необходимых форматов и свяжите его с веб-страницей, применяя правило CSS3 `@font-face`. Следует отметить, что каждый файл шрифта соответствует одному варианту жирности или начертания шрифта. Поэтому, если желательно применить обычную, полужирную и курсивную его версии, нужно разместить три разных файла шрифта и ссылаться в CSS на каждый из них отдельно.

- **Шаг 1. Найдите шрифт.** Это может стать сложной задачей, поскольку лицензионное соглашение с конечным пользователем (End User License Agreement, EULA), имеющее место практически для всех коммерческих шрифтов, не распространяется на их использование в Сети. Обязательно приобретите дополнительную лицензию на использование выбранного шрифта в Интернете, если она доступна. Тем не менее, учитывая возрастающую потребность, некоторые типографы предоставляют шрифты для использования в Интернете, а также растет число шрифтов с открытым исходным кодом, которые можно применять бесплатно. Сервис Fontspring (fontspring.com), созданный Итаном Данхэмом (Ethan Dunham), — удобное место для приобретения имеющих лицензию на публикацию в Интернете шрифтов, которые можно использовать на сайте или на своем компьютере. Сайт Font Squirrel ([fontsquirrel.com](http://www.fontsquirrel.com/fontface/generator)), также созданный Итаном Данхэмом, служит источником шрифтов с открытым исходным кодом, которые можно бесплатно использовать в коммерческих целях.
- **Шаг 2. Сохраните шрифт в нескольких форматах.** На момент подготовки книги уже стало реальностью предоставление нескольких форматов шрифтов (EOT, WOFF, TTF, SVG). Рекомендованным источником для шрифтов различных форматов может служить поставщик, у которого шрифты приобретены, поскольку тогда и сами шрифты будут иметь высшее качество, а также не будет сомнений по части их использования в соответствии с лицензионным соглашением. Если вы выбрали шрифт с открытым исходным кодом (свободный от лицензионных ограничений) и вам необходимы альтернативные его форматы, воспользуйтесь услугами специальной службы, которая примет ваш шрифт и сделает все необходимое, — «@font-face Generator» от Font Squirrel (www.fontsquirrel.com/fontface/generator). Перейдите на указанную страницу, загрузите шрифт, и вы получите возможность скачать этот шрифт в форматах TTF, EOT, WOFF, WOFF2 и SVG, а также CSS-код, требуемый для их функционирования.
- **Шаг 3. Загрузите файлы шрифтов на сервер.** Разработчики обычно хранят файлы шрифтов в том же каталоге, что и CSS-файлы, но это лишь вопрос предпочтения. Если загружается пакет от Font Squirrel, обязательно свяжите все части вместе, как их и получили.
- **Шаг 4. Создайте код.** Свяжите шрифт с вашим сайтом, применив правило `@font-face` в файле таблицы стилей (документе `*.css`). Правило таблицы стилей дает шрифту имя семейства шрифтов, которое затем можно применять в таблице стилей. Также в нем указано местоположение файлов шрифтов в их различных форматах. Приведенный далее пример кросс-браузерного кода для адресации ошибки в браузере IE разработан Итаном Данхэмом (да-да, снова им!). Я рекомендую вам ознакомиться с полной версией этой статьи, доступной по адресу: blog.fontspring.com/2011/02/further-hardening-of-the-bulletproof-syntax/. Также просмотрите обновленную версию статьи Пауля Айриша (Paul Irish), которая доступна на сайте: paulirish.com/2009/bulletproof-font-face-implementation-syntax/.

```
► @font-face {  
    font-family: 'MyWebFont';  
    src: url('webfont.eot'); /* IE9 Compat Modes */  
    src: url('webfont.eot?#iefix') format('embeddedopentype'),  
        /* IE6-IE8 */  
    url('webfont.woff') format('woff'),  
        /* Modern Browsers */  
    url('webfont.ttf') format('truetype'),  
        /* Safari, Android, iOS */  
    url('webfont.svg#svgFontName') format('svg');  
        /* Legacy iOS */  
}
```

Затем просто в правилах шрифта сошлитесь на название установленного шрифта — например, так:

```
p {font-family: MyWebFont; }
```

Обращение в сервис встраивания шрифтов

Если все здесь описанное покажется вам слишком сложной процедурой, вы можете зарегистрироваться в одном из сервисов встраивания шрифтов, который выполнит эту работу за вас. За определенную плату там можно получить доступ к высококачественным шрифтам, сервис также сам занимается лицензированием и защитой шрифтов для типографов. На таком сервисе вам обычно предоставляются интерфейс и инструменты, которые превращают встраивание шрифта в столь же простую процедуру, как копирование и вставка.

Сервисы предлагают различные схемы оплаты. Некоторые взимают ежемесячную плату, другие же берут оплату сразу за каждый шрифт. Иногда взимается дополнительная плата за обеспечение повышенной пропускной способности сайта. Как правило, имеются и многоуровневые формы оплаты, которые варьируются от бесплатных услуг до услуг, предоставляемых за сотни долларов в месяц.

Далее указаны некоторые сервисы встраивания шрифтов, которые были популярны на момент подготовки книги. Помимо этого, стоит самостоятельно поискать такие спривисы в Интернете и ознакомиться с новейшими предложениями.

- Google Web Fonts (www.google.com/webfonts) — этот сервис предоставляет доступ к сотням шрифтов с открытым исходным кодом, которые бесплатны для коммерческого использования. Вам необходимо выбрать шрифт, скопировать и вставить в свой документ код, который сгенерирован для вас. Если вы не хотите тратить деньги на покупку шрифтов и вы не очень хорошо разбираетесь в тонкостях вопроса, этот способ наиболее подходящий. Мы обратимся к нему при выполнении первого упражнения главы.
- Typekit от Adobe (www.typekit.com) — это первый сервис веб-шрифтов от Adobe. Он применяет JavaScript для связывания шрифтов с сайтом так, чтобы повысить производительность и качество при отображении во всех браузерах. Я также рекомендую обратиться к блогу, где размещены статьи о принципах работы этого сервиса (blog.typekit.com).
- Fonts.com (fonts.com) — этот сервис располагает самой большой коллекцией шрифтов от крупнейших типографов. Если вам необходим какой-то конкретный шрифт, то здесь, вероятно, вы его и обнаружите.

- ▶ Есть и другие сервисы: Cloud Typography от Hoefler & Co. (www.typography.com/cloud/welcome/), Typotheque (www.typotheque.com/webfonts) и Fonts Live (www.fontslive.com). Эти сервисы различаются по количеству шрифтов, которые они предлагают, и схемами оплаты, поэтому, возможно, вы захотите к ним присмотреться. Сервис Fontstand ([fontstand.com/](http://fontstand.com)) позволяет арендовать шрифты ежемесячно, что, в зависимости от того, как вы их используете, может быть выгоднее покупки шрифта напрямую.

Подведем по веб-шрифтам некоторые итоги

На ваше усмотрение предоставляется выбор метода добавления шрифтов на сайт. Если вам требуется полное управление ситуацией, собственноручное размещение имеющегося шрифта может стать для вас хорошим вариантом. Если же вам нужен конкретный, широко известный шрифт, поскольку этого требует бренд вашего клиента, можно найти его на каком-либо сервисе веб-шрифтов. Если же хочется поэкспериментировать с веб-шрифтами и вам это доставляет удовольствие, выбирайте из того, что свободно доступно, обращаясь к Google Web Fonts.

Представленная здесь информация служит хорошей основой для включения веб-шрифтов в веб-страницы. Скорее всего, в течение ближайших нескольких лет ситуация изменится, поэтому обязательно проведите собственное исследование, когда приступите к самостоятельной работе.

Общие семейства шрифтов

Упоминание о «каком-либо другом шрифте без засечек» — заслуживает детального рассмотрения. «Шрифт без засечек» — это только одно из пяти общих семейств шрифтов, которые можно указать с помощью свойства `font-family`. Если указано общее семейство шрифтов, браузер выбирает доступный шрифт из этой стилистической категории. На рис. 12.2 показаны примеры каждого семейства шрифтов.

- **serif** — у шрифтов с засечками (например: Times, Times New Roman, Georgia) на концах некоторых буквенных штрихов имеются декоративные пластинчатые выступы (засечки).
- **sans-serif** — у шрифтов без засечек (например: Arial, Arial Black, Verdana, Trebuchet MS, Helvetica, Geneva) штрихи прямые, не оканчивающиеся засечками.
- **monospace** — в моноширинных шрифтах (иногда их называют шрифтами с постоянной шириной символов) все символы занимают в строке одинаковое пространство (например: Courier, Courier New и Andale Mono). Заглавная буква W будет у них иметь такую же ширину, как и строчная буква i. Сравните с пропорциональными шрифтами (например, со шрифтом, которым набран этот текст), у которых разные символы имеют разную, соответствующую им ширину.
- **cursive** — курсивные шрифты (например: Apple Chancery, Zapf-Chancery и Comic Sans) имитируют рукописный текст.
- **fantasy** — шрифты семейства **fantasy** (например: Impact, Western и им подобные) являются чисто декоративными и подходят в основном для заголовков.

ЗАПИСЬ НАЗВАНИЙ СЕМЕЙСТВ ШРИФТОВ

В правилах стиля названия общих семейств шрифтов не вводятся с заглавной буквы.

шрифт с засечками	Декоративные штрихи		Hello Times	Hello Georgia
шрифт без засечек	Прямые штрихи		Hello Times New Roman	Hello Lucida
моноширинный шрифт	Пропорциональный шрифт (каждый символ имеет свою ширину)	 Wi	Hello Verdana	Hello Trebuchet MS
		 Wi	Hello Arial	Hello Arial Black
курсив		 Hello	Hello Courier	Hello Andale Mono
декоративный шрифт		 Hello	Hello Comic Sans	Hello Snell
		 HELLO	HELLO Stencil	HELLO Mojo

Рис. 12.2. Примеры пяти общих семейств шрифтов

Стратегии стека шрифтов

Удобнее всего при выборе для веб-страницы шрифта, который, на ваш взгляд, сформирует у пользователей нужное стилистическое впечатление, начинать с первого приемлемого варианта, добавить также некоторые аналогичные альтернативы, а завершить выбор общим семейством шрифтов. Например, если вам необходим один прямой шрифт без засечек, можно первым назначить веб-шрифт (Oswald), добавить в стек несколько наиболее распространенных аналогичных шрифтов (Univers, Tahoma, Geneva) и завершить назначение просто общим шрифтом без засечек:

```
font-family: Oswald, Univers, Tahoma, Geneva, sans-serif;
```

Количество выбираемых шрифтов не ограничено, но многие дизайнеры стараются включать в стек не более чем 10 шрифтов.

Хороший стек шрифтов должен включать стилистически связанные шрифты, про которые известно, что они установлены на большинстве компьютеров. Придерживаясь шрифтов, которые поставляются с операционными системами Windows, macOS и Linux, а также шрифтов, устанавливаемых с популярными пакетами программного обеспечения, такими, как Microsoft Office и Adobe Creative Suite, получим для выбора полный список «веб-безопасных» шрифтов. Удобно искать стилистически связанные

веб-безопасные шрифты в CSS Font Stack (www.cssfontstack.com). В Интернете доступно большое количество статей о стратегиях, относящихся к стеку шрифтов. Я же настоятельно рекомендую работу Майкла Така (Michael Tuck) «8 Definitive Font Stacks» (www.sitepoint.com/eight-definitive-font-stacks), которая будет вам несомненно полезна.

Итак, как вам уже понятно, указание шрифтов для Интернета — всего лишь их предложение. Невозможно строго задать, какой именно шрифт увидят пользователи. Вы можете предложить что-то на выбор первым, можете также подготовить общий запасной вариант. В этом состоит одна из особенностей веб-дизайна, которую следует учитывать.

Похоже, именно сейчас удобно приступить к форматированию меню «Black Goose Bistro». В процессе знакомства с новыми свойствами мы будем по одному добавлять к нему новые правила стиля и начнем с *упражнения 12.1*.

УПРАЖНЕНИЕ 12.1. ФОРМАТИРОВАНИЕ МЕНЮ

В этом упражнении мы изменим шрифты для основного текста и основных заголовков документа `menu.html`, содержащего меню «Black Goose Bistro» и доступного на сайте по следующему адресу: learningwebdesign.com/5e/materials. Откройте этот документ в текстовом редакторе. Вы также можете открыть его в браузере для просмотра «необработанного» состояния. Меню имеет примерно такой вид, как показано на рис. 12.1, сзади. Не упускайте из виду этот документ, потому что в последующих упражнениях мы продолжим с ним работать, задавая шрифтам дополнительные свойства.

Я включила в это упражнение встроенный шрифт, чтобы показать вам, как легко работать с такими сервисами, как Google Web Fonts.

1. В этом упражнении мы применим встроенную таблицу стилей. Начните с добавления в заголовок документа (`head`) элемента `style` — например, так:

```
<head>
  <title>Black Goose Bistro</title>
  <style>

  </style>
</head>
```

2. Желательно, чтобы основной текст отображался с помощью шрифта *Verdana* или какого-либо другого шрифта без засечек. Вместо написания правила для каждого элемента документа напишем одно правило для элемента `body`, которое затем будет наследоваться всеми содержащимися в нем элементами. Добавьте такое правило во встроенную таблицу стилей:

```
<style>
  body {font-family: Verdana, sans-serif;}
</style>
```

3. Для заголовка «Black Goose Bistro, Summer Menu» я решила назначить декоративный шрифт и выбрала бесплатно распространяемый шрифт *Marko One* с сервиса Google Web Fonts (www.google.com/webfonts). Вместе со шрифтом

Google предоставила нам код для связи находящегося на сервере файла шрифта с нашим HTML-файлом (в действительности он представляет собой ссылку на внешнюю таблицу стилей). Этот код должен размещаться в разделе `head` документа, поэтому скопируйте его точно так, как он отображается, но расположите в одной строке и поместите после элемента `title` и перед элементом `style`:

```
<head>
    <title>Black Goose Bistro</title>
    <link href="http://fonts.googleapis.com/>
css?family=Marko+One" rel="stylesheet">
    <style>
    ...

```

4. Запишите правило, которое применяет шрифт к элементу `h1`. Заметьте, что в качестве запасных вариантов я выбрала шрифт Georgia и любой шрифт с заческами:

```
<style>
    body {font-family: Verdana, sans-serif;}
    h1 {font-family: "Marko One", Georgia, serif;}
</style>
```

5. Сохраните документ и перезагрузите страницу в окне браузера — вы получите изображение, похожее на то, что показано на рис. 12.3. Обратите внимание, что для просмотра заголовка, выделенного шрифтом Marko One, необходимо подключение к Интернету и ваш браузер. Размер текста будет изменен при выполнении следующего упражнения.

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

Hours: Monday through Thursday: 11 to 9, Friday and Saturday: 11 to midnight

Appetizers

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. \$3.95

Southwestern napoleons with lump crab — new item!

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. \$7.95

Main courses

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — new item!

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy.** \$12.95

Shrimp sate kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. \$12.95

Grilled skirt steak with mushroom fricassee

Flavorful skirt steak marinated in asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. \$16.95

* We are required to warn you that undercooked food is a health risk.

Рис. 12.3. Вид меню после внесения изменений лишь в семейства шрифтов

Свойство **font-size** (указание размера шрифта)

Для указания размера текста воспользуйтесь свойством `font-size`:

font-size

- **Значения:** единица длины | процентное соотношение | `xx-small` | `x-small` | `small` | `medium` | `large` | `x-large` | `xx-large` | `smaller` | `larger`.
- **По умолчанию** — `medium`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Размер текста можно указать несколькими способами:

- с помощью одной из единиц длины CSS:

```
h1 { font-size: 1.5em; }
```

При указании размера таким способом убедитесь, что название единицы измерения (ее аббревиатура) сразу же следует за числом без дополнительного пробела между ними (см. врезку «*Назначение размеров*»). Напомню, что единицы длины CSS рассматривались в главе 11. Вы также можете обратиться к врезке «*Единицы длины CSS*».

- значением в процентах, увеличенным или уменьшенным относительно размера унаследованного шрифта элемента:

```
h1 { font-size: 150%; }
```

- применением одного из абсолютных ключевых слов: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`. Для большинства современных браузеров заданному по умолчанию размеру шрифта соответствует параметр `medium`.

```
h1 { font-size: x-large; }
```

НАЗНАЧЕНИЕ РАЗМЕРОВ

При назначении размеров единица измерения должна указываться за числом размера без пробела — например:

```
margin: 2em;
```

Добавление пробела перед единицей измерения приводит к тому, что свойство окажется неработоспособным:

```
INCORRECT: margin: 2 em;
```

Для нулевых значений единицу измерения можно опускать:

```
margin: 0;
```

ЕДИНИЦЫ ДЛИНЫ CSS

А теперь вспомним единицы CSS, применяемые для значений длины:

Относительные единицы измерения:

`em` `ex` `rem` `ch`

`vw` `vh` `vmin` `vmax`

Абсолютные единицы измерения:

`px` `in` `mm` `cm`

`q` `pt` `pc`

- применением относительных ключевых слов: `larger` или `smaller` — это изменяет размер текста по отношению к окружающему его тексту (делает его соответственно больше или меньше):

```
strong { font-size: larger; }
```

Следует также заметить, что независимо от описанных здесь возможностей в современном веб-дизайне предпочтительными значениями для `font-size` являются относительные единицы длины `em` и `rem`, а также их процентные значения. Размер шрифта можно указать и в пикселях (`px`), но, в целом, такой подход не обеспечивает гибкости, необходимой при дизайне веб-страницы. Остальные абсолютные единицы (`pt`, `pc`, `in` и т. д.) также не применяются, разве что таблица стилей создается специально для вывода документа на печать.

ПРЕДПОЧТИТЕЛЬНЫЕ ЗНАЧЕНИЯ

Предпочтительными значениями для `font-size` являются `em`, `rem` и `%`.

Вскоре я объясню вам смысл назначения размера шрифта на основе ключевых слов, но давайте сначала рассмотрим наилучшую практику — использование относительных значений.

Определение размеров текста с помощью относительных величин

Наиболее успешной практикой по установке размера шрифта элементов веб-страницы является учет предпочтений пользователя. Относительные значения размеров `rem`, `em` и `%` позволяют использовать размер шрифта по умолчанию в качестве основы для пропорционального определения размеров других текстовых элементов. Не столь важно, чтобы к заголовкам применялось именно 24 пикселя — важно, чтобы заголовки хорошо смотрелись, то есть имели величину в 1,5 раза больше основного текста. При этом, если пользователь изменит настройки, увеличивая размер шрифта, заданный по умолчанию, заголовки также соответственно увеличиваются.

Для поддержания заданного по умолчанию в браузере размера шрифта установите значение свойства `font-size` в корневом элементе `html`, равное 100%:

```
html {  
    font-size: 100%;  
}
```

УСТАНАВЛИВАЙТЕ ЗНАЧЕНИЕ СВОЙСТВА `FONT-SIZE`, РАВНОЕ 100%, В ЭЛЕМЕНТЕ `HTML`

Хотя обычной практикой является присваивание свойству `font-size` значения, равного 100%, в элементе `body`, но его установка в элементе `html` демонстрирует более гибкий подход.

Тем самым вы зададите основу для относительной калибровки. Поскольку заданный по умолчанию размер шрифта для всех современных браузеров составляет 16 пикселов, мы примем для дальнейших рассуждений, что базовый размер шрифта этой величины соответствует 16 пикселям (впрочем, не следует забывать, что действительный размер шрифта может от нее отличаться).

Значения rem

Единица измерения `rem`, которая означает «root em» («корневое em»), всегда зависит от размера корневого (`html`) элемента. Если корневой размер равен 16 пикселям, `rem` также равна 16 пикселям. Блоки текста, которым присвоен размер с помощью единицы `rem`, всегда относятся к одному и тому же элементу и имеют одинаковый размер по всему документу, в котором используются. Таким образом, единица `rem` функционирует как абсолютная единица. Однако, если корневой размер отличается от 16 пикселов, отмеченные значениями `rem` соответствующие элементы пропорционально изменят размеры. И этот вариант — наилучший выбор.

В следующем примере приводится заголовок, размер шрифта которого задан с помощью значений `rem`:

```
h1 { font-size: 1.5rem; } /* 1.5 x 16 = 24 */
```

Единицы измерения em

Единицы измерения `em` основаны на размере шрифта текущего элемента. Когда вы указываете `font-size` в единицах измерения `em`, размер шрифта определяется относительно унаследованного этим элементом размера. Вычисленное значение `em` для элемента можно применять и для задания других размеров, таких, как поля, отступы, ширина элементов и любых прочих параметров, которые относительны по отношению к размеру шрифта.

ЕДИНИЦЫ ИЗМЕРЕНИЯ REM НЕ ПОДДЕРЖИВАЮТСЯ БРАУЗЕРОМ INTERNET EXPLORER 8 И БОЛЕЕ РАННИМИ

Учтите, что единицы измерения `rem` не поддерживаются браузером Internet Explorer 8 и более ранними версиями. Если необходима поддержка устаревшими браузерами, следует предоставить резервное объявление, заданное в пикселях. Имеются также инструменты, автоматически изменяющие единицы `rem` на пиксели (см. главу 20).

Для указания размера `h1`, который наследует от корня заданный по умолчанию размер шрифта, равный 16 пикселям, здесь использованы единицы `em`:

```
h1 { font-size: 1.5em; } /* 1.5 x 16 = 24 */
```

При использовании единиц `em` имеется несколько трудностей. Одна из них заключается в том, что из-за ошибок округления при обращении к единицам `em` возникает некоторая несогласованность в том, как браузеры и платформы интерпретируют текст, размер которого задан в `em`.

Другая трудность в применении единиц `em` состоит в том, что они базируются на унаследованном размере элемента, то есть их размер основан на применяемом контексте.

В следующем примере элемент `h1` основан на унаследованном размере, равном 16 пикселям. Если бы этот элемент `h1` отобразился в элементе `article`, размер шрифта которого установлен равным 14 пикселям, этот элемент наследовал бы размер 14 пикселов, а его результирующий размер стал бы равен 21 пикселу ($1.5 \times 14 = 21$). На рис. 12.4 показаны полученные результаты.

Разметка

```
<h1>Headline in Body</h1>
<p>Pellentesque ligula leo, ...</p>
<article>
  <h1>Headline in Article</h1>
  <p>Vivamus ...</p>
</article>
```

Стили

```
h1 {
  font-size: 1.5em; /* sets all h1s to 1.5em */
}
article {
  font-size: .875em /* 14 pixels based on 16px default */
}
```

Headline in Body

Pellentesque ligula leo, dictum sit amet gravida ac, tempus at risus. Phasellus pretium mauris mi, in tristique lorem egestas sit amet. Nam nulla dui, porta in lobortis eu, dictum sed sapien. Pellentesque sollicitudin faucibus laoreet. Aliquam nec neque ultrices, faucibus leo a, vulputate mauris. Integer rhoncus sapien est, vel eleifend nulla consectetur a. Suspendisse laoreet hendrerit eros in ultrices. Mauris varius lorem ac nisi bibendum, non consectetur nibh feugiat. Vestibulum eu eros in lacus mollis sollicitudin.

Headline in Article

Vivamus a nunc mi. Vestibulum ullamcorper velit ligula, eget iaculis augue ultricies vitae. Fusce eu erat neque. Nam auctor nisl ut ultricies dignissim. Quisque vel tortor mi. Mauris sed aliquet orci. Nam at lorem efficitur mauris suscipit tincidunt a et neque.

Рис. 12.4. Размеры всех элементов h1 составляют 1,5em, но их размеры различны вследствие контекста, в котором они отображаются

Из этого примера понятно, что размеры элементов, выраженные в em, могут быть разными, если эти элементы находятся в разных частях документа. Если необходимо, чтобы значение h1 в статье также составляло 24 пикселя, можно рассчитать значение em, разделив целевой размер на его контекст: $24 : 14 = 1,71428571\text{em}$ (нет необходимости округлять эту цифру в сторону уменьшения — браузеру известно, как с ней обращаться).

Если элементы вложены на несколько слоев в глубину, размер компонентов может как увеличиваться, так и уменьшаться, что иногда приводит к проблемам. При наличии большого количества уровней вложенности текст может отобразиться слишком малым. При работе с единицами измерения em обращайте внимание на контекст

ФОРМУЛА ИТАНА МЕРКОТТА

Итан Меркотт (Ethan Marcotte) в своей книге «*Responsive Web Design*» (*A Book Apart*) представил формулу: «цель : контекст = результат». Полезно ознакомиться с этой книгой для преобразования значений пикселов в проценты и em.

и создайте учитывающие контекст правила стиля.

Весьма сложная природа единиц измерения em привела к популярности более предсказуемой единицы измерения rem.

Процентные значения

Ранее отмечалось, что процентное значение (100%) применялось для сохранения заданного по умолчанию размера шрифта, но процентные значения можно применять для любого элемента. Это достаточно просто.

В следующем примере `h1` наследует от элемента `html` заданный по умолчанию размер, равный 16px, а примененное значение, равное 150%, увеличивает это унаследованное значение, в результате чего мы получаем величину `h1`, равную 24 пикселям:

```
h1 { font-size: 150%; } /* 150% of 16 = 24 */
```

ФОРМУЛА ЦЕЛЕВОГО РАЗМЕРА

Для вычисления значений % и em применяйте следующую формулу:

целевой размер : размер контекста = результат.

Ключевые слова

Альтернативную возможность для указания `font-size` представляет собой применение предварительно заданного одного абсолютного ключевого слова: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` или `xx-large`. Ключевые слова не соответствуют определенным единицам измерения, а масштабируются последовательно, по отношению друг к другу. Для современных браузеров характерен размер `medium`, заданный по умолчанию. На рис. 12.5 показано, каким образом каждое из абсолютных ключевых слов отображается в браузере, если по умолчанию для текста установлено значение, равное 16 пикселям. Чтобы продемонстрировать значительную разницу в удобочитаемости при небольших размерах и одинаковом базовом размере, в примере задействованы образцы шрифтов `Verdana` и `Times`. Шрифт `Verdana` разработан для разборчивого отображения текста на экранах при небольших размерах шрифта, а шрифт `Times` создан для печати, поэтому в нашем представлении он менее разборчив.

This is an example of the default text size in `Verdana`.

`xx-small` | `x-small` | `small` | `medium` | `large` | `x-large` | `xx-large`

This is an example of the default text size in `Times`.

`xx-small` | `x-small` | `small` | `medium` | `large` | `x-large` | `xx-large`

Рис. 12.5. Изменение размера текста с помощью абсолютных ключевых слов

Относительные ключевые слова, `larger` и `smaller`, применяются для изменения размера текста относительно размера текста родительского элемента. Точная величина изменения размера определяется каждым браузером и не контролируется пользователем. Несмотря на данное ограничение, это простой способ увеличить или уменьшить размер шрифта, если точные пропорции не являются критическими.

Применим новые сведения, полученные при изучении шрифтов CSS, при выполнении упражнения 12.2.

УПРАЖНЕНИЕ 12.2. НАСТРОЙКА РАЗМЕРА ШРИФТА

Уточним размер некоторых текстовых элементов, что позволит придать меню нашего бистро изысканный вид. Откройте в текстовом редакторе файл `menu.html` и следуйте инструкциям. В любой момент можно сохранить документ и посмотреть в браузере результаты работы. Вы также можете поэкспериментировать с другими значениями размеров.

- Существует большое число подходов, используемых для задания размера текста на веб-страницах. В этом упражнении мы начнем с установления размера шрифта для элемента `body`, равного 100%, что затем облегчит задание размеров с помощью единиц `em`:

```
body {
    font-family: Verdana, sans-serif;
    font-size: 100%;
}
```

- Текст, отображаемый в браузере по умолчанию, имеет размер 16 пикселов, чего вполне достаточно для основного текста главной страницы, а вот внешний вид заголовков желательно улучшить. Пусть теперь размер основного заголовка составит 24 пикселя, что в полтора раза больше основного текста [$\text{цель} (24) : \text{текст} (16) = 1.5$]. Добавим новое правило, устанавливающее размер `h1` равным 1,5`em`. Этого же результата можно достичь, используя значение, равное 150%:

```
h1 {
    font-size: 1.5em;
}
```

- Зададим заголовкам `h2` тот же размер, что и у основного текста, — так они будут лучше сочетаться со страницей:

```
h2 {
    font-size: 1em;
}
```

На рис. 12.6 показан результат наших усилий по изменению размера шрифта

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts
Hours: Monday through Thursday: 11 to 9, Friday and Saturday; 11 to midnight

Appetizers

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. \$3.95

Southwestern napoleons with lump crab — **new item!**

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas.
\$7.95

Main courses

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Рис. 12.6. Меню сайта после внесения нескольких небольших изменений в размеры шрифтов заголовков

Свойство **font-weight** («жирность» шрифта)¹

После рассмотрения семейств шрифтов и правил задания их размеров остальные свойства шрифтов уже не покажутся вам сложными. Например, если нужно, чтобы текстовый элемент отображался полужирным шрифтом, примените свойство `font-weight` для настройки уровня «жирности» шрифта:

font-weight

- **Значения:** `normal` | `bold` | `bolder` | `lighter` | `100` | `200` | `300` | `400` | `500` | `600` | `700` | `800` | `900`.
- **По умолчанию** — `normal`.
- **Применение** — ко всем элементам.
- **Наследование** — да

Как можно видеть, свойство `font-weight` имеет большое число предопределенных значений, включая описательные термины: `normal`, `bold`, `bolder` и `lighter` и девять числовых значений (от 100 до 900) для указания различных степеней «жирности» шрифта, если они доступны.

Поскольку большинство обычно используемых в Интернете шрифтов имеют только два варианта «жирности»: обычный (или `Roman`) и полужирный (`bold`), единственное значение, задающее «жирность» шрифта, которое используется в большинстве случаев, — `bold`. Вы также можете применить значение `normal`, чтобы устраниТЬ форматирование полужирным шрифтом какого-либо текста — например, выделенного (`strong text`) или заголовков, — который в противном случае отображался бы полужирным шрифтом.

Чтобы разобраться с использованием веб-шрифтов с большим диапазоном «жирностей», лучше всего обратиться к числовой диаграмме, приведенной на рис. 12.7 (существует несколько веб-шрифтов Google, для которых необходимы числовые значения размеров). Если какие-либо термины, описывающие «жирность», недоступны, для выделения текста полужирным

This is an example of the default text in Verdana.

`normal` | **bold** | **bolder** | `lighter`

`100` | `200` | `300` | `400` | `500`

600 | **700** | **800** | **900**

This is an example of the default text in Times.

`normal` | **bold** | **bolder** | `lighter`

`100` | `200` | `300` | `400` | `500`

600 | **700** | **800** | **900**

Рис. 12.7. Эффект применения различных значений свойства `font-weight` (а также отсутствие эффекта!)

СВОЙСТВО **FONT-SYNTHESIS**

В модуле CSS Fonts уровня 3 появилось свойство `font-synthesis`, благодаря которому авторы отключают (со значением `none`) или разрешают синтез полужирных шрифтов (значение `weight`). Однако эти возможности пока находятся на стадии эксперимента.

¹ В русской традиции принято называть «жирные» шрифты полужирными, так мы далее и поступим там, где это уместно (Прим. ред.).

шрифтом, как показано на рис. 12.7, можно применить числовые настройки, равные 600 и выше (хотя эти значения и варьируются в зависимости от браузера).

Если отдельный «жирный» вид шрифта недоступен, браузер может «синтезировать» такой шрифт, увеличивая «жирность» доступного обычного шрифта.

Свойство **font-style** (курсив)

Свойство **font-style** оказывает влияние на *стиль* текста, то есть определяет вертикальны ли очертания букв (`normal`) или наклонны (`italic` и `oblique`):

font-style

- **Значения:** `normal` | `italic` | `oblique`.
- **По умолчанию** — `normal`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Примените свойство `font-style` для выбора курсивного (`italic`) стиля текста. Другое распространенное использование этого свойства — задание стиля `normal` для текста, оформленного курсивом согласно заданному в браузерах по умолчанию правилу форматирования текста, подлежащего выделению. Применяется также и значение `oblique`, тоже определяющее наклонную версию шрифта, но браузеры обычно отображают `oblique` так же, как `italic`.

Поэкспериментируйте со значениями «жирности» и стиля текста при выполнении *упражнения 12.3*.

УПРАЖНЕНИЕ 12.3. ПРЕОБРАЗОВАНИЕ ТЕКСТА С ПРИМЕНЕНИЕМ СВОЙСТВ **bold** И **italic**

Вернемся к нашему меню. Я решила, что все названия пунктов меню должны выделяться полужирным шрифтом. Но для этого не следует заключать каждый пункт в теги `` — как мы делали в 1996-м году! Я также не собираюсь отмечать их как элементы `strong` — такой подход не является семантически точным. Вместо этого следует применить стиль к семантически корректным элементам `dt` — тогда они все сразу станут выделены полужирным шрифтом. Добавьте такое правило в конец таблицы стилей, сохраните файл и просмотрите его в браузере:

```
dt { font-weight: bold; }
```

Теперь все названия пунктов меню выделены полужирным шрифтом, но часть текста, которая помечена как `strong`, выделяется не так сильно, как хотелось бы, поэтому выделим ее курсивом для большего акцентирования. Это можно сделать, применив свойство `font-style` к элементу `strong`:

```
strong { font-style: italic; }
```

Снова сохраните документ и перезагрузите окно браузера. Меню должно иметь вид, показанный на рис. 12.8.

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

Hours: Monday through Thursday: 11 to 9, Friday and Saturday; 11 to midnight

Appetizers

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. \$3.95

Southwestern napoleons with lump crab — new item!

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. \$7.95

Main courses

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — new item!

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy.** \$12.95

Shrimp saté kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. \$12.95

Рис. 12.8. Результат применения свойств font-weight и font-style

Свойство **font-variant** в CSS2.1 (капитель)

font-variant

- **Значения:** normal | small-caps.
- **По умолчанию** — normal.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Некоторые гарнитуры шрифтов включают шрифты, называемые *капитель* (small caps, маленькие буквы). Речь при этом идет об отдельном дизайне шрифта, в котором строчные знаки выглядят как уменьшенные заглавные. Для поддержания гармонии буквы капители прописаны с учетом соответствия размеру и плотности шрифта нижнего регистра.

Капитель может быть использована для появляющихся в текстовом потоке строк из трех или более заглавных букв — таких, как аббревиатуры и сокращения, которые, будучи набраны заглавными буквами, обычно выглядят слишком громоздко. Сравните: НАСА и США — в стандартном шрифте с наса и сша при использовании капители. Капитель также рекомендуется для указания времени — например: 1:00 или 2017AD.

Когда свойство font-variant только появилось в CSS2.1, единственная возможность, которую оно предоставляло дизайнерам, — просто задание капители для текстовых элементов. CSS3 значительно расширил роль font-variant, о чем речь пойдет в разд. «Расширенная CSS3-тиография» далее. Но пока мы рассматриваем CSS2.1 версию свойства font-variant.

В большинстве случаев браузеры просто имитируют капитель, уменьшая заглавные буквы текущего шрифта. Для любителей типографики это далеко не идеальный подход, который нарушает гармонию строк, однако можно подобрать приемлемый

вариант для внесения разнообразия в небольшие отрывки текста. Обратите внимание на пример использования маленьких заглавных букв при выполнении *упражнения 12.5* (свойство `font-variant`).

Свойство `font-stretch` (сжатый и разреженный шрифт)

`font-stretch`

- **Значения:** `normal` | `ultra-condensed` | `extra-condensed` | `condensed` | `semi-condensed` | `semi-expanded` | `expanded` | `extra-expanded` | `ultra-expanded`.
- **По умолчанию** — `normal`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Design
Universe Ultra Condensed

Design
Universe Condensed

Design
Univers

Design
Universe Extended

Рис. 12.9. Сверху вниз: примеры сильно сжатой, сжатой, нормальной и разреженной версий гарнитуры Universe

CSS3-свойство `font-stretch` указывает браузеру на необходимость выбора в семействе шрифтов обычного, сжатого или разреженного шрифта (рис. 12.9). Если браузер не может найти подходящий шрифт, он *не будет* предпринимать попыток синтезировать ширину шрифта, растягивая или сжимая текст, а заменит текущий шрифт на шрифт другой ширины. В настоящее время поддержка браузерами этого свойства лишь только реализуется, и на момент подготовки книги она присутствовала в браузерах IE11+, Edge, Firefox, Chrome 48+, Opera и Android 52+, но еще отсутствовала в Safari или iOS Safari. Впрочем, в скором времени все может измениться.

Сокращенное свойство `font`

Указание нескольких свойств шрифта для каждого текстового элемента может представлять повторяющийся и длинный ряд значений, поэтому разработчики CSS предоставили сокращенное свойство `font`, которое объединяет в одно правило все связанные со шрифтом свойства.

`font`

- **Значения:** `font-style font-weight font-variant font-stretch font-size/line-height font-family` | `caption` | `icon` | `menu` | `message-box` | `small-caption` | `status-bar`
- **По умолчанию** — для каждого свойства в списке зависит от значения, заданного по умолчанию.

- **Применение** — ко всем элементам.
- **Наследование** — да.

Значение свойства `font` представляет собой список разделенных символьными пробелами значений для всех показанных свойств шрифта. Следует отметить, что в сокращенном свойстве `font` можно применять только версию CSS2.1 `font-variant (small-caps)`, что является одной из причин, почему я описываю это свойство в отдельном разделе. В свойстве `font` важен порядок следования значений:

```
{ font: style weight stretch variant size/line-height font-family; }
```

Как минимум, свойство `font` должно в указанном порядке включать значения `font-size` и `font-family`. Если пропустить одно из значений или разместить их в неправильном порядке, правило полностью станет недействительным. Рассмотрим пример минимального значения свойства шрифта:

```
p { font: 1em sans-serif; }
```

После указания размера и семейства шрифта другие значения не являются обязательными и могут отображаться в любом порядке, но до `font-size`. Если стиль (курсив/не курсив), жирность, сжатие/разреженность или указание на капитель для шрифта пропущены, его значение по этим свойствам устанавливается как `normal`. Это позволяет с помощью сокращенного свойства легко переопределять предыдущие настройки, поэтому будьте внимательны при его использовании.

В этом разделе мы увидели новое для нас значение: `line-height`. Как следует из названия, это значение регулирует высоту текстовой строки и применяется для создания интервала между строками текста. Записывается оно сразу после `font-size` и разделяется слэшем (косой чертой), как показано в следующих примерах. Свойство `line-height` подробно рассматривается в этой главе далее.

```
h3 { font: oblique bold  
small-caps 1.5em/1.8em  
Verdana, sans-serif; }  
h2 { font: bold 1.75em/2  
sans-serif; }
```

ПРОПУЩЕННОЕ СВОЙСТВО СБРАСЫВАЕТСЯ ДО ЗАДАННОГО ПО УМОЛЧАНИЮ

Будьте внимательны при использовании сокращенных свойств — таких, как `font`. Пропущенное свойство тут же сбрасывается до заданного по умолчанию значения. С другой стороны, короткая цепочка в свойстве `font` — хороший способ сбросить все предыдущие настройки прочих свойств, если это необходимо.

ЕЩЕ О О КРОСС-БРАУЗЕРНОЙ ПОДДЕРЖКЕ

При включении значения для нового свойства `font-stretch` в сокращенное свойство `font` сначала задайте версию, где сжатие/разреженность не используется, — для браузеров, которые его не поддерживают. В итоге получим два объявления:

```
h3 {  
    font: bold 1.25em  
    Helvetica;  
    font: bold extended 1.25em  
    Helvetica;  
}
```

При выполнении *упражнения 12.4* мы воспользуемся сокращенным свойством `font` для внесения некоторых изменений в заголовки `h1` меню нашего бистро.

Системные шрифты для ключевых слов

Свойство `font` также имеет ряд значений, применяемых для выделения ключевых слов (`caption`, `icon`, `menu`, `message-box`, `small-caption` и `status-bar`), представляющих *системные шрифты*, то есть шрифты, используемые операционными системами для таких объектов, как надписи для значков и пунктов меню. Эти значения могут быть полезны при разработке веб-приложений с учетом требований среды, в которой работает пользователь. Указанные ключевые слова являются сокращенными значениями, поскольку инкапсулируют шрифт, размер, стиль и «жирность» шрифта для каждой из возможных целей.

Упражнение 12.4 столь же короткое, как и сокращенное свойство `font`, и несложное в выполнении.

УПРАЖНЕНИЕ 12.4. ПРИМЕНЕНИЕ СОКРАЩЕННОГО СВОЙСТВА `FONT`

Добавим в наше меню еще одно новшество, а потом сделаем небольшой перерыв. Для экономии места можно заменить все указанные для элемента `h1` свойства шрифта одним объявлением сокращенного свойства `font`:

```
h1 {  
    font: bold 1.5em "Marko One",  
          Georgia, serif;  
}
```

Может показаться излишним, что в это правило включено значение для полужирного шрифта. В конце концов, элемент `h1` уже по умолчанию выделен полужирным шрифтом, верно? Особенность сокращенных свойств заключается в том, что если вы опустите значение, оно будет сброшено к значению по умолчанию для этого *свойства*, а не к значению браузера по умолчанию.

В нашем случае, если свойство `font-weight` в объявлении `font` было бы пропущено, оно по умолчанию имело бы значение `normal`. Поскольку таблица стилей переопределяет заданный по умолчанию стиль полужирного заголовка браузера, если явно не выделить его полужирным шрифтом в свойстве `font`, элемент `h1` отобразится в тексте в «нормальном» виде.

Сокращенные свойства могут быть весьма коварными в этом смысле. Поэтому обратите внимание, чтобы чего-то не пропустить и случайно не переопределить заданное по умолчанию значение или же унаследованное значение, на которое рассчитывали ранее.

Можно сохранить эти настройки и посмотреть результат в окне браузера. Если все выполнено правильно, меню практически не изменится.

РАСШИРЕННАЯ CSS3-ТИПОГРАФИКА

К этому моменту мы уже располагаем базовым инструментарием для форматирования шрифтов с помощью CSS. Если вам понадобятся декоративные шрифты, рекомендую ознакомиться со всеми свойствами CSS Fonts Module уровня 3, которые позволяют управлять не только выбором символов, но и их расположением. Описания свойств шрифтов в этом разделе будут весьма краткими в целях экономии пространства в книге, а также в связи с тем, что многие из рассматриваемых функций являются экспериментальными и имеют ограниченную поддержку со стороны браузеров. Если вы нуждаетесь в действительно хорошей типографике, вам придется дополнитель но разобраться с этой темой, начиная с ознакомления со спецификацией, находящейся по адресу: www.w3.org/TR/css-fonts-3.

Коллекция свойств *font-variant* в CSS3

Коллекция свойств с префиксом *font-variant* в CSS3 предоставляет дизайнерам и разработчикам доступ к шрифтам со специальными символами (*глифами*), которые могут сделать типографику страницы более выразительной.

Ранее упоминалось, что модуль CSS3 Font Module значительно расширил определение свойства *font-variant*. Теперь оно может служить сокращенным свойством для ряда свойств коллекции *font-variant*. Эти свойства относятся к экспериментальным, но поддержка их браузерами постоянно расширяется. Так что давайте сейчас и посмотрим, как раз вивается управление шрифтами в веб-дизайне.

- *font-variant-ligatures*. *Лигатура* — это глиф, объединяющий в один символ два или более символов. Одним из распространенных примеров лигатуры служит комбинация строчных букв f и i, где точка над i становится частью f (fi). Лигатуры немного сглаживают отображение некоторых некрасивых пар букв, а лигатурные глифы включены во многие шрифты. Свойство *font-variant-ligatures* позволяет управлять применением лигатур на веб-страницах. Это свойство поддерживается качественнее, чем другие свойства, и уже доступно в IE10+, Chrome 34+, а также в Safari и Opera (с префиксом -webkit). Ожидается, что поддержка его браузерами будет постоянно совершенствоваться.
- *font-variant-caps*. Свойство позволяет из набора символов шрифта выбирать глифы капители (*small-caps*, маленьких заглавных букв) вместо моделирования их в браузере. Значение *all-smallcaps* определяет применение капители в качестве прописных и строчных букв. Значение *unicase* приводит

ЕЩЕ О СВОЙСТВАХ *FONT-VARIANT*

За исключением свойства *font-variant-position*, имеющего определенную цель (см. далее), прочие свойства *font-variant* дают отличные возможности для практического совершенствования веб-страниц. Они могут принести пользу, но без них можно и обойтись.

СВОЙСТВО *FONT-VARIANT-LIGATURES*

Свойство *font-variant-ligatures* характеризуется длинным списком значений, который можно найти по адресу: www.w3.org/TR/css-fonts-3/#propdef-font-variant-ligatures.

к применению капитали только для символов верхнего регистра, при этом строчные буквы в слове остаются без изменений. Значение `titling-caps` обеспечивает применение капитали для заголовков, но спроектировано не так строго. Также применяются значения `petite-caps` и `all-petite-caps`.

- `font-variant-position`. Из набора символов шрифта это свойство выбирает глифы верхнего индекса (`super`) или нижнего индекса (`sub`), если они доступны. В противном случае браузер формирует надстрочный или подстрочный текст для элементов `sup` и `sub`, сжимая символ и перемещая его выше или ниже базовой линии.
- `font-variant-numeric`. Свойство позволяет выбирать различные стили числовых символов, если они доступны. Например, можно поставить цифры последовательно или выстроенным в столбцы — как в случае использования электронных таблиц (`proportional-numbers/tabular-numbers`), выбрать цифры в старом стиле (`old-style-nums`), когда некоторые символы опускаются ниже базовой линии, и указать, должны ли дроби представляться через косую черту или в виде вертикальной простой дроби (`diagonal-fractions/stacked-fractions`). Порядковые числа могут отображаться в виде 2^{oe} вместо 2-e (`ordinal`), а нули — с перечеркивающей их косой чертой, как это предпочтительно в некоторых контекстах (`slashed-zero`).
- `font-variant-alternates` Шрифты иногда предлагают для конкретного символа более одного глифа — например, несколько штриховых рисунков для буквы S или старомодную букву s, более похожую на f. Свойство `font-variant-alternates` поддерживает способ для указания ударений и других альтернативных символов. Многие значения этого свойства зависят от шрифта и должны определяться сначала через правило с помощью свойства `@font-features-values`. Обратитесь к спецификации, где содержится более глубокое объяснение этого вопроса.
- `font-variant-east-asian` — это свойство позволяет выбирать определенные азиатские глифы.

Наконец, обновлено старое свойство `font-variant`, существовавшее с начала возникновения CSS, — оно стало сокращенным для всех упомянутых здесь свойств. Вы можете применять его с исходным значением `small-caps`, причем это будет совершенно корректным. Как только эти свойства получат широкое распространение, они станут универсальными.

Прочие свойства из CSS3

Пришло время завершить обзор свойств шрифтов модуля Fonts Module уровня 3. Я дам вам здесь общее представление о том, что доступно (или станет доступным после расширения поддержки браузерами), а также возможность самостоятельно поглубже изучить спецификацию:

- `font-size-adjust`. То, как *выглядит* размер текста на странице, часто связано с высотой строчной буквы x (*x-высотой*), а не с заданным размером текста.

Например, 10-точечный шрифт с относительно большой x-высотой, вероятно, легче читать, чем 10-точечный тип с изящными маленькими строчными буквами. Свойство `font-size-adjust` позволяет браузеру отрегулировать размер резервного шрифта с учетом того, чтобы его x-высота соответствовала x-высоте уже отобранного шрифта. Тогда обеспечивается лучшая разборчивость, даже при использовании резервного шрифта.

- `font-kerning`. *Кернинг* — это пространство между глифами символов. Шрифты обычно содержат метаданные о том, какие пары букв необходимо сблизить, чтобы согласовать интервалы в слове. Свойство `font-kerning` позволяет применять информацию о кернинге шрифта (`normal`), отключать (`none`) или оставлять на усмотрение браузера (`auto`).
- `font-feature-settings`. Это свойство позволяет авторам управлять в шрифтах OpenType расширенными типографскими функциями, которые не имеют широкого применения, — такими, как штрихи, капитали, лигатуры, автоматические дроби и многими другими. Эти функции выглядят знакомо, поскольку многие из них управляются различными свойствами шрифта. Фактически спецификация рекомендует использовать, если это возможно, свойство `font-variant` и резервировать настройки `font-feature-settings` только для крайних случаев. Однако на момент подготовки книги свойство `font-feature-settings` лучше поддерживалось браузерами, поэтому сейчас это — наилучший вариант. Просто имейте в виду, что оно плохо каскадируется, то есть ее легко отменить. Различные приемы работы с CSS содержатся в обзоре Робина Рендла (Robin Rendle), доступном на сайте: css-tricks.com/almancard/properties/f/font-feature-settings.
- `font-language-override` — это экспериментальное свойство управляет применением специфичных для языка глифов.

Итак мы завершили рассмотрение различных способов управления шрифтами в CSS (и это заняло достаточно времени!). Тем не менее это лишь один из аспектов представления текста. Изменение цвета текста также имеет большое значение в дизайне веб-страниц.

ОБЗОР ВОЗМОЖНОСТЕЙ OPENTYPE

Подробный обзор возможностей шрифтов OpenType и их преимуществ содержится в статье «*Caring about OpenType Features*» Тима Брауна (Tim Brown), доступной на сайте Adobe Typekit: (practice.typekit.com/lesson/caring-about-opentype-features/).

СВОЙСТВО **COLOR** (ИЗМЕНЕНИЕ ЦВЕТА ТЕКСТА)

Из главы 11 вы узнали, как изменить цвет текста, и, честно говоря, здесь мало что можно к этому добавить. Да-да, для этого используется свойство `color`:

color

- **Значения:** значение цвета (название или числовое значение).
- **По умолчанию** — зависит от браузера и предпочтений пользователя.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Применение свойства `color` совершенно несложно. Значением его может служить название цвета (обратитесь к врезке «*Названия цветов*») или описывающее определенный цвет числовое значение RGB. Далее приведено несколько примеров, каждый из которых окрашивает элементы `h1`, используемые в документе, в серый цвет:

```
h1 { color: gray; }
h1 { color: #666666; }
h1 { color: #666; }
h1 { color: rgb(102,102,102); }
```

Вам сейчас не следует озабочиваться по поводу числовых значений — просто посмотрите, как это выглядит. Цвета RGB подробно рассматриваются в главе 13, а в этой главе для демонстрационных целей применяются только названия цветов.

Цвет наследуется, поэтому можно задавать в документе цвет для всего текста, применяя свойство `color` к элементу `body`:

```
body { color: fuchsia; }
```

Полагаю, принцип вы поняли, хотя наверняка вряд ли хотите, чтобы весь текст был окрашен в пурпурный цвет.

НАЗВАНИЯ ЦВЕТОВ

В CSS2.1 определяется 17 стандартных названий цветов:

black	white	purple
lime	navy	aqua
silver	maroon	fuchsia
olive	blue	orange
gray	red	green
yellow	teal	

Обновленный модуль CSS Color Module уровня 3 позволяет задавать в таблицах стилей цвета из большего набора, содержащего 140 названий. Образцы каждого из них можно посмотреть на рис. ЦВ-13.2 и на сайте: learningwebdesign.com/colornames.html.

Корректности ради следует отметить, что свойство `color` не является строго связанным со свойствами текста. Согласно спецификации CSS, это свойство применяется для изменения цвета элемента *на переднем плане* (а не фона). Передний план элемента включает как собственно текст, так и его границы. Поэтому, задавая элементу цвет (включая элементы изображения), учтите, что цвет будет применяться и для его границ, если только не указано свойство `border-color`, которое его переопределяет (более подробно о границах и цвете границ речь пойдет в главе 14).

Прежде чем мы займемся добавлением цвет в наше онлайн-меню, я хочу немножко отвлечься и познакомить вас с несколькими типами селекторов, которые предоставляют нам больше гибкости при стилинге элементов в документе.

ЕЩЕ О ТИПАХ СЕЛЕКТОРОВ

До сих пор в качестве селекторов использовались названия элементов. В предыдущей главе было показано, каким образом можно группировать селекторы в списке, располагая их названия через запятую, чтобы позволяет применять свойства сразу к нескольким элементам. Вот уже известные вам примеры селекторов:

Селектор элемента:

```
p { color: navy; }
```

Сгруппированные селекторы:

```
p, ul, td, th { color: navy; }
```

Недостаток выбора элементов таким путем состоит в том, что свойство (присвоение темно-синего цвета) применяется к каждому абзацу и другим имеющимся в документе элементам. А иногда правило желательно применить к определенному абзацу или абзацам. В этом разделе мы рассмотрим три типа селекторов, которые позволяют добиться такого результата: селекторы потомков, селекторы ID и селекторы классов.

Селекторы потомков

Селекторы потомков (descendant selectors) указывают элементы, содержащиеся внутри (и, следовательно, являются потомками) другого элемента. Примером этого может служить *контекстный селектор*, когда элемент выбирается на основе контекста или его отношения к другому элементу. Во врезке «*Другие контекстные селекторы*» приведены некоторые другие селекторы этого типа.

ДРУГИЕ КОНТЕКСТНЫЕ СЕЛЕКТОРЫ

Селекторы потомков — это один из четырех типов контекстных селекторов: так называемые *комбинаторы* (combinators) в спецификациях селекторов уровня 3 и 4). Остальные три контекстных селектора: *дочерние селекторы* (child selectors), *селекторы ближайших братьев и сестер* (next-sibling selectors) и *селекторы последующих братьев и сестер* (subsequent-sibling selectors).

Дочерний селектор

Дочерний селектор походит на селектор потомков, но нацелен только на прямых потомков элемента. Между ними не может быть каких-либо иерархических уровней. Они обозначаются символом «больше чем» (>). Следующее правило влияет на выделенный текст, но только если он непосредственно содержится в элементе `p`. Элемент `em` внутри ссылки (`a`) внутри абзаца затронут не будет:

```
p > em { font-weight: bold; }
```

Селектор ближайших братьев (сестер)

Селектор ближайших братьев (сестер) предназначается для элемента, который идет непосредственно после другого элемента с тем же родителем. Обозначается он знаком «плюс» (+). Следующее правило придает особый режим абзацам, которые следуют за `h1`, а другие абзацы не затрагиваются:

```
h1 + p { font-style: italic; }
```



► Селектор последующих братьев (сестер)

Селектор последующих братьев (сестер) выбирает элемент, который использует родительский элемент совместно с указанным элементом и встречается после него в исходном порядке. Им не нужно следовать напрямую друг за другом. Этот тип селектора является новым в CSS3 и не поддерживается браузером Internet Explorer 8 и более ранними версиями. Следующее правило выбирает любой элемент *h2*, который совместно с *h1* использует родительский элемент (например, *section* или *article*) и отображается в документе после него:

```
h1 ~ h2 {font-weight: normal;}
```

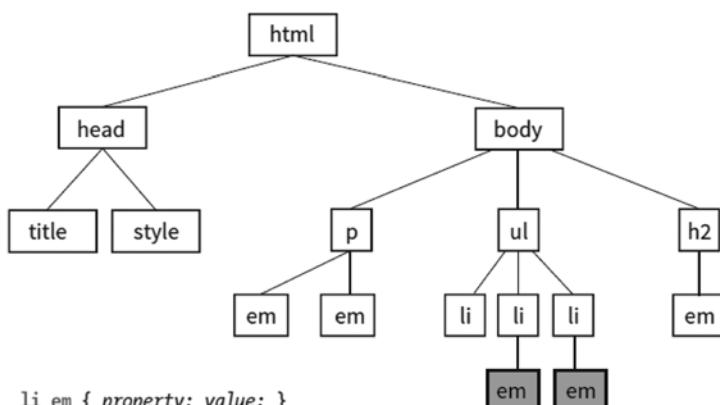
Селекторы потомков указываются в списке через пробел. В следующем примере цвет присваивается выделенным элементам текста (*em*), но только в случае, когда они содержатся в пунктах списка (*li*).

О СМЫСЛЕ СИМВОЛЬНОГО ПРОБЕЛА

Символьный пробел между названиями элементов означает, что второй элемент должен содержаться внутри первого.

Выделенный текст в простых абзацах и в других элементах остается незатронутым (рис. 12.10).

```
li em { color: olive; }
```



```
li em { property: value; }
```

Рис. 12.10. Выбираются только элементы *em*, находящиеся внутри элементов *li*. Другие элементы *em* остаются без изменений

Рассмотрим пример, который показывает, каким образом контекстные селекторы могут группироваться в список через запятую, как утверждалось ранее. Это правило определяет присвоение цвета элементам *em*, но только при отображении их в заголовках *h1*, *h2* и *h3*:

```
h1 em, h2 em, h3 em { color: red; }
```

Можно вложить селекторы потомков и в несколько уровней по глубине. Следующий пример предназначен для элементов *em*, которые отображаются в якорях (*a*) упорядоченных списков (*ol*):

```
ol a em { font-variant: small-caps; }
```

Селекторы ID

В главе 5 шла речь об атрибуте `id`, который придает элементу уникальное идентифицирующее имя (его ссылку `id`).

Атрибут `id` применяется вместе с любым элементом и обычно используется для придания смысла общим элементам `div` и `span`. Селекторы ID позволяют выбирать элементы по их значениям. Индикатором, идентифицирующим селекторы ID, является символ (#), известный нам как символ «решетки».

Рассмотрим пример элемента списка со ссылкой `id`:

```
<li id="sleestak">Sleestak T-shirt</li>
```

Применяя селектор ID, запишем правило стиля для этого пункта списка, например, таким образом (обратите внимание, что символ # предшествует ссылке `id`):

```
li#sleestak { color: olive; }
```

Поскольку значения `id` в документе должны быть уникальными, можно опускать имя элемента. Следующее правило эквивалентно только что приведенному:

```
#sleestak { color: olive; }
```

Селектор ID можно использовать и как часть контекстного селектора. В следующем примере стиль применяется только к элементам `a`, которые содержатся внутри элемента, обозначенного как `resources`. В результате можно обрабатывать ссылки в элементе с названием `resources` иначе, чем прочие ссылки на странице, не имеющие дополнительной разметки:

```
#resources a { text-decoration: none; }
```

Вам уже пора представлять возможности селекторов, а также стратегии их применения вместе с хорошо спланированной семантической разметкой.

Селекторы класса

Рассмотрим последний тип селектора, а затем вернемся к свойствам стиля текста. Другой идентификатор элемента, речь о котором шла в главе 5, — это идентификатор `class`, применяемый для классификации элементов в концептуальной группе. В отличие от атрибута `id`, использовать имя `class` совместно могут несколько элементов. Сам элемент также может принадлежать более чем одному классу.

С помощью селектора классов можно обращаться к элементам, принадлежащим к одному классу. Названия классов обозначаются точкой (.) в начале селектора.

Так, для выбора всех абзацев, содержащих `class="special"`, нужно применять следующий селектор (точка указывает, что следующее слово служит селектором класса):

```
p.special { color: orange; }
```

ЗНАЧЕНИЕ СИМВОЛА

Символ # идентифицирует селектор `ID`.

ЗНАЧЕНИЕ СИМВОЛА ТОЧКИ

Символ точки (.) указывает селектор класса.

Для применения свойства ко *всем* элементам одного класса не указывайте в селекторе имя элемента (но обязательно оставляйте точку — это обозначит класс). Следующий пример предназначен для всех абзацев и любого другого элемента, отмеченного как `class="special"`:

```
.special { color: orange; }
```

Специфичность

В главе 11 мы ввели термин «специфичность», который означает, что некоторые специфичные селекторы имеют больший вес при обработке конфликтов между правилами стиля. Поскольку вам уже известны несколько селекторов, вернемся к этой важной концепции.

Следующий список типов селекторов, расположенных от более специфичных к менее специфичным, поможет вам в большинстве сценариев:

- **встроенные стили** с атрибутом `style` более специфичны, чем (и будут переопределять...);
- **селекторы ID** более специфичны, чем (и будут переопределять...);
- **селекторы класса** более специфичны, чем (и будут переопределять...);
- **индивидуальные селекторы элементов**.

Полное объяснение немного сложнее, но я постараюсь изложить его покороче. Для расчета специфичности нарисуйте три блока:

```
[ ] [ ] [ ]
```

Подсчитайте в селекторе количество ID и поместите полученное число в первый блок. Затем подсчитайте в селекторе количество классов и псевдоклассов и поместите это число во второй блок. И, наконец, подсчитайте имена элементов и поместите это число в третий блок.

Специфичность сравнивается от блока к блоку. Первый блок, которое не связан, определяет, какой селектор будет преобладать. Рассмотрим простой пример двух конфликтующих правил для элемента `h1`:

```
h1 { color: red; } [0] [0] [1]
h1.special { color: lime; } [0] [1] [1]
```

Во втором правиле имеется селектор класса, а в первом его нет — следовательно, второй более специфичен и имеет больший вес.

Хотите что-нибудь посложнее?

```
article#main aside.sidebar:hover > h1:first-of-type [1] [3] [3]
.x.x.x.x.x.x.x.x a:link [0] [8] [1]
```

Второй селектор предназначен для ссылки в элементе с помощью строки имен классов (представленных как `«.x»`). Но первый селектор имеет ID (`#main`) и поэтому более специфичен.

Возможно, надо бы привести здесь полный расчет специфичности, но в большинстве случаев вы определите, какой селектор более специфичен, следуя приведенным общим рекомендациям.

Можно применять специфичность стратегически, что упростит таблицы стилей и минимизирует разметку. Например, можно задать стиль для элемента (в приведенном примере `p`), а затем, при необходимости, переопределить его, применяя более специфичные селекторы:

<code>p { line-height: 1.2em; }</code>	[0] [0] [1]
<code>blockquote p { line-height: 1em; }</code>	[0] [0] [2]
<code>p.intro { line-height: 2em; }</code>	[0] [1] [1]

В этом примере элементы `p` в сочетании с `blockquote` имеют меньшую высоту строки по сравнению с обычными абзацами. Однако все абзацы, помеченные как `class intro`, будут иметь высоту строки `2em`, даже если она сочетается с `blockquote`, поскольку селекторы классов более специфичны.

Понимание концепций наследования и специфичности имеет решающее значение для освоения CSS, и о специфичности можно было бы еще много чего сказать, но нам надо двигаться дальше, а во *врезке «Подробнее о специфичности»* вы найдете ссылки на полезные источники, с которыми я рекомендую вам ознакомиться.

ПОДРОБНЕЕ О СПЕЦИФИЧНОСТИ

Сведения о специфичности, приведенные в этой главе, достаточны для начала работы, но по мере приобретения определенного опыта веб-дизайна ваши таблицы стилей усложняются, поэтому вам потребуется более глубокое понимание ее механики.

Техническое объяснение расчета специфичности содержится в спецификации модуля CSS Selectors уровня 4: www.w3.org/TR/selectors4/#specificity.

Эрик Мейер (Eric Meyer) приводит подробное и более понятное описание этой системы в своей книге «*Selectors, Specificity, and the Cascade: Applying CSS to Documents*» (издательство O'Reilly). Эти материалы также включены в его книгу, написанную в соавторстве с Эстель Вейл (Estelle Weyl), «*CSS: The Definitive Guide*», 4-е издание (издательство O'Reilly).

Если вы ищете помощь в Интернете, могу порекомендовать вам статью Виталия Фридмана (Vitaly Friedman) «*CSS Specificity: Things You Should Know*» (coding.smashingmagazine.com/2007/07/27/css-specificitythings-you-should-know/). Работе более десяти лет, но изложенные в ней концепции до сих пор актуальны.

По большинству тем веб-дизайна исчерпывающие объяснения можно найти на сайте MDN Web Docs: developer.mozilla.org/en-US/docs/Web/CSS/Specificity.

УНИВЕРСАЛЬНЫЙ СЕЛЕКТОР

Универсальный селектор элемента (*) соответствует любому элементу — например, подстановочному знаку в языках программирования. Следующее правило стиля:

```
* { border: 1px solid gray; }
```

отображает серую рамку в один пиксель вокруг каждого элемента в документе.

- ▶ Подобный подход полезно применять и в качестве контекстного селектора, как показано в следующем примере, где выбираются все элементы из раздела «intro»:

```
#intro * { color: gray; }
```

Обратите внимание, что здесь каждый элемент выбирается с помощью универсального селектора, включая и те, которые вы, возможно, не собирались размечать стилем. При этом некоторые стили могут испортить элементы управления вашей формы, поэтому, если страница содержит входные данные формы, желательно избегать универсального селектора.

А мы вернемся к нашему меню. Страница «Black Goose Bistro» тщательно семантически размечена, поэтому имеется большое число вариантов для выбора специфичных элементов. Давайте опробуем селекторы, с которыми мы только что познакомились, выполнив *упражнение 12.5*.

УПРАЖНЕНИЕ 12.5. ПРИМЕНЕНИЕ СЕЛЕКТОРОВ

Добавим к нашему меню несколько правил стиля, применяя в сочетании со свойствами шрифта `font` и цвета `color` селекторы потомков, ID и классов.

1. Мне представляется необходимым добавить привлекающего внимания цвета надписям «new item!», расположенными рядом с некоторыми пунктами меню. Они помечены как `strong`, поэтому к элементу `strong` мы и применим свойство `color`. Добавьте это правило во встроенную таблицу стилей, сохраните файл и перезагрузите его в браузере:

```
strong {  
    font-style: italic;  
    color: tomato;  
}
```

Отлично, но теперь помеченная элементом `strong` надпись «Very spicy» в описании также приобрела «томатный» цвет, а это нежелательно. Решение состоит в применении контекстного селектора, нацеленного именно на элементы `strong`, которые появляются в элементах `dt`. Удалите из правила для `strong` только что добавленное объявление цвета `color` и создайте новое правило, предназначенное только для элементов `strong` в терминах списка определений:

```
dt strong { color: tomato; }
```

2. Взгляните на исходный код документа, и вы увидите, что его содержимое разделено на три уникальных элемента `div`: `info`, `appetizers` и `entrees`. Воспользуемся этим в наших интересах, когда речь идет о стиле. Пока выполним простое действие — применим «бирюзовый» цвет (`teal`) к тексту в элементах `div` с помощью ID «`info`». Поскольку цвет наследуется, свойство нужно применить только к `div`, а оно передастся вниз — к элементам `h1` и `p`:

```
#info { color: teal; }
```

3. Давайте развлечемся и пометим абзац в разделе «`info`» курсивом таким образом, чтобы это не повлияло на другие абзацы страницы. Здесь также уместно

применение контекстного селектора. Следующее правило выбирает абзацы, содержащиеся только в разделе `info` документа:

```
#info p { font-style: italic; }
```

4. Желательно уделить особое внимание и всем ценам в меню. К счастью, все они размечены элементами `span`:

```
<span class="price">$3.95</span>
```

Осталось написать правило, используя селектор класса, что позволит изменить шрифт на Georgia или какой-либо другой шрифт с засечками, выделить цены курсивом и вернуть им серый цвет:

```
.price {
    font-family: Georgia, serif;
    font-style: italic;
    color: gray;
}
```

5. Аналогично в разделе «`info`» элемента `div` можно изменить внешний вид диапазонов, помеченных как принадлежащие классу «`label`», чтобы выделялись метки:

```
.label {
    font-weight: bold;
    font-variant: small-caps;
    font-style: normal;
}
```

6. Наконец, в нижней части страницы имеется предупреждение, которое желательно уменьшить и пометить красным цветом. Присвоим ему класс «`warning`», и тогда его можно будет применить в качестве селектора для определения цели при стайлинге только этого абзаца. Заодно применим тот же стиль и к элементу `sup` (звездочка в сноске), также присутствующему на странице чуть выше. Обратите внимание, что используется сгруппированный селектор, поэтому отдельные правила писать не требуется:

```
p.warning, sup {
    font-size: small;
    color: red;
}
```

На рис. ЦВ-12.11 показаны результаты этих изменений. Полагаю, вы получили некоторое представление о присвоении цветности и оформлении специальных типографских процедур.

НАСТРОЙКИ ТЕКСТОВЫХ СТРОК

Следующая часть текстовых свойств связана с обработкой целых строк текста, а не отдельных его символов. Эти свойства позволяют веб-дизайнерам форматировать веб-текст с отступами, дополнительными интервалами между строками (лидингом) и различными горизонтальными выравниваниями, аналогично тому, как текст готовится для печати.

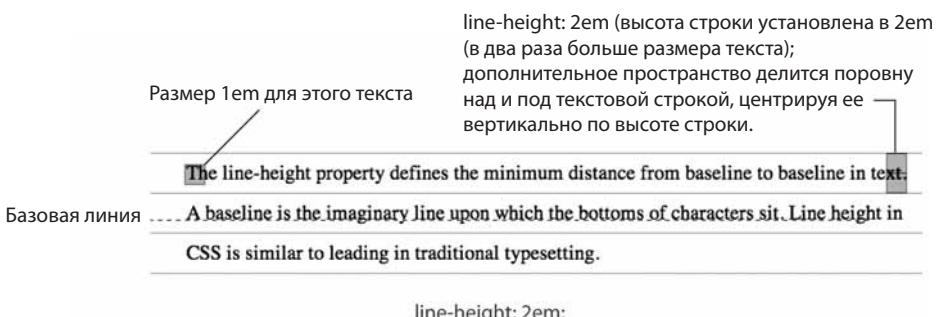
Свойство *line-height* (высота строки)

`line-height`

- **Значения:** `number` | `length measurement` | `percentage` | `normal`.
- **По умолчанию** — `normal`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Свойство `line-height` определяет минимальное расстояние между базовыми линиями строк в тексте. Ранее это свойство входило в сокращенное свойство `font`. Свойство `line-height` задает «минимальное» расстояние, поскольку при размещении в строке высокого изображения или больших символов высота этой строки соответствующим образом увеличивается.

Базовая линия — это воображаемая линия, на которой находятся символы. Установка высоты строки в CSS аналогична добавлению лидинга в традиционной верстке, однако вместо добавляемого между строками интервала дополнительное пространство добавляется над и под текстом. В результате высота строки определяет высоту *линейного блока*, где текстовая строка центрирована вертикально (рис. 12.12).



`line-height: 2em;`

Рис. 12.12. Текстовые строки центрированы вертикально по высоте строки

Следующие примеры демонстрируют три различных способа задания высоты строки, в два раза большей, чем высота размера шрифта:

```
p { line-height: 2; }
p { line-height: 2em; }
p { line-height: 200%; }
```

Если число указывается отдельно, как показано в первом примере, оно воспринимается в качестве коэффициента масштабирования, который для вычисления значения `line-height` затем умножается на текущий размер шрифта.

Высота линии может быть также указана в одной из единиц измерения CSS. Значения в `ems` и процентные значения основываются на текущем размере шрифта элемента. Так, в трех примерах, показанных на рис. 12.12, при размере шрифта, равном 16 пикселям, вычисленная высота строки равна 32 пикселям.

Разница между применением коэффициента масштабирования (числовое значение) и относительного значения (em или %) заключается в наследовании. Если установить высоту строки с масштабным коэффициентом для всего документа в элементе body, его потомки наследуют этот множитель. Так, если коэффициент масштабирования для body равен 2, заголовок в 24 пикселя получит высоту 48 пикселов.

Если же установить значение свойства line-height в элементе body, применяя em или проценты, его потомки наследуют вычисленный размер на основе размера шрифта body. Например, если для высоты строки в элементе body установлено значение 1em (16 пикселов), заголовок размером 24 пикселя наследует рассчитанную высоту строки в 16 пикселов, а не значение 1em. Вероятно, это не тот эффект, на который вы рассчитывали... Поэтому в рассматриваемом случае лучше использовать числовые значения, ведущие к интуитивно понятному варианту.

Свойство **text-indent** (отступы)

Свойство text-indent создает отступ первой строки текста на указанную величину:

text-indent

- **Значения:** величина | процентное соотношение.
- **По умолчанию** — 0.
- **Применение** — к блочным контейнерам.
- **Наследование** — да.

Величину отступа для свойства text-indent можно задать в численном выражении или в процентном значении. Рассмотрим несколько примеров (рис. 12.13):

```
p#1 { text-indent: 2em; }
p#2 { text-indent: 25%; }
p#3 { text-indent: -35px; }
```

2em

 Paragraph 1. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.

25%

 Paragraph 2. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.

-35px

 Paragraph 3. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.

Рис. 12.13. Примеры использования свойства text-indent

ОРГАНИЗАЦИЯ ОТСТУПА ДЛЯ ВСЕГО ТЕКСТОВОГО БЛОКА

При обращении к свойству `text-indent` с отступом отображается только первая строка блока. Если необходимо сформировать пространство вдоль всей стороны текстового блока, примените одно из свойств `margin` (поле) или `padding` (внутренний отступ). Дизайнеры обычно указывают отступы и поля вместе, но для иллюстрации того, каким образом CSS их трактует, поля рассматриваются как часть блочной модели в главе 14.

Значения в процентах рассчитываются на основе ширины *родительского* элемента и передаются дочерним элементам в виде процентных значений (но не рассчитанных значений). Таким образом, если элемент `div` содержит свойство `text-indent`, равное 10%, таковыми будут все его потомки.

В третьем примере обратите внимание, что указано отрицательное значение, и это приводит к весьма любопытному эффекту — первая строка текста будет выступать левее от левого края текста (это называется *висячим отступом*).

ЕЩЕ О ВИСЯЧЕМ ОТСТУПЕ

Если применяется висячий отступ, убедитесь, что к элементу также применено выравнивание влево. В противном случае «висячий» текст может исчезнуть за левым краем окна браузера.

Свойство `text-align` (горизонтальное выравнивание текста)

Текст на веб-страницах можно выравнивать с помощью свойства `text-align` так же, как это делается в компьютерной программе обработки текстов или публикаций:

`text-align`

- **Значения:** `left` | `right` | `center` | `justify` | `start` | `end`.
- **По умолчанию** — `start`.
- **Применение** — к блочным контейнерам.
- **Наследование** — да.

Свойство `text-align` весьма просто в использовании:

- `text-align: left` — выравнивает текст по левому полю (границе);
- `text-align: right` — выравнивает текст по правому полю;

СВОЙСТВА `TEXT-ALIGN-LAST` И `TEXT-JUSTIFY`

Модуль CSS *Text Module* уровня 3 также определяет два новых свойства, связанных с выравниванием текста: `text-align-last` (для выравнивания последней строки текста) и `text-justify` (для более точного управления тем, каким образом для выравнивания по ширине в выравниваемый текст вставляются пробелы).

- `text-align: center` — центрирует текст в текстовом блоке;
- `text-align: justify` — выравнивает текст как по левому, так и по правому полям (выравнивание по ширине).

Результаты применения приведенных здесь значений `text-align` из спецификации CSS2.1 показаны на рис. 12.14.

В модуле CSS Text Module уровня 3 добавлены значения `start` и `end`, которые определяют сторону строки, по которой проводится выравнивание текста. Подобный подход применяется к тем языкам, написание символов в которых выполняется по вертикали и справа налево. Для тех языков, для которых чтение выполняется слева направо, `start` соответствует `left`.

`text-align:left;`

Paragraph 1. The `text-align` property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page. The resulting text behavior of the various values should be fairly intuitive.

`text-align:right;`

Paragraph 2. The `text-align` property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page. The resulting text behavior of the various values should be fairly intuitive.

`text-align:center;`

Paragraph 3. The `text-align` property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page. The resulting text behavior of the various values should be fairly intuitive.

`text-align:justify;`

Paragraph 4. The `text-align` property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page. The resulting text behavior of the various values should be fairly intuitive.

Рис. 12.14. Примеры применения свойства `text-align` из спецификации CSS2.1

Отличная новость: осталось рассмотреть лишь пять текстовых свойств! А затем мы опробуем некоторые из них, преобразуя меню бистро «Black Goose Bistro».

Свойство `text-decoration` (подчеркивания и другие украшательства)

Если нужно подчеркнуть, надчеркнуть или перечеркнуть строку или отключить подчеркивание ссылок, следует воспользоваться свойством `text-decoration`:

`text-decoration`

- **Значения:** `none` | `underline` | `overline` | `line-through` | `blink`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — нет, но поскольку строки проходят через дочерние элементы, они могут иметь такой вид, как будто также «украшены».

Значения `text-decoration` имеют интуитивно понятный характер (рис. 12.15):

- `underline` — подчеркивает выбранный элемент;
- `overline` — рисует линию над выбранным текстом;
- `line-through` — перечеркивает линией выбранный текст.

I've got laser eyes.

text-decoration: underline;

I've got laser eyes.

text-decoration: overline;

~~I've got laser eyes.~~

text-decoration: line-through;

Рис. 12.15. Примеры применения значений свойства `text-decoration`

Наиболее распространенное применение свойства `text-decoration` состоит в отключении подчеркивания, которое автоматически отображается под текстом ссылки:

```
a { text-decoration: none; }
```

Относительно применения свойства `text-decoration` имеется несколько предсторожностей:

- прежде всего, если необходимо устраниить подчеркивание ссылок, убедитесь, что имеются другие возможности компенсировать это — такие, как цвет и «жирность»;
- с другой стороны, подчеркивание несет сильный визуальный сигнал, вынуждая «щелкнуть здесь», поэтому подчеркивание текста, не являющегося ссылкой, может вводить пользователя в заблуждение. Подумайте, не будет ли курсив приемлемой альтернативой;
- наконец, необязательно, чтобы текст мерцал (значение `blink`). Производители браузеров согласились с этим и отказались от поддержки мерцания текста. Кстати, браузер IE никогда и не поддерживал это свойство.

ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА УКРАШЕНИЯ ТЕКСТА

Текстовый модуль CSS3 *Text Module* содержит дополнительные средства украшения текста, среди которых: `text-decoration-line`, `text-decoration-color`, `text-decoration-style`, `text-decoration-skip` и `text-underline-position`. Ни одна из версий IE или Edge не поддерживает эти свойства, но, за исключением `-skip`, они поддерживаются в других современных браузерах. Обратитесь к [CanIUse.com](#) для более близкого знакомства со спецификациями.

Свойство `text-transform` (изменение вида заглавных букв)

Я хорошо помню тот момент, когда в настольных издательских программах появилась функция, которая позволяла на лету изменять вид заглавных (прописных) букв текста (да-да, можете считать меня такой старой). Это давало возможность увидеть, как будут выглядеть в тексте заглавные буквы без необходимости распечатки страниц. CSS также обеспечивает такую функцию с помощью свойства `text-transform`:

text-transform

- **Значения:** none | capitalize | lowercase | uppercase | full-width.
- **По умолчанию** — none.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Применение к текстовому элементу свойства `text-transform` изменяет отображение заглавных букв независимо от того, как они набирались при вводе с клавиатуры. Это свойство может иметь следующие значения (рис. 12.16):

- none — так, как введено изначально;
- capitalize — делает заглавной первую букву каждого слова;
- lowercase — все буквы становятся строчными;
- uppercase — делает заглавными первые буквы всех слов;
- full-width — выбирает из шрифта «полноразмерную» версию символа, если он существует (не всеми браузерами поддерживается).

`text-transform: none;`
(как введено изначально) **And I know what you're thinking.**

`text-transform: capitalize;` **And I Know What You're Thinking.**

`text-transform: lowercase;` **and i know what you're thinking.**

`text-transform: uppercase;` **AND I KNOW WHAT YOU'RE THINKING.**

Рис. 12.16. Свойство `text-transform` изменяет заглавные буквы независимо от того, как они введены изначально

Свойства ***letter-spacing*** и ***word-spacing*** (вставка пробелов)

Следующие два текстовых свойства применяются для вставки пробелов между буквами (`letter-spacing`) или словами (`word-spacing`):

letter-spacing

- **Значения:** величина | normal.
- **По умолчанию** — normal.
- **Применение** — ко всем элементам.
- **Наследование** — да.

word-spacing

- **Значения:** величина | normal.
- **По умолчанию** — normal.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Свойства `letter-spacing` и `word-spacing` выполняют действия, вытекающие из их названия: добавляют пробелы между буквами в слове или словами в строке, соответственно.

На рис. 12.17 показаны результаты применения свойства `letter-spacing` — для вставки пробелов между буквами (*вверху*) и свойства `word-spacing` — для вставки пробелов между словами (*внизу*) применительно к контенту следующего простого абзаца:

```
<p>Black Goose Bistro Summer Menu</p>
```

Black Goose Bistro Summer Menu

`letter-spacing: 8px;`

Black Goose Bistro Summer Menu

`word-spacing: 1.5em;`

Рис. 12.17. Действие свойств `letter-spacing` (*вверху*) и `word-spacing` (*внизу*)

Следует отметить, что при указании единиц измерений ем вычисленный размер передается дочерним элементам, даже если они имеют меньший по сравнению с родительским размер шрифта.

При выполнении упражнения 12.6, расположенного в этой главе далее, мы вернемся к меню «Black Goose Bistro» и воспользуемся свойством `letter-spacing` для заголовков уровня `h2`.

Свойство `text-shadow` (добавление к тексту «тени»)

Свойство `text-shadow` добавляет к тексту «тень», эффект которой заключается в том, что текст как бы «всплывает» над страницей. Поскольку в наши дни популярность приобрел плоский дизайн, тени применяются редко, однако они могут послужить полезным визуальным инструментом, особенно если текст размещен на узорном фоне или на фоне фотографии.

Текстовые тени располагаются позади текста, но перед фоном и границами, если таковые имеются. Свойство `text-shadow` поддерживается всеми современными браузерами, кроме браузеров Internet Explorer версии 9 и более ранних:

`text-shadow`

- **Значения:** '*смещение по горизонтали*' '*смещение по вертикали*' '*радиус размытия*' '*цвет*' | `none`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Значением для свойства `text-shadow` служат две или три величины: горизонтальное смещение, вертикальное смещение и необязательное значение радиуса размытия, а также — цвет:

```
h1 {  
    color: darkgreen;  
    text-shadow: .2em .2em silver;  
}  
h1 {  
    color: darkgreen;  
    text-shadow: -.3em -.3em silver;  
}
```

Первое значение в свойстве — это смещение по горизонтали, размещающее тень справа от текста, а второе значение — смещение по вертикали, которое перемещает тень вниз на заданную величину (рис. 12.18, *вверху*). Отрицательные значения в свойстве приводят к расположению тени *слева* и *вверху* по отношению к тексту (рис. 12.18, *внизу*). Объявление заканчивается заданием цвета тени (`silver`, серебро). Если цвет не указан, тень остается в исходном цвете текста.

Рассмотрев, к чему приводят первые два значения свойства `text-shadow`, заметим, что резкая и четкая тень выглядит не очень хорошо и немного мешает восприятию текста (см. рис. 12.18). Хорошо бы ее немного размыть... С этим нам поможет величина радиуса размытия, причем значение радиуса размытия, равное 0, вовсе не дает размытия, которое усиливается при задании более высоких значений радиуса (рис. 12.19). Обычно для получения желаемого эффекта приходится со значениями свойства `text-shadow` немного поэкспериментировать.

The Jenville Show

`text-shadow: .2em .2em silver;`

The Jenville Show

`text-shadow: -.3em -.3em silver;`

Рис. 12.18. Четкая тень, падающая от текста вниз и вправо (*вверху*) и вверх и влево (*внизу*)

The Jenville Show

`text-shadow: .2em .2em .1em silver;`

The Jenville Show

`text-shadow: .2em .2em .3em silver;`

Рис. 12.19. Добавление радиуса размытия к падающей от текста тени: величина радиуса размытия 3em (*внизу*) дает более сильное размытие, чем величина 1em (*вверху*)

К одному элементу можно применить и несколько текстовых теней. Изменяя положение тени и степень размытия, вы сможете придать тексту такой вид, как будто на него направлено несколько источников света.

Смело экспериментируйте с текстовыми тенями, но не переусердствуйте. Тени иногда затрудняют чтение текста, а наличие теней способно уменьшить скорость отображения страницы (операции прокрутки, взаимодействия с мышью и т. д.), что может создать проблемы для браузеров мобильных устройств, не располагающих большой вычислительной мощностью. Кроме того, не сделайте так, чтобы вашему тексту требовалась тень, чтобы стать видимым, — тогда пользователи браузеров, которые не поддерживают этот эффект, вообще ничего не увидят. Мой вам совет: применяйте тени как дополнение, но не столь критичное, чтобы без них нельзя было бы обойтись.

ПРОЧИЕ ТЕКСТОВЫЕ СВОЙСТВА

В целях экономии места, а также для сохранения в книге уровня изложения «для начинающих» приведенные далее свойства не получили исчерпывающего описания, но упомянуть о них стоит.

Для ознакомления с другими дополнительными связанными с текстом свойствами, находящимися еще в разработке, обратитесь к следующим модулям CSS Text Modules:

- CSS Text Module Level 3: www.w3.org/TR/css-text-3;
- CSS Text Decoration Module Level 3: www.w3.org/TR/css-text-decor-3;
- Модуль CSS Text Module уровня 4 (в рамках рабочего проекта) и считается экспериментальным: www.w3.org/TR/css-text-4.

Каждое свойство из следующего списка помечено уровнем CSS, на котором оно представлено:

- свойство `white-space` (CSS2) — определяет, каким образом обрабатываются в макете пробелы, содержащиеся в исходном элементе. Например, значение сохраняет пробелы и возвращает найденные в источнике результаты, аналогично HTML-элементу `pre`;
- свойство `vertical-align` (CSS2) — определяет выравнивание по вертикали базовой линии строчного элемента относительно базовой линии окружающего текста. Также применяется для установки вертикального выравнивания контента в ячейке таблицы (`td`);
- свойства `word-break` и `line-break` (CSS3) — влияют на результат переноса текста в словах и строках соответственно при использовании разных языков, включая восточноазиатские (китайский, японский, корейский);
- свойство `text-justify` (CSS3) — определяет способ добавления пробела внутри и между словами, если свойству `text-align` элемента присвоено `justify`;
- свойство `text-align-last` (CSS3) — указывает, каким образом необходимо выравнивать последнюю строку блока текста, если свойству `text-align` элемента присвоено значение `justify`. Например, иногда желательно, чтобы последняя строка выровненного текста выравнивалась по левому краю, что позволяет избежать выравнивания по всей ширине текста слишком короткой строки;

- свойство `tab-size` (CSS3) — задает длину символа табуляции (код Unicode 0009) с использованием количества символов или величины длины;
- свойство `hyphens` (CSS3) — управляет переносом текста. Значение свойства `manual` означает, что перенос слов происходит, только если в них изначально добавлен дефис. Значение свойства `auto` предоставляет управление браузеру, а значение `none` полностью отключает переносы;
- свойство `overflow-wrap` (CSS3) — определяет, разрешено ли браузерам разбивать слова, чтобы текст мог поместиться в ограниченной по размеру рамке;
- свойство `hanging-punctuation` (CSS3) — определяет, может ли знак препинания выходить за пределы поля строки элемента в начале или в конце строки. «Висячая» пунктуация делает поля более аккуратными;

Следующие свойства содержатся в спецификации, но их не следует применять. Вместо них применяйте HTML-атрибут `dir`:

- свойство `direction` (CSS3) — определяет направление, в котором текст читается: слева направо (`ltr`) или справа налево (`rtl`);
- свойство `unicode-bidi` (CSS2) — относится к двунаправленным функциям Unicode. В Рекомендации говорится, что оно позволяет автору веб-страницы генерировать уровни встраивания с учетом алгоритма встраивания Unicode. Если вы не понимаете, что это значит, не озабочивайтесь. Я этого тоже не понимаю.

Выполните теперь *упражнение 12.6* — оно даст вам возможность попробовать еще несколько свойств форматирования текста, которые добавят больше блеска в меню «Black Goose Bistro».

УПРАЖНЕНИЕ 12.6. ФИНАЛЬНЫЕ ПРЕОБРАЗОВАНИЯ

Итак, добавим в меню `menu.html` несколько последних штрихов. Я рекомендую вам сохранять файл и просматривать его в браузере после выполнения каждого шага, чтобы замечать изменения и оставаться в уверенности, что вы находитесь на правильном пути. Подготовленная таблица стилей находится в папке `materials` для этой главы.

1. Во-первых, сделаем несколько глобальных изменений элемента `body`. У меня несколько изменилось мнение по поводу применяемого семейства шрифтов (`font-family`). Я полагаю, что шрифт с засечками, такой, как `Georgia`, более изящен и вполне подходит для меню бистро. Воспользуемся также свойством `line-height`, чтобы «открыть» текстовые строки, что сделает их более удобными для чтения. Добавим указанные обновления в правило стиля `body`:

```
body {  
    font-family: Georgia, serif;  
    font-size: small;  
    line-height: 1.75em;  
}
```

2. Мне также хочется изменить раздел "info" документа. Отмените присвоение бирюзового цвета (teal color), удалив это правило целиком. Затем окрасьте раздел h1 в оливковый цвет, а абзац в заголовке сделайте серым. Добавим указанные обновления цветов к существующим правилам объявления цвета:

```
#info { color: teal; } /* delete */
h1 {
    font: bold 1.5em "Marko One", Georgia, serif;
    color: olive;
}
#info p {
    font-style: italic;

    color: gray;
}
```

3. Затем, в подражание меню модного ресторана, центрируйте несколько ключевых элементов на странице с помощью свойства text-align. Напишите для этого правила с групповым селектором для центрирования заголовков и раздела "info":

```
h1, h2, #info {
    text-align: center;}
```

4. Желательно сделать более броскими заголовки уровня h2 «Appetizer» и «Main Courses». Заменим для них крупный полужирный шрифт на шрифт полностью из заглавных букв, добавим между символами дополнительные пробелы, а также окрасим их в оливковый цвет, что привлечет к этим заголовкам повышенное внимание. Добавьте в документ новое правило для элементов h2, учитывающее эти изменения:

```
h2 {
    font-size: 1em;
    text-transform: uppercase;
    letter-spacing: .5em;
    color: olive;}
```

5. Еще чуть-чуть — и цель будет достигнута. Но давайте добавим еще несколько настроек в абзацы, следующие сразу после заголовков h2, — центрируем их и выделим курсивом:

```
h2 + p {
    text-align: center;
    font-style: italic;}
```

Обратите внимание, что для выбора любого абзаца, следующего за h2, применяется next-sibling selector (селектор «последующего брата»): h2 + p.

6. Исправьте в именах пунктов меню (в элементах dt) цвет на более мягкий. Здесь я выбрали цвет sienna (охра) в модуле цвета CSS3. Обратите внимание, что элементы strong в элементах dt остаются «томатно»-красными, поскольку цвет, примененный к элементам strong, переопределяет цвет, унаследованный от их родителей:

```
dt {
    font-weight: bold;
    color: sienna;}
```

7. Наконец, для получения ударного эффекта добавьте тень к заголовку `h1`. Вы можете поиграть со значениями свойства `text-shadow`, чтобы лучше разобраться, как оно функционирует. В общем-то для нашего меню белый фон не вполне уместен, но, если бы у нас здесь присутствовало узорчатое фоновое изображение, тень могла бы создать эффект, необходимый для выделения текста. Обратите внимание, насколько невелики значения тени, но даже эти небольшие значения приводят к заметному эффекту!

```
h1 {  
    font: bold 1.5em "Marko One", Georgia, serif;  
    color: olive;  
    text-shadow: .05em .05em .1em lightslategray;}
```

Вот, собственно и все! На рис. ЦВ-12.20 показан результирующий вид нашего меню — вполне заметны улучшения по сравнению с исходной версией, не оформленной стилями, а ведь мы применили всего лишь только свойства текста и цвета. Обратите внимание, что не подвергся изменениям ни один символ разметки документа. Вот в этом и заключена вся прелест формирования стиля отдельно от структуры.

ИЗМЕНЕНИЕ МАРКИРОВАННОГО И НУМЕРОВАННОГО СПИСКОВ

В завершение главы о свойствах текста рассмотрим несколько настроек, с помощью которых можно вносить изменения в маркованные и нумерованные списки. Как известно, браузеры автоматически вставляют маркеры перед элементами неупорядоченных списков и номера перед элементами в упорядоченных списках (*маркеры списка*). Чаще всего вид этих маркеров определяется браузером. Тем не менее CSS предоставляет несколько свойств, которые позволяют авторам выбирать тип и положение маркеров или полностью отключать их.

Свойство `list-style-type` (выбор маркера списка)

Применение свойства `list-style-type` к элементу `ul`, `ol` или `li` означает выбор типа маркера, отображаемого перед каждым пунктом списка.

`list-style-type`

- **Значения:** `none` | `disc` | `circle` | `square` | `decimal` | `decimal-leading-zero` | `lower-alpha` | `upper-alpha` | `lower-latin` | `upper-latin` | `lower-roman` | `upper-roman` | `lower-greek`.
- **По умолчанию** — `disc`.
- **Применение** — к `ul`, `ol` и `li` (или к элементам, чьим отображаемым значением является элемент списка).
- **Наследование** — да.

О ТАБЛИЦАХ СТИЛЕЙ CSS3

В этом разделе представлены типы свойств `list-style` для CSS2.1, которые хорошо поддерживаются в современных браузерах. Таблицы стилей CSS3 расширяют упомянутые здесь функции маркеров, описывают методы, позволяющие авторам веб-страниц определять собственные стили списков, разрешают использовать нумерацию на многих языках (www.w3.org/TR/css3-lists/).

Свойство `list-style-type` разработчики часто применяют со значением `none` — для удаления из списков маркеров или цифр. Подобный подход удобен при использовании разметки списка как основы для горизонтального меню навигации или для названий полей в веб-формах. При этом надоедливые маркеры устраняются, а семантика разметки сохраняется.

Значения `disc` (точка), `circle` (кружок) и `square` (квадратик) генерируют формы маркеров так, как это делали когда-то первые браузеры (рис. 12.21). К сожалению, отсутствует способ изменения внешнего вида (размера, цвета и т. д.) уже сгенерированных маркеров, поэтому мы здесь остановимся на заданных по умолчанию настройках браузера.

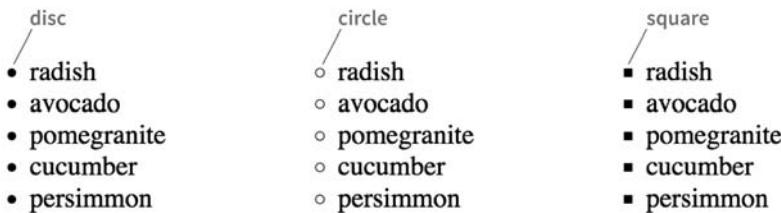


Рис. 12.21. Значения `list-style-type: disc, circle` и `square`

Оставшиеся значения (см. табл. 12.1) применяются при использовании упорядоченных списков, определяя различные стили для нумерации и букв.

ПРАВИЛО @COUNTER-STYLE CSS3

В таблицах стилей CSS3 появилось правило `@counter-style`, поддерживающее типы маркеров `box`, `check`, `diamond` и `dash`, а также возможность при отсутствии предопределенных указывать собственные маркеры. Подробности содержатся в спецификации.

Таблица 12.1. Буквы и цифры, применяемые в списках (CSS2.1)

Значение	Система нумерации
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... (так же, как в случае применения свойства lower-alpha)
upper-latin	A, B, C, D, E... (так же, как в случае применения свойства upper-alpha)
lower-roman	I, ii, iii, iv, v...
upper-roman	I, II, III, IV, V...
lower-greek	α, β, γ, δ, ε...

Свойство *list-style-position* (положение маркера)

По умолчанию маркер находится за пределами области контента для элемента списка и отображается как висячий отступ. Свойство *list-style-position* позволяет перетащить маркер внутрь области контента — тогда он окажется в контенте списка:

list-style-position

- **Значения:** `inside` | `outside` | `hanging`.
- **По умолчанию** — `outside`.
- **Применение** — к `ul`, `ol` и `li` (или к элементам, чьим отображаемым значением является элемент списка).
- **Наследование** — да.

Чтобы вам было понятнее, как приведенные в следующем примере значения свойства *list-style-position* изменяют положение маркеров списков, на рис. 12.22 я подсветила светло-зеленым цветом отображение границ областей их контента:

```
li {background-color: #F99;}  
ul#outside {list-style-position: outside;}  
ul#inside {list-style-position: inside;}
```

outside

- **Radish.** Praesent in lacinia risus. Morbi urna ipsum, efficitur id erat pellentesque, tincidunt commodo sem. Phasellus est velit, porttitor vel dignissim vitae, commodo ut urna.
- **Avocado.** Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur lacinia accumsan est, ut malesuada lorem consectetur eu.
- **Pomegranite.** Nam euismod a ligula ac bibendum. Aenean ac justo eget lorem dapibus aliquet. Vestibulum vitae luctus orci, id tincidunt nunc. In a mauris odio. Duis convallis enim nunc.

inside

- **Radish.** Praesent in lacinia risus. Morbi urna ipsum, efficitur id erat pellentesque, tincidunt commodo sem. Phasellus est velit, porttitor vel dignissim vitae, commodo ut urna.
- **Avocado.** Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur lacinia accumsan est, ut malesuada lorem consectetur eu.
- **Pomegranite.** Nam euismod a ligula ac bibendum. Aenean ac justo eget lorem dapibus aliquet. Vestibulum vitae luctus orci, id tincidunt nunc. In a mauris odio. Duis convallis enim nunc.

Рис. 12.22. Значения *list-style-position: outside* (вверху) и *inside* (внизу)

Как можно видеть, если положению присвоено значение `outside` (*вверху*), маркеры «выходят» за пределы области контента. Когда выбрано значение `inside` (*внизу*), маркеры находятся в области контента.

В CSS3 для свойства `list-style-position` появилось значение `hanging`. Эффект от применения этого свойства напоминает эффект применения свойства `inside`, но при этом маркеры отображаются снаружи и прымкают к левому краю подсвеченной области.

РОЛЬ ЭЛЕМЕНТА СПИСКА ПРИ ОТОБРАЖЕНИИ

Возможно, вы заметили, что свойства стиля списка применяются к «элементам, отображаемым значением которых является `list-item`». Спецификация CSS2.1 позволяет каждому элементу служить элементом списка, если присвоить свойству `display` значение `list-item`. Это свойство может применяться к любому HTML-элементу или элементам из других XML-языков. Например, можно автоматически промаркировать или пронумеровать ряд абзацев, присваивая значение `list-item` свойству `display` элементов абзаца (`p`), как показано в следующем примере:

```
p.lettered {
    display: list-item;
    list-style-type: upper-alpha;
}
```

Свойство `list-style-image` (создание собственных маркеров)

В качестве маркера можно использовать и какое-либо изображение. Для этого нужно обратиться к свойству `list-style-image`:

`list-style-image`

- **Значения:** `url(местоположение)` | `none`.
- **По умолчанию** — `none`.
- **Применение** — к `ul`, `ol` и `li` (или к элементам, чьим отображаемым значением является элемент списка).
- **Наследование** — да.

Значением свойства `list-style-image` служит интернет-адрес (URL) изображения, которое требуется применить в качестве маркера. А свойству `list-style-type` присваивается значение `disc` в качестве резервной копии, если изображение не появляется на экране либо свойство `list-style-image` не поддерживается браузером или другим пользовательским агентом (рис. 12.23):

```
ul {
    list-style-type: disc;
    list-style-image: url(/images/rainbow.gif);
    list-style-position: outside;
}
```

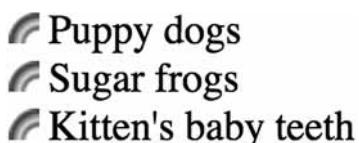
- 
- Puppy dogs
 - Sugar frogs
 - Kitten's baby teeth

Рис. 12.23. Применение изображения в качестве маркера

Ура! Подошла к завершению эта длинная глава! Сначала мы рассмотрели свойства, применяемые для определения шрифтов и форм символов, затем сделали обзор всех настроек и эффектов уровня текста. Мы также изучили возможности, связанные с применением селекторов потомков, селекторов ID (идентификаторов) и селекторов классов и немного подробнее разобрались со специфичностью. Мы мознакомились со свойствами, обеспечивающими добавление стилей в списки. Конечно, не все эти свойства останутся в памяти (но многие осядут там при практическом применении). А теперь попробуйте ответить на контрольные вопросы.

СВОЙСТВО *LIST-STYLE*

Существует свойство *list-style*, которое в любом порядке объединяет значения типа, позиции и изображения маркера списка. Например:

```
ul {  
    list-style:  
    url(/images/rainbow.gif)  
    disc outside;  
}
```

Используя такое свойство, не забывайте об особенностях применения сокращенных свойств, будьте внимательны и не переопределите случайно свойства стиля списка, установленные ранее в таблице стилей.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Посмотрим, насколько хорошо вы освоили представленные в этой главе свойства шрифта и селекторы. Ответы на эти вопросы вы сможете найти в *приложении 1*.

1. Сопоставьте свойства стиля с текстовыми примерами, представленными на рис. 12.24.
 - a. _____ {font-size: 1.5em;}
 - b. _____ {text-transform: capitalize;}
 - c. _____ {text-align: right;}
 - d. _____ {font-family: Verdana; font-size: 1.5em;}
 - e. _____ {letter-spacing: 3px;}
 - f. _____ {font: bold italic 1.2em Verdana;}
 - g. _____ {text-transform: uppercase;}
 - h. _____ {text-indent: 2em;}
 - i. _____ {font-variant: small-caps;}
2. А теперь вам предоставляется шанс немного попрактиковаться в написании селекторов. Используя показанную на рис. 12.25 схему, напишите правила стиля, которые окрашивают красным (`color: red;`) каждый из представленных в следующем списке элементов. Запишите селектор максимально эффективно, насколько это возможно.
 - a. Все текстовые элементы документа
 - b. Элементы `h2`

- c. Элементы `h1` и все абзацы
- d. Элементы, относящиеся к классу `special`
- e. Все элементы из раздела «intro»
- f. Элементы `strong` из раздела «main»
- g. Дополнительный бонус: только абзац, который появляется после `h2`

`Look for the good in others and they'll see the good in you.`
заданный по умолчанию шрифт и размер

- ➊ `Look For The Good In Others And They'll See The Good In You.`
- ➋ `Look for the good in others and they'll see the good in you.`
- ➌ **`Look for the good in others and they'll see the good in you.`**
- ➍ `Look for the good in others and they'll see the good in you.`
- ➎ `Look for the good in others and they'll see the good in you.`
- ➏ `LOOK FOR THE GOOD IN OTHERS AND THEY'LL SEE THE GOOD IN YOU.`
- ➐ `Look for the good in others and they'll see the good in you.`
- ➑ **`Look for the good in others and they'll see the good in you.`**

Рис. 12.24. Текстовые примеры с примененными стилями

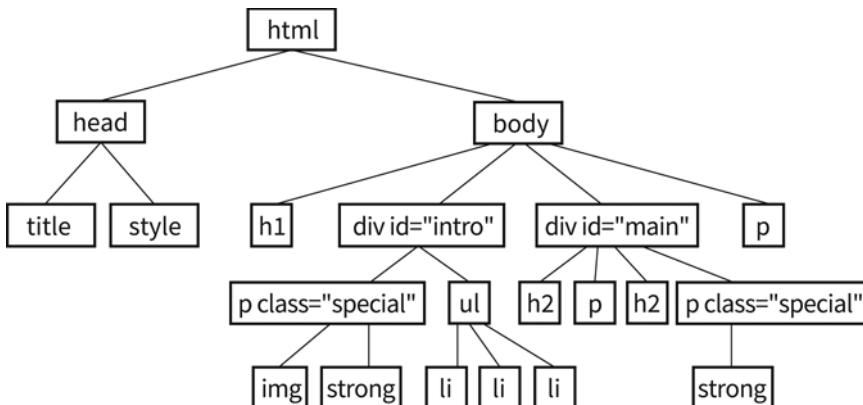


Рис. 12.25. Образец структуры документа

ОБЗОР CSS: СВОЙСТВА `FONT` И `TEXT`

В табл. 12.2 в алфавитном порядке приводится краткий обзор свойств, применяемых для форматирования текстовых элементов.

Таблица 12.2. Свойства, применяемые для форматирования текстовых элементов.

Свойство	Описание
color	Определяет цвет переднего плана (текста и границ) элемента
direction	Указывает, читается ли текст слева направо или справа налево
font	Сокращенное свойство, комбинирующее свойства шрифта
font-family	Определяет семейство шрифтов или общий шрифт
font-feature-settings	Предоставляет доступ к менее используемым свойствам OpenType
font-kerning	Управляет тем, каким образом браузеры реализуют данные кернинга (установки пробелов между символами)
font-language-override	Управляет использованием специфичных для языка глифов
font-size	Определяет размер шрифта
font-size-adjust	Соответствует х-высоте резервного шрифта для указанного шрифта
font-stretch	Выбирает сжатый, обычный или разреженный шрифт
font-style	Определяет курсив или наклонные шрифты
font-synthesis	Определяет, может ли браузер имитировать полужирный или курсивный шрифты
font-variant	Определяет шрифт капители (small-caps, маленьких заглавных букв)
font-variant-alternates	Выбирает альтернативные версии символьных глифов
font-variant-caps	Выбирает маленькие заглавные буквы и подобные альтернативы, если доступно
font-variant-east-asian	Выбирает альтернативные глифы на китайском, японском и корейском языках
font-variant-ligatures	Выбирает лигатуры для определенных пар букв, если доступно
font-variant-numeric	Выбирает альтернативные глифы числа
font-variant-position	Выбирает глифы нижних или верхних индексов
font-weight	Определяет степень «жирности» шрифта
hanging-punctuation	Указывает, может ли пунктуация «висеть» вне поля контента
hyphens	Управляет переносом текста
letter-spacing	Вставляет пробелы между буквами
line-break	Описывает правила для разрыва строк
line-height	Указывает расстояние между базовыми линиями соседних текстовых строк
list-style-image	Определяет изображение, которое используется как маркер списка

Табл. 12.2. (окончание)

Свойство	Описание
list-style-position	Размещает маркер списка внутри или снаружи области контента
list-style-type	Выбирает тип маркера для элементов списка
overflow-wrap	Указывает, может ли браузер разбивать строки, чтобы уместить текст в пределах выделенного блока
tab-size	Определяет длину символа табуляции
text-align	Указывает горизонтальное выравнивание текста
text-align-last	Определяет, как выравнивается последняя строка в выровненном тексте
text-decoration	Определяет подчеркивание, надчеркивание и перечеркивание текста
text-indent	Определяет количество отступов первой строки в блоке
text-justify	Обозначает, каким образом пространство распределяется в выровненном тексте
text-shadow	Добавляет к тексту тень
text-transform	Изменяет заглавные буквы текста при отображении
unicode-bidi	Работает с двунаправленными алгоритмами Unicode
vertical-align	Корректирует вертикальное положение встроенных элементов относительно базовой линии
white-space	Определяет, как отображаются пробелы в исходном тексте
word-break	Определяет, будут ли разрывы строк по словам
word-spacing	Вставляет пробелы между словами
word-wrap	Указывает, может ли браузер разбивать строки по словам, чтобы уместить текст в пределах выделенного блока (аналогично свойству overflow-wrap)

ГЛАВА 13

ЦВЕТА И ФОНЫ, А ТАКЖЕ ЕЩЕ О СЕЛЕКТОРАХ И ВНЕШНИХ ТАБЛИЦАХ СТИЛЕЙ

В этой главе...

- ▶ *Названия цветов в CSS*
- ▶ *Цветовые значения RGB*
- ▶ *Цвета переднего и заднего планов изображения*
- ▶ *Мозаичные фоновые изображения*
- ▶ *Градиенты цвета*
- ▶ *Селекторы псевдоклассов, псевдоэлементов и атрибутов*
- ▶ *Внешние таблицы стилей*

Если бы вы вернулись в далекий 1993-й год и стали заниматься серфингом в Интернете, это занятие показалось бы вам весьма скучным. Не слишком интересно просматривать черно-белые веб-страницы с серым фоном и черным текстом. Чуть позже появился браузер Netscape Navigator, а вместе с ним несколько HTML-атрибутов, которые обеспечивали хотя и элементарное (но столь необходимое) управление цветами шрифтов и фонами. Ну а сейчас в нашем распоряжении имеются свойства таблиц стилей, которые отправили эти допотопные атрибуты представления отдыхать.

В этой главе мы собираемся охватить много базовых тем. Я познакомлю вас со всеми свойствами, определяющими цвета и фоны. Здесь также будет пополнена ваша коллекция типов селекторов и рассказано, как создать внешнюю таблицу стилей. Однако наша первая задача — рассмотреть варианты определения цвета в CSS, включая описание природы цвета, отображаемого на компьютерных мониторах.

УКАЗАНИЕ ЦВЕТОВЫХ ЗНАЧЕНИЙ

Имеются два основных способа задания цветов в таблицах стилей: с помощью заранее определенного имени цвета, как это мы делали до сих пор:

```
color: red;
```

```
color: olive;
```

```
color: blue;
```

или, что встречается значительно чаще, — с помощью числового значения, которое описывает определенный *цвет RGB* (цветовую модель, применяемую для формирования изображения на компьютерных мониторах). Возможно, вы где-нибудь видели такие значения цвета:

```
color: #FF0000; color: #808000; color: #00F;
```

Мы кратко рассмотрим все особенности, связанные с определением цветов RGB, но сначала перейдем к небольшому, но интересному разделу, посвященному стандартным названиям цветов.

Названия цветов

Наиболее интуитивный способ указать цвет — назвать его по имени. К сожалению, нельзя просто придумать название цвета и ожидать, что компьютер его воспримет. Компьютерный цвет определяется с помощью одного из ключевых слов, предопределенных в рекомендациях CSS. В CSS1 и CSS2 приняты 16 стандартных названий цветов, изначально представленных в HTML 4.01. С появлением CSS2.1 к ним добавилось название цвета *orange*, и количество названий цветов увеличилось до 17 (рис. ЦВ-13.1).

В CSS3 включена поддержка расширенного набора из 140 (весьма причудливых) названий цветов. Теперь можно указывать названия цветов и так: *burlwood* (твердое дерево), *peachpuff* (пушок персика), *oldlace* (старинное кружево), а вот и мой самый любимый цвет — *papayawhip* (побег папайи)! Расширенный набор цветов приведен на рис. ЦВ-13.2, но если вы хотите получить более точное его представление, обратитесь на сайт learningwebdesign.com/colornames.html. В CSS3 также появилось ключевое слово *transparent* (прозрачность), которое можно применять вместе с любым свойством, имеющим значение цвета.

ЗАБАВНЫЙ ФАКТ

Расширенный список названий цветов, также известных как названия цветов X11, изначально был предоставлен в системе X Window для UNIX.

Названия цветов просты в применении — просто используйте их в качестве значения для любого связанного с цветом свойства:

```
color: silver;  
background-color: gray;  
border-bottom-color: teal;
```

Цветовые значения RGB

Названия цветов просты, но, перечень их, как бы его ни расширяли, ограничен. Безусловно, более распространенным способом указания цвета служит указание его RGB-значения. Это даст вам выбор из миллионов цветов.

И, прежде чем углубляться в синтаксис CSS, я сначала поясню читателям, незнакомым с тем, как компьютеры работают с цветом, основы этих процессов.

Кратко о цветах RGB

Компьютеры формируют видимые на мониторе цвета, комбинируя три цвета: красный (Red), зеленый (Green) и синий (Blue). Отсюда и название — *цветовая модель RGB*. Вы можете предоставлять компьютеру своего рода «рецепты приготовления» цветов, сообщая ему, какая часть каждого цвета включается в их комбинацию. Степень яркости в каждом цветовом «канале» обычно описывается в диапазоне от 0 (нет) до 255 (полный свет), иногда ее указывают и в процентах. Чем ближе значение каждого цвета к 255 (100%), тем ближе полученный цвет к белому (рис. ЦВ-13.3). Интересно, почему диапазон яркости цветов ограничен значениями от 0 до 255? — обратитесь к врезке «Почему 255?».

ПОЧЕМУ 255?

В истинном цвете RGB для каждого цветового канала выделено 8 битов информации. Поскольку 8 битов могут описывать 256 оттенков (2 в 8-й степени равно 256), цвета оцениваются в диапазоне от 0 до 255.

Любой цвет, представленный на мониторе, описывается последовательностью из трех чисел: значение красного, значение зеленого и значение синего цветов. Это один из способов, которыми графические редакторы, такие, как Adobe Photoshop, отслеживают цвета для каждого пикселя в изображении. В цветовой системе RGB приятный лавандовый цвет может описываться так: R: 200, G: 178, B: 230.

255 цветов каждого канала, взятые вместе, определяют около 16,7 миллионов цветовых комбинаций. Это цветовое пространство, состоящее из миллионов цветов, известно как *Truecolor*. Разработаны различные способы для кодирования этих цветов (то есть для преобразования их в байты для компьютеров), а кодировка, применяемая в Интернете, называется *sRGB*. Поэтому, если в графической программе предоставляется возможность сохранить изображение в формате sRGB, щелкните на кнопке **Yes** (Да).

Выбор цвета

Существует несколько способов выбрать цвет и найти его RGB-значения. Один из быстрых и простых вариантов — зайти на сайт Google.com и выполнить поиск по ключевым словам *color picker* (палитра цветов), и вот, пожалуйста — перед вами полнофункциональный инструмент выбора цвета (рис. ЦВ-13.4, слева)! Если у вас открыта программа редактирования изображений, такая, как Adobe Photoshop, можно обратиться и к ее встроенной палитре цветов (рис. ЦВ-13.4, справа).

ВЕБ-ПАЛИТРА

В инструментах веб-дизайна, таких, как Dreamweaver или Photoshop, можно встретить термины *веб-палитра* или *веб-безопасные цвета*. Интернет зародился в те времена, когда компьютерные мониторы одновременно отображали только 256 цветов. Веб-палитра состояла из 216 цветов, которые можно было отображать в операционных системах Windows и Macintosh без сглаживания, и именно они были «безопасными» для Интернета. Эта эпоха давно позади, так же как и необходимость ограничивать выбор цвета веб-палитрой.

Как палитры цветов Google, так и палитры графического редактора демонстрируют, в каком виде выбранный цвет отображается в различных цветовых моделях (чтобы отобразить значения в Google, щелкните под палитрой на кнопке **Show color values** (Показать значения цвета). Наиболее популярна в веб-дизайне модель RGB, поэтому мы и рассматриваем ее подробнее. Цветовая модель *HSL*: Hue (оттенок), Saturation (насыщенность), Lightness (яркость) или Luminosity (светимость) — представляет собой еще один вариант указания цвета в таблицах стилей (речь о ней пойдет немного позже). Цветовая модель *CMYK*: Cyan (голубой), Magenta (пурпурный), Yellow (желтый), black (черный) — применяется, главным образом, в процессе полиграфической печати и в веб-дизайне обычно не задействуется, за исключением, возможно, случая, когда цветные печатные изображения переводятся в их экранные эквиваленты.

МОДЕЛЬ HSL

Модель *HSL* — не то же самое, что *HSB*: Hue, Saturation, Brightness (оттенок, насыщенность, яркость). Это другая цветовая модель, представленная в Photoshop и других графических редакторах.

При выборе цвета из палитры цветов в ней отображаются его красные, зеленые и синие значения, как показано на рис. ЦВ-13.4. Однако обратите внимание на запись, начинающуюся с символа #, — после него указаны три значения, преобразованные в шестнадцатеричные эквиваленты, уже подготовленные к применению в таблице стилей. Рассмотрим смысл таких шестизначных шестнадцатеричных значений.

Запись значений RGB в таблицах стилей

Таблицы CSS позволяют указывать значения цвета RGB в нескольких форматах. Возвращаясь к приятному лавандовому цвету, можно добавить его в таблицу стилей, задавая каждое значение составляющих его цветов с учетом диапазона значений от 0 до 255:

```
color: rgb(200, 178, 230);
```

Можно задать их и в виде процентных значений, хотя подобный подход встречается реже:

```
color: rgb(78%, 70%, 90%);
```

А теперь взглянем на шестизначную шестнадцатеричную версию записи цвета, речь о которой шла при рассмотрении палитры цветов. Эти шесть цифр представляют три значения цветов RGB, преобразованные в шестнадцатеричные (или для краткости — hex) эквиваленты. Шестнадцатеричным значениям RGB предшествует символ #, и они не нуждаются в имеющейся в предыдущих примерах записи `rgb()`. Отображаются они с использованием верхнего или нижнего регистра, но рекомендуется записывать их без пробелов:

```
color: #C8B2E6;
```

Есть и еще один способ для указания шестнадцатеричных значений цвета. Если значение состоит из трех пар одинаковых цифр или букв:

```
color: #FFCC00; или color: #993366;
```

можно ужать каждую такую пару до одной цифры (буквы). Тогда будет проще вводить и просматривать значения цветов, это также немножко уменьшает размер файла. Следующие примеры эквивалентны приведенным ранее — вместо них можно записать:

`color: #FC0;` или `color: #936;`

Все эти примеры определяют белый цвет.

УКАЗАНИЕ ЗНАЧЕНИЙ RGB

Существуют четыре формата для предоставления значений RGB в таблицах стилей CSS:

```
rgb(255, 255, 255)
rgb(100%, 100%, 100%)
#FFFFFF
#FFF
```

Еще о шестнадцатеричных значениях

Уточним, что же означает шестизначная строка символов. На самом деле вы видите последовательность из трех двузначных чисел — по одной для красного, зеленого и синего цветов. Вместо привычной нам десятичной системы (с основанием 10) эти значения записаны в шестнадцатеричном формате, или в системе с основанием 16. На рис. ЦВ-13.5 показана структура шестнадцатеричного RGB-значения.

Система шестнадцатеричного счисления использует 16 цифр: 0–9 и A–F (для представления величин от 10 до 15) — на рис. 13.6 показано, каким образом это происходит. Шестнадцатеричная система широко применяется в вычислениях, поскольку уменьшает объем, требуемый для хранения определенной информации. Например, значения RGB после их преобразования в шестнадцатеричные уменьшаются с трех до двух цифр.

Десятичная	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Шестнадцатеричная	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

	место для шестнадцати	место для единиц	2 раза по шестнадцать и 0 единиц	2 раза по шестнадцать и 10 единиц
Десятичное число 32 представляется так:	20	Десятичное число 42 представляется так:	2A	

Рис. 13.6. Шестнадцатеричная система счисления (с основанием 16)

Поскольку в настоящее время большая часть программного обеспечения для графики и веб-дизайна поддерживает удобный доступ к шестнадцатеричным значениям цвета (как показано на рис. ЦВ-13.4), нет необходимости самостоятельно переводить значения RGB в шестнадцатеричные значения, как это требовалось раньше, — можно обратиться к онлайн-конвертерам десятичных чисел в шестнадцатеричные.

УДОБНЫЕ ШЕСТНАДЦАТЕРИЧНЫЕ ЗНАЧЕНИЯ

```
Белый = #FFFFFF или #FFF
(эквивалент для 255,255,255)
Черный = #000000 или #000
(эквивалент для 0,0,0)
```

Цвет RGBa

Цвет RGBa позволяет указывать цвет и добавлять к нему столько прозрачности, сколько необходимо. Буква «а» в аббревиатуре «RGBa» — эта первая буква слова *alpha* (альфа), которое обозначает дополнительный канал, управляющий уровнем прозрачности в диапазоне от 0 (полностью прозрачный) до 1 (полностью непрозрачный). Вот как это записывается при определении правила стиля:

```
color: rgba(0, 0, 0, .5);
```

Первые три значения в скобках являются обычными прежними значениями RGB (в рассматриваемом случае формируется черный цвет). Четвертое значение: .5 — определяет уровень прозрачности. Наш черный цвет при этом имеет прозрачность, равную 50%. Поэтому другие цвета или фоновые рисунки слегка просвечивают через него (рис. 13.7).



```
color: rgba(0, 0, 0, .1);
color: rgba(0, 0, 0, .5);
color: rgba(0, 0, 0, 1);
```

Рис. 13.7. Заголовки с различными уровнями прозрачности при использовании значений RGBa

О ПОДДЕРЖКЕ БРАУЗЕРАМИ ЦВЕТОВ RGBa

Браузер Internet Explorer версии 8 и более ранних не поддерживают цвета RGBa, поэтому, если значительное количество ваших пользователей имеют подобные браузеры, желательно применять резервный вариант. Выберите цвет RGB, приблизительно соответствующий нужному внешнему виду, и первым включите его в правило стиля. Браузер IE проигнорирует значение RGBa, а поддерживающие такие значения браузеры переопределят непрозрачный цвет при обращении ко второму объявлению в правиле:

```
h1 {
    color: rgb(120, 120, 120);
    color: rgba(0, 0, 0, .5);
}
```

Цвет HSL

В CSS3 появилась возможность задавать цвета по их HSL-значениям: Hue (оттенок, цвет), Saturation (насыщенность) и Lightness (яркость) или Luminosity (светимость). В этой системе значения цвета располагаются по кругу, как в радуге, с красным (Red) в верхней позиции (12 часов). Значения Hue распределяются по этому кругу в градусах: красный — при 0°/360°, зеленый — при 120° и синий — при 240°, с другими цветами между ними. Значения Saturation (насыщенность) — измеряются в процентах:

от 0% (серый цвет) до 100% (цвет полного насыщения). Значения Lightness (яркость) или Luminosity (светимость) также представлены в процентах: от 0% (самый темный вид) до 100% (самый светлый).

На рис. ЦВ-13.8 показан один оттенок (Hue) — голубой (cyan), расположенный на отметке 180° круга, с соответствующими уровнями насыщенности и яркости. Можно понять, почему некоторые пользователи считают эту систему интуитивно понятной в применении, поскольку, зафиксировав оттенок, легко усилить его, затемнить или осветлить, увеличивая или уменьшая процентные значения насыщенности и яркости. Значения RGB не носят столь интуитивный характер, хотя некоторые опытные дизайнеры чувствуют и их.

В CSS значения HSL представлены с помощью значения оттенка и двух процентных величин. Они не преобразуются в шестнадцатеричные значения, как это делается при работе с RGB. Приятный лавандовый цвет, показанный на рис. 13.3 (см. крайний правый вариант), здесь получен с использованием HSL:

```
color: hsl(265, 51%, 80%);
```

Палитры цветов HSL

В Интернете можно найти множество палитр цветов HSL. В палитре цветов Google щелкните под панелью на кнопке **Show color values** (Показать значения цвета) для отображения значения HSL для выбранного цвета. Рассмотрим другие интересные инструменты, которые привлекут ваше внимание:

- «A Most Excellent HSL Color Picker» от Брэндона Матиса (Brandon Mathis) (hslpicker.com/);
- «HSL Color Picker» (www.workwithcolor.com/hsl-color-picker-01.htm);
- «HSLa Explorer» от Криса Коуера (Chris Coyier), представленный в CSS-Tricks (css-tricks.com/examples/HSLaExplorer/).

О ЦВЕТОВОЙ МОДЕЛИ HSB

Помните, что имеющаяся в палитре цветов Photoshop (см. рис. ЦВ-13.3) цветовая модель HSB, отличается от HSL и не может применяться для CSS.

Цвет HSLa

Как и в случае с RGB, для придания прозрачности цветам HSL можно добавить альфа-канал, чтобы получить цветовую модель HSLa. Как и в RGBa, четвертое значение определяет степень прозрачности по шкале от 0 (полностью прозрачная) до 1 (полностью непрозрачная). В следующем примере определяется весенний зеленый цвет, непрозрачный на 65%:

```
color: hsla(70, 60%, 58%, .65);
```

О ПОДДЕРЖКЕ БРАУЗЕРАМИ ЦВЕТОВ HSL И HSLa

Цвета HSL и HSLa не поддерживаются браузерами Internet Explorer версии 8 и более ранних, так что если необходимо поддерживать подобные браузеры, применяйте резервные варианты.

Краткие промежуточные итоги

Несмотря на то что тема выбора и задания цвета заняла несколько страниц, сам процесс выбора и указания цветов в таблицах стилей достаточно прост:

- выберите одно из предварительно определенных названий цветов, или
- для выбора цвета обратитесь к палитре цветов и скопируйте значения RGB (предпочтительно шестизначные шестнадцатеричные значения). Поместите эти значения в правило стиля, используя один из четырех форматов значений RGB, и на этом все. Или же обратитесь к возможностям модели HSL.

Существует еще один весьма красочный способ заливки элемента цветом, заключающийся в использовании *градиентов* (цвета, переходящие от одного оттенка в другой), но я расскажу о нем в конце главы.

СВОЙСТВО **COLOR** (ЦВЕТ ПЕРЕДНЕГО ПЛАНА)

ПЕРЕДНИЙ ПЛАН

Передний план элемента состоит из текста и рамки (если она указана).

переднего плана и фона. Задать можно также и цвет рамки, обратившись к свойствам `border-color`, но их мы рассмотрим позже, в главе 14.

Передний план элемента состоит из текста и рамки (если она указана). Как показано в предыдущей главе, цвет переднего плана задается с помощью свойства `color`. Рассмотрим некоторые аспекты свойства `color` еще раз:

color

- **Значения:** значение цвета (название или числовое значение).
- **По умолчанию** — зависит от браузера и предпочтений пользователя.
- **Применение** — ко всем элементам.
- **Наследование** — да.

В следующем примере для переднего плана элемента `blockquote` с помощью соответствующего названия выбран зеленый цвет. Применение свойства `color` к элементу `blockquote` означает, что цвет наследуется содержащимися в нем элементами `p` и `em` (рис. ЦВ-13.9). Широкая пунктирная рамка вокруг элемента `blockquote` также окрашивается в зеленый цвет (однако, если к этому элементу будет применено свойство `border-color`, заданный им цвет переопределит установку для этой рамки зеленого цвета):

Правило стиля

```
blockquote {  
    border: 4px dashed;  
    color: green;  
}
```

Теперь, когда мы знаем, как записываются значения цветов, перейдем к определяющим цвет свойствам. Для любого элемента HTML можно задавать цвета

Разметка

```
<blockquote>  
In the latitude of central New England, cabbages are not  
secure ...  
</blockquote>
```

СВОЙСТВО **BACKGROUND-COLOR** (ЦВЕТ ФОНА)

Цвет фона (заднего плана) любого элемента определяется свойством `background-color`:

background-color

- **Значения:** значение цвета (название или число) | `transparent`.
- **По умолчанию** — `transparent`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Цвет фона заполняет *холст* позади элемента — сюда входят область его содержимого и любые отступы (дополнительные пробелы), добавляемые вокруг этого содержимого, включая его границы вплоть до внешнего края. Рассмотрим, что произойдет, если применить свойство `background-color` для окрашивания в зеленый цвет фона элемента `blockquote` из предыдущего примера (рис. ЦВ-13.10):

```
blockquote {  
    border: 4px dashed;  
    color: green;  
    background-color: #c6de89;  
}
```

Как и следовало ожидать, цвет фона заполняет область за текстом вплоть до ее границ. Посмотрите внимательно на пропуски в пунктирной рамке, и вы увидите, что цвет фона доходит до самого ее внешнего края. Но далее фоновый цвет не распространяется, и, если мы зададим вокруг нашего элемента поле, фон не станет распространяться и на это поле. Мы рассмотрим все эти тонкости, когда начнем говорить о блочной модели CSS. А пока просто знайте, что по умолчанию, если в рамке, окружающей ваш элемент, имеются пропуски, фон будет через них просвечивать.

Следует отметить, что цвета фона не наследуются, но, поскольку по умолчанию настройка фона для всех элементов прозрачна (`transparent`), цвет фона родителя просвечивает через элементы-потомки. Например, изменить цвет фона всей страницы можно, применив свойство `background-color` к элементу `body`, причем этот цвет будет отображаться для всех элементов страницы (см. врезку «*Важное исключение*»).

ВАЖНОЕ ИСКЛЮЧЕНИЕ

Когда фоновый цвет применяется к элементу `body` (или, более обобщенно, к корневому элементу `html`), его обработка проходит специальным образом. Фон при этом не привязывается к блоку, а распространяется на весь экран.

ОКРАШИВАНИЕ ФОНОВОЙ ОБЛАСТИ ВСЕЙ СТРАНИЦЫ

Для окрашивания фоновой области всей страницы примените свойство `background-color` к элементу `body`.

Слова, помеченного как `glossary` (словарное), использовались свойства `color` и `background-color`. Как показано на рис. ЦВ-13.11, цвет фона заполняет небольшой прямоугольник, образованный встроенным элементом `dfn`:

Правило стиля

```
.glossary {
    color: #0378a9; /* blue */
    background-color: yellow;
}
```

Разметка

```
<p>Every variety of cabbage had their origin in the wild
cabbage of Europe (<dfn class="glossary"><i>Brassica oleracea</i></dfn>)</p>
```

СВОЙСТВО **BACKGROUND-CLIP** (ОБРЕЗКА ФОНА)

Традиционно область заливки фона (область применения цветов заливки) элемента простирается до внешнего края его границы, как показано на рис. ЦВ-13.10. В CSS3 появилось свойство `background-clip`, позволяющее дизайнерам точнее управлять местоположением области рисования:

`background-clip`

- **Значения:** `border-box` | `padding-box` | `content-box`.
- **По умолчанию** — `border-box`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Заданное по умолчанию значение `border-box` свойства `background-clip` формирует область заливки фона, доходящую до внешнего края границы элемента (рис. ЦВ-13.12, *вверху*), что мы уже видели и ранее (см. рис. 13.10). На рис. ЦВ-13.12

Не обязательно задавать фоновый цвет для всей страницы — можно изменять цвет фона любого элемента, как на уровне блока (см. например, предыдущий пример), так и для встроенного элемента. В следующем примере для выделения

слова, помеченного как `glossary` (словарное), использовались свойства `color` и `background-color`. Как показано на рис. ЦВ-13.11, цвет фона заполняет небольшой прямоугольник, образованный встроенным элементом `dfn`:

Правило стиля

```
.glossary {
    color: #0378a9; /* blue */
    background-color: yellow;
}
```

Разметка

```
<p>Every variety of cabbage had their origin in the wild
cabbage of Europe (<dfn class="glossary"><i>Brassica oleracea</i></dfn>)</p>
```

СВОЙСТВО **BACKGROUND-CLIP** (ОБРЕЗКА ФОНА)

Традиционно область заливки фона (область применения цветов заливки) элемента простирается до внешнего края его границы, как показано на рис. ЦВ-13.10. В CSS3 появилось свойство `background-clip`, позволяющее дизайнерам точнее управлять местоположением области рисования:

`background-clip`

- **Значения:** `border-box` | `padding-box` | `content-box`.
- **По умолчанию** — `border-box`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Заданное по умолчанию значение `border-box` свойства `background-clip` формирует область заливки фона, доходящую до внешнего края границы элемента (рис. ЦВ-13.12, *вверху*), что мы уже видели и ранее (см. рис. 13.10). На рис. ЦВ-13.12

в *центре* показано, каким образом значение padding-box определяет область фоновой заливки на внешнем крае области отступа для элемента (и на внутренней части границы этой области). Наконец, свойство content-box позволяет выполнять заливку фоном только в области содержимого элемента (рис. ЦВ-13.12, *внизу*).

Я чувствую, что забегаю вперед и немного порчу сюрприз, который вас ожидает в следующей главе, где я буду рассказывать вам об элементах блочной модели и их свойствах, но здесь мне пришлось добавить небольшие отступы (пробелы между содержимым и границей), чтобы лучше показать вам эффекты, связанные с применением настроек свойства background-clip:

```
blockquote {  
padding: 1em; border: 4px dashed; color: green; background-  
color: #C6DE89;}
```

НЕСКОЛЬКО ПРОСТЫХ СОВЕТОВ ПО РАБОТЕ СО ЦВЕТОМ

- Ограничьте количество применяемых на странице цветов. Ничто не создает визуальный хаос быстрее, чем слишком много цветов. Поэтому я обычно выбираю один доминирующий и один дополняющий цвет. Я также могу добавить пару оттенков каждого, но предпочитаю не задействовать слишком много разных оттенков.
- При указании цвета переднего плана и фона убедитесь, что создается достаточно контрастная картинка. Пользователи Интернета предпочитают читать темный текст на светлом фоне.
- При выборе цвета не забывайте о дальтониках. Статья Криса Койера (Chris Coyier) «Accessibility Basics: Testing Your Page for Color Blindness» (css-tricks.com/accessibility-basics-testing-your-page-for-color-blindness/) содержит изложение стратегий, применяемых для реализации дружественного к дальтоникам дизайна.

Цвет способствует эстетическому восприятия и удобству использования сайта, поэтому важно правильно его формировать. Книга Гери Коади (Geri Coady) «Color Accessibility Workflows» содержит большое число практических рекомендаций по этой теме.

СВОЙСТВО **OPACITY** (УПРАВЛЕНИЕ НЕПРОЗРАЧНОСТЬЮ)

Ранее речь шла о цветовом формате RGBa, который добавляет цвету переднего плана или фона определенный уровень прозрачности. Однако имеется еще один способ управления уровнем прозрачности элемента — CSS3-свойство opacity:

opacity

- **Значения:** число (от 0 до 1).
- **По умолчанию** — 1.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Значением `opacity` является число в интервале величин от 0 (полностью прозрачное) до 1 (полностью непрозрачное). Значение .5 соответствует уровню непрозрачности элемента, равному 50%. Свойство `opacity` (если оно установлено) применяется ко всему элементу — как к его переднему плану, так и к фону. Если необходимо повлиять только на передний план или только на фон, применяйте вместо свойства `opacity` значение цвета модели RGBa.

СВОЙСТВО `OPACITY`

Значение *непрозрачности* применяется ко всему элементу — как к его переднему плану, так и к фону.

отображается как через текст, так и через поле элемента:

```
h1 {color: gold; background: white; opacity: .25;}
h1 {color: gold; background: white; opacity: .5;}
h1 {color: gold; background: white; opacity: 1;}
```

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Свойство `opacity` не поддерживается в Internet Explorer версии 8 и более ранних. Если необходима поддержка IE8, примените правило стиля с собственным свойством `filter Microsoft`, а затем замените его стандартным правилом стиля `opacity`:

```
h1 {
    filter:alpha(opacity=50);
    opacity: .5;
}
```

В следующем примере кода (и на рис. ЦВ-13.13) заголовок исходно окрашен в цвет золота (`gold`), а цвет фона — белый (`white`). После задания значения свойства `opacity` синий фон страницы

Возможно, вам очень хочется попрактиковаться в применении только что рассмотренных свойств цвета переднего плана и фона, но сначала я хочу познакомить вас с некоторыми наиболее интересными CSS-селекторами, завершающими нашу коллекцию. А во врезке «*Краткий обзор селекторов*» собраны уже знакомые вам по предыдущим главам селекторы, с которыми вы должны чувствовать себя комфортно.

КРАТКИЙ ОБЗОР СЕЛЕКТОРОВ

Здесь приводится краткий обзор типов уже рассмотренных селекторов. Для сокращения записи «E» означает «Element» («Элемент»):

- селектор типа элемента — E {свойство: значение;}
- сгруппированные селекторы — E1, E2, E3 {свойство: значение;}
- селектор потомка — E1 E2 {свойство: значение;}
- дочерний селектор — E1 > E2 {свойство: значение;}
- селектор ближайших братьев (сестер) — E1 + E2 {свойство: значение;}
- селектор последующих братьев (сестер) — E1 ~ E2 {свойство: значение;}
- ID селектор — E#id {свойство: значение;}
- #id {свойство: значение;}
- селектор класса — E.class {свойство: значение;}
- .class {свойство: значение;}
- универсальный селектор — * {свойство: значение;}

СЕЛЕКТОРЫ ПСЕВДОКЛАССА

Замечали ли вы, что ссылка, когда вы щелкаете на ней, имеет один цвет, но при возвращении на ту же страницу приобретает другой цвет? Дело в том, что в фоновом режиме браузер отслеживает, какие ссылки были активированы (или «посещались», если применить жargon пользователей Интернета). Браузер отслеживает и другие состояния — например, находится ли курсор пользователя над элементом (при наведении), является ли элемент первым в своем типе, является ли он первым или последним дочерним элементом своего родительского элемента, а также является ли элемент формы активным, или находится в отключенном состоянии.

При работе в CSS к элементам в таких состояниях можно применять стили, используя специальный тип селектора, называемый селектором *псевдокласса*. Это название может показаться странным, но вы просто представьте себе, что находящиеся в определенном состоянии элементы принадлежат как бы одному классу. Тем не менее имя класса отсутствует в разметке — браузер таким образом просто воспринимает подобные элементы при отслеживании. Так что этот класс не совсем настоящий, поэтому и называется *псевдоклассом*.

Селекторы псевдоклассов обозначаются двоеточием (:). Обычно оно ставится сразу после имени элемента — например, `li:first-child`.

В CSS3 содержится весьма много псевдоклассов — похоже, W3C перестарался, включая в CSS Selector Module уровня 4 столько новых псевдоклассов, большинство из которых на момент подготовки книги не имели поддержки со стороны браузеров. В следующем разделе я познакомлю вас с наиболее часто применяемыми и имеющими лучшую поддержку псевдоклассами, которыми вы можете воспользоваться в начале своего творческого пути. А по мере накопления практических навыков вы сможете включить в свой арсенал и самые современные селекторы. Полный список CSS-селекторов (включая селекторы уровня 4) с соответствующими описаниями приведен в *приложении 3*.

Ссылки для псевдоклассов

Основные селекторы псевдоклассов предназначаются для ссылок (элементов `a`), которые были нажаты. Ссылки псевдоклассов — это тип *динамического псевдокласса*, поскольку применяются при взаимодействии пользователя со страницей, а не при обращении к разметке:

- `:link` — применяет стиль к неактивным (не посещенным) ссылкам;
- `:visited` — применяет стиль к ссылкам, на которых уже выполнен щелчок.

По умолчанию браузеры отображают текст ссылок, которые еще не были нажаты, синим цветом, а ссылки, щелчок на которых уже был произведен, — фиолетовым, но изменить эти настройки можно с помощью нескольких правил стиля. Между тем имеются ограничения, определяющие свойства, которые можно применить к ссылкам `:visited`, как уточнено во врезке «*Посещенные ссылки и безопасность*».

В следующих примерах цвет неактивных ссылок изменен на темно-бордовый, а посещенные ссылки окрашиваются в серый цвет. Обычно посещенные ссылки имеют слегка приглушенный цвет по сравнению с неактивными ссылками:

НЕ УТРАТЬТЕ ВНЕШНИЙ ВИД ССЫЛОК

При изменении внешнего вида ссылок и посещаемых ссылок, убедитесь, что они по-прежнему выглядят как ссылки.

```
a:link {  
    color: maroon;  
}  
  
a:visited {  
    color: gray;  
}
```

ПОСЕЩЕННЫЕ ССЫЛКИ И БЕЗОПАСНОСТЬ

Браузеры отслеживают посещенные ссылки, но иногда пользователям может быть нежелательно отслеживание посещенных ими ссылок (они иногда «подсматриваются» вредоносными сайтами). А для некоторых пользователей из регионов с жесткими ограничениями на просмотр онлайн-контента подобное отслеживание может стать угрозой для их жизни. Когда оказалось, что применяемые к посещенным ссылкам визуальные стили, а также методы, задействуемые браузерами для отслеживания таких ссылок, могут использоваться при отслеживании истории просмотров пользователя, в способ обработки посещенных ссылок были внесены некоторые изменения.

Первым изменением служит ограничение свойств визуального представления, которые можно применять к посещаемым ссылкам. Правила стиля с селекторами псевдоклассов `:visited` могут использовать только следующие свойства: `color`, `background-color`, `border-color` (и отдельные свойства боковой границы) и `outline-color`. Любое иное свойство будет проигнорировано. Кроме того, нельзя применять какое-либо значение, которое делает ссылку прозрачной, включая ключевое слово `transparent` и значения цвета, относящиеся к модели RGBa и HSLa.

При этом механизм DOM, который отслеживает посещенные ссылки, будет постоянно возвращать состояние «не посещено», даже при отображении на экране посещенных стилей. Подобный подход также скрывает историю просмотра на уровне DOM.

Будущность псевдокласса `:visited` не определена, поэтому не применяйте стили, имеющие важное значение при использовании вашего сайта.

Псевдоклассы действий пользователя

Иной тип динамических псевдоклассов предназначен для состояний, являющихся результатом прямых действий пользователя:

- `:focus` — применяется, если элемент выбран и готов к вводу;
- `:hover` — применяется, если указатель мыши находится над элементом;
- `:active` — применяется, когда элемент (например, ссылка или кнопка) участвует в процессе нажатия или касания.

Состояние фокуса

Если вы использовали веб-форму, вам известно, каким образом при выборе элемент формы браузер визуально его выделяет. Если элемент выделен и готов к вводу,

говорят, что он имеет «фокус». Селектор `:focus` позволяет применять пользовательские стили к элементам, когда они находятся в состоянии фокуса (выделения).

В следующем примере, когда пользователь выбирает ввод текста, цвет фона становится желтым, что выделяет его среди других полей ввода формы:

```
input:focus { background-color: yellow; }
```

Состояние наведения

Селектор `:hover` представляет особый интерес. Этот селектор выбирает элементы, если над ними находится указатель мыши. Состояние наведения можно использовать для любого элемента, но чаще всего оно связывается со ссылками и предоставляет пользователю визуальную обратную связь о возможности действия. Состояние наведения также применяется для активизации (вызыва) всплывающих меню при навигации или для получения дополнительной информации об объекте страницы.

Следующее правило приводит к тому, что фон ссылки при наведении на нее указателя мыши окрашивается в светло-розовый цвет:

```
a:hover {  
    color: maroon;  
    background-color: #ffd9d9;  
}
```

В предыдущей главе речь шла о свойстве `text-decoration`, с помощью которого можно отключить в ссылках подчеркивание. Точно так же вы можете применять селектор `:hover`, чтобы подчеркивание отображалось при наведении:

```
a:hover {  
    text-decoration: underline;  
}
```

Важно отметить, что для устройств с сенсорным экраном, таких, как смартфоны и планшеты, не существует реального состояния наведения, поэтому эффектами наведения следует пользоваться осторожно и включать их в правила стилей в качестве альтернативных решений (см. врезку «Наведение на сенсорных устройствах»).

НАВЕДЕНИЕ НА СЕНСОРНЫХ УСТРОЙСТВАХ

На экране компьютера вы легко можете наводить указатель мыши на расположенные на нем элементы, однако сенсорные устройства реагируют только на прикосновение к экрану. Поэтому отсутствие эффекта наведения на смартфонах и планшетах требует принятия особых мер.

Так, если эффекты наведения применяются к ссылке (элементу `a`), мобильные операционные системы отображают стили состояния наведения после совершения первого ее касания. А для перехода по ссылке пользователь должен коснуться ссылки еще раз. Другие элементы, вызываемые при наведении курсора, — например, всплывающие меню, могут остаться в открытом положении, и тогда пользователю придется коснуться другого места страницы или перезагрузить страницу для ее очистки (что не слишком это хорошо, а для некоторых ситуаций — губительно). ▶

- ▶ Эта проблема не имеет на основе использования таблиц CSS единого решения. Иногда выручают стили :focus, :active и :hover. В ряде случаев можно обратиться к JavaScript и с его помощью запрограммировать для мобильных устройств желаемый эффект. Можно также вовсе избегать состояния :hover и просто выполнять последовательные щелчки. А можно включать в таблицу стилей специально предназначенные для сенсорных устройств стили, не предусматривающие состояния наведения. Рассмотрение возможностей JavaScript выходит за пределы этой главы, поэтому я настоятельно рекомендую вам следующие ресурсы для приобретения необходимых навыков работы с JavaScript:
- «4 novel ways to deal with sticky :hover effects on mobile devices» (www.javascriptkit.com/dhtmltutors/sticky-hoverissue-solutions.shtml);
 - выполните поиск по ключевым словам hover states on touch devices на сайте StackOverflow и посмотрите связанные с этой проблемой вопросы и ответы. Stack Overflow — это форум, где программисты, задавая вопросы, получают помощь других программистов. Здесь доступно большое число решений, но встречаются и тупиковые ситуации.

Активное состояние

Наконец, селектор :active применяет стили к элементу, находящемуся в процессе активации, — когда на ссылке щелкают мышью или касаются ее пальцем на экране сенсорного устройства. Этот стиль может отображаться только мгновение, но при этом точно указать, что произошло. В следующем примере я усилила яркость цвета для активного состояния (от темно-бордового до красного):

```
a:active {  
    color: red;  
    background-color: #ffd9d9;  
}
```

Краткие промежуточные итоги

Обязательный порядок для псевдоклассов:

```
:link  
:visited  
:focus  
:hover  
:active
```

Веб-дизайнеры обычно поддерживают стили для всех состояний ссылок, поскольку на каждом этапе нажатия на ссылку необходим какой-нибудь отклик (что улучшает заданные по умолчанию настройки браузера). Да и пользователи ожидают иметь подобную обратную связь: они сразу видят, какие ссылки уже проидены, понимают, что ссылка при наведении на нее активизирована, и получают подтверждение об успешном нажатии ссылок.

Для правильного функционирования стилей, применяемых к элементам, располагающим всеми пятью псевдоклассами, важен порядок их отображения. Например, если последними указать :link или :visited, они переопределят другие состояния и предотвратят их отображение. Поэтому для ссылок псевдоклассов необходимый соблюдать следующий порядок: :link, :visited, :focus, :hover, :active (LVFHA — что можно легко запомнить как LoVe For Hairy Animals, или выберите другое мнемоническое правило на свой вкус).

Пользователям, которые для перехода по ссылкам на странице используют клавиатуру, а не щелкают мышью, рекомендуется предоставить стиль :focus. Обычно им предоставляется и стиль :hover, но это не является обязательным.

В следующем примере, суммируя сказанное, я показала, как должна выглядеть разметка стилей ссылок в таблице стилей (рис. ЦВ-13.14):

```
a { text-decoration: none; } /* turns underlines off for all links */  
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; }
```

Другие селекторы псевдоклассов

Итак... вы познакомились с пятью псевдоклассами CSS3, и осталось всего каких-то 40! Времени знакомство с ними займет немало, а ведь у нас для изучения имеются и другие типы селекторов. Тем не менее вам следует знать о том, что сейчас находится в разработке, поэтому во врезке «Еще о псевдоклассах CSS3» приведена о селекторах псевдоклассов CSS3 дополнительная информация. Более того, полный список селекторов уровня 3 и 4 с кратким описанием можно найти в *приложении 3*.

Также я настоятельно рекомендую ознакомиться со статьей «An Ultimate Guide to CSS Pseudo-Classes and Pseudo-Elements» Рикардо Зеа (Ricardo Zea) в журнале Smashing Magazine (www.smashingmagazine.com/2016/05/an-ultimate-guide-to-css-pseudo-classes-andpseudo-elements/). В ней подробно на примерах рассматриваются все селекторы псевдоклассов CSS3.

ЕЩЕ О ПСЕВДОКЛАССАХ CSS3

Консорциум W3C предложил интересные варианты для стайлинга выделяемого содержимого на основе отслеживаемых браузером «на лету» состояний.

В CSS3 представлен целый ряд псевдоклассов, большинство которых уже поддерживаются браузерами. Конечно, Internet Explorer 8 и более ранние его версии не поддерживают эти псевдоклассы, но, в тех случаях когда нужно включить поддержку IE6–8, воспользуйтесь полифилом Selectivizr (selectivizr.com).

Отличным ресурсом для получения дополнительной информации о CSS-селекторах уровня 3 и 4, включая информацию о поддержке браузера, является сайт **CSS4-selectors.com** от Нейла Брекардина (Nelly Brekardin). ▶

▶ Структурные псевдоклассы

При использовании структурных псевдоклассов выбор разрешен в зависимости от местонахождения элемента в структуре документа (используется дерево документа):

```
:root  
:empty  
:first-child  
:last-child  
:only-child  
:first-of-type  
:last-of-type  
:only-of-type  
:nth-child()  
:nth-last-child()  
:nth-of-type()  
:nth-last-of-type()
```

Псевдоклассы ввода

Эти селекторы применяются к состояниям, типичным для ввода формы:

```
:enabled  
:disabled  
:checked
```

Псевдоклассы местоположения (в дополнение к :link и :visited)

```
:target (идентификатор фрагмента)
```

Лингвистические псевдоклассы

```
:lang()
```

Логические псевдоклассы

```
:not()
```

СЕЛЕКТОРЫ ПСЕВДОЭЛЕМЕНТОВ

Псевдоклассы — не единственный вид псевдоселекторов. Есть также четыре псевдоэлемента, действующие таким образом, как будто они для стайлинга вставляют в структуру документа вымышленные элементы. В CSS3 псевдоэлементы обозначаются двойным символом двоеточия (::), что позволяет отличать их от псевдоклассов. Однако все браузеры поддерживают этот синтаксис и с одиночным двоеточием (:), определенным в CSS2, поэтому разработчики придерживаются подобного обозначения для обратной совместимости со старыми браузерами.

Первая буква и строка

Следующие псевдоэлементы применяются для выбора первой строки или первой буквы текста в отображаемом в браузере элементе:

::first-line

Этот селектор задает правило стиля для первой строки указанного элемента. Учтите, что таким образом можно применять только следующие свойства:

color	text-decoration
font properties	vertical-align
background properties	text-transform
word-spacing	line-height
letter-spacing	

::first-letter

Этот селектор задает правило стиля для первой буквы указанного элемента. Используются только следующие свойства:

color	vertical-align (если отсутствует режим float «плавания»)
font properties	padding properties
background properties	margin properties
letter-spacing	border properties
word-spacing	line-height
text-decoration	float
text-transform	

В следующем примере показаны примеры использования селекторов псевдоэлементов `::first-line` и `::first-letter` (рис. 13.15):

НОВЫЕ СВОЙСТВА

В этом разделе упомянуты несколько новых для вас свойств. Свойства, связанные с блоками (margin, padding, border), будут рассмотрены в главе 14. Свойство float будет рассмотрено в главе 15.

```
p::first-line { letter-spacing: 9px; }
p::first-letter { font-size: 300%; color: orange; }
```

`::first-line` In some of the best cabbage-growing sections of the country, until within a comparatively few years it was the very general belief that cabbage would not do well on upland. Accordingly the cabbage patch would be found on the lowest tillage land of the farm.

`::first-letter` **I**n some of the best cabbage-growing sections of the country, until within a comparatively few years it was the very general belief that cabbage would not do well on upland. Accordingly the cabbage patch would be found on the lowest tillage land of the farm.

Рис. 13.15. Примеры использования селекторов псевдоэлементов `::first-line` и `::first-letter`

Генерирование контента с помощью псевдоэлементов `::before` и `::after`

Ранее было показано, каким образом браузеры автоматически добавляют в списки маркеры и цифры, даже если такие отсутствуют в исходном коде HTML. Этот подход служит примером добавления *сгенерированного контента*, то есть контента, который браузеры вставляют «на лету». Можно указать браузерам на

МОЖНО ПРИМЕНЯТЬ И ОДИНОЧНЫЕ ДВОЕТОЧИЯ

Несмотря на то что в CSS3 указаны двойные двоеточия, для обратной совместимости со старыми браузерами можно применять и одиночные двоеточия. Современные браузеры также обязаны поддерживать одиночные двоеточия.

необходимость генерировать контент до или после любого выбранного вами элемента, обращаясь к псевдоэлементам `::before` и `::after`.

Сгенерированный контент можно применять для добавления значков к элементам списка, для отображения URL-адресов следом за ссылками при выводе на печать веб-документов, для добавления соответствующих языку кавычек к цитатам и во многих других случаях. Рассмотрим простой пример, когда изображение с помощью функции `url()` вставляется перед абзацем, а выражение «Thank you» — в конце абзаца. Сравните эту разметку с тем, что отображается окне браузере (рис. ЦВ-13.16):

Правило стиля

```
p.warning::before {
    content: url(exclamation.png);
    margin-right: 6px;
}
p.warning::after {
    content: "Thank you.";
    color: red;
}
```

Разметка

```
<p class="warning">We are required to warn you that
undercooked food is health risk.</p>
```

В этом примере следует отметить несколько важных моментов:

- селектор псевдоэлемента указывается сразу после целевого элемента без пробела;
- правило псевдоэлемента включает контент и определяет его стилинг в одном блоке объявления;
- обязательно применяется свойство `content`, которое поддерживает включаемый контент. Селектор без указания этого свойства ничего не выполнит;
- если между сгенерированным контентом и контентом из исходного документа нужно добавить пробелы, символьные пробелы должны заключаться в кавычки значений или же после поля отступа.

Если же нужно включить изображение — например, значок или другую метку, укажите URL без кавычек:

```
li:before { content: url(images/star.png) }
```

ДОПОЛНИТЕЛЬНЫЙ ИСТОЧНИК

Статья Луи Лазариса (Louis Lazaris) «Learning to Use the `:before` and `:after` Pseudo-Elements in CSS» (www.smashingmagazine.com/2011/07/learning-to-use-the-before-and-afterpseudo-elements-in-css/).

При использовании сгенерированного контента учитывайте, что вставляемое содержание не становится частью DOM документа. Сгенерированный контент существует только на дисплее браузера и недоступен для вспомогательных средств — таких, как программы чтения с экрана. Лучше всего использовать

сгенерированный контент для оформления страницы и других «дополнений», которые не критичны для сути и содержания сообщения.

СЕЛЕКТОРЫ АТРИБУТОВ

В этом разделе мы завершим рассмотрение селекторов. Селекторы атрибутов выбирают элементы на основе имен или значений атрибутов, что обеспечивает большую гибкость при выборе элементов без необходимости добавления значительного количества разметки `class` или `id`.

Вот перечень селекторов атрибутов CSS3:

- `element[атрибут]` — простой селектор атрибута выбирает целевые элементы с определенным атрибутом независимо от его значения. В следующем примере выбирается изображение с атрибутом `title`:

```
img[title] {border: 3px solid;}
```

- `element[атрибут = "точное значение"]` — селектор точного значения атрибута выбирает элементы с определенным значением атрибута. Следующий селектор сопоставляет изображения с точным значением заголовка `«first grade»`:

```
img[title="first grade"] {border: 3px solid;}
```

- `element[атрибут~= "значение"]` — селектор частичного значения атрибута (указывается с помощью тильды ~) позволяет указать одну часть из значения атрибута. В следующем примере в заголовке выполняется поиск слова `«grade»`, поэтому выбираются изображения `title` со значениями `«first grade»` и `«second grade»`:

```
img[title~="grade"] {border: 3px solid;}
```

- `element[атрибут|= "значение"]` — селектор значения атрибута, разделенного дефисами (указывается с помощью символа |), выбирает разделенные дефисом значения. Следующий селектор рассматривает любую ссылку, указывающую на документ, написанный на английском языке (en), независимо от того, является ли значением атрибута `en-us` (американский английский), `en-in` (индийский английский), `en-au-tas` (австралийский английский) и т. д.:

```
[hreflang|= "en"] {border: 3px solid;}
```

- `element[атрибут^= "первая часть значения"]` — селектор значения атрибута начальной подстроки (указывается с помощью с помощью символа ^) соответствует элементам, значения атрибутов которых начинаются в селекторе со строки символов. В следующем примере стиль применяется только к тем изображениям, которые находятся в каталоге `/images/icons`:

```
img[src^="/images/icons"] {border: 3px solid;}
```

ЗАБАВНЫЙ ФАКТ

Селекторы класса и ID — это особые типы селекторов атрибутов.

- `element [атрибут$="последняя часть значения"]` — селектор значения атрибута последней подстроки (указывается с помощью символа \$) соответствует элементам, указанные значения атрибута которых заканчиваются в селекторе строкой символов. В следующем примере стиль можно применить только к элементам a, которые ссылаются на PDF-файлы:

```
a[href$=".pdf"] {border-bottom: 3px solid;}
```

- `element [атрибут*=“любая часть значения”]` — селектор значения атрибута в произвольной подстроке (указывается с помощью символа *) разыскивает указанное значение атрибута в произвольной подстроке. В следующем примере правило выбирает любое изображение, включающее в свой заголовок слово «February»:

```
img[title*="February"] {border: 3px solid;}
```

Ну вот, рассмотрение селекторов завершено! И, прежде чем перейти к фоновым изображениям, давайте выполним *упражнение 13.1*, в котором поэкспериментируем с окрашиванием переднего плана и фона, а также испытаем некоторые новые типы селекторов .

УПРАЖНЕНИЕ 13.1. ДОБАВЛЕНИЕ ЦВЕТА В ДОКУМЕНТ

При выполнении этого упражнения мы начнем с простого черно-белого меню и дополним его окрашиванием переднего плана и фона (рис. ЦВ-13.17). К этому моменту вы располагаете достаточным багажом знаний для написания правил стиля, поэтому я не стану вести вас за руку и излагать материал так подробно, как делала это в предыдущих упражнениях. Создавайте правила самостоятельно. А потом можете сравнить выполненную вами работу с уже готовой таблицей стилей, находящейся в материалах этой главы.

Откройте в текстовом редакторе файл `summer-menu.html` (этот файл доступен на странице: learningwebdesign.com/5e/materials). Там вы найдете встроенную таблицу стилей, поддерживающую базовое форматирование текста. Вам просто надо выполнить обработку цветов. Сохраняйте документ на каждом этапе работы и просматривайте результат выполнения работы в браузере.

1. Заголовок `h1` окрасьте в фиолетовый цвет (задайте его с помощью значений R:153, G:51, B:153 или #993399), добавив новое объявление к имеющемуся правилу `h1`. Обратите внимание: поскольку приведенное значение включает двузначные числа, можно применить его сокращенную версию (#939).
2. Заголовки `h2` окрасьте в светло-коричневый цвет (задайте его с помощью значений R:204, G:102, B:0, #cc6600 или #c60).
3. Фон всей страницы сделайте светло-зеленым (задайте его с помощью значений R:210, G:220, B:157 или #d2dc9d). На этом этапе самое время сохранить результат, просмотреть его в браузере и устранить недоработки, если фон и заголовки не отображаются в цвете.
4. Фон `header` сделайте белым с прозрачностью 50% (с помощью указания значений R:255, G:255, B:255, .5), чтобы сформировать намек на цвет фона.

5. Я уже добавила правило, отключающее подчеркивание ссылок (`text-decoration: none`), поэтому при отображении ссылок нам придется полагаться на цвет. Создайте правило, окрашивающее ссылки в такой же фиолетовый цвет, как и `h1` (#939).
6. Посещенные ссылки окрасьте в приглушенный фиолетовый цвет (#937393).
7. Когда мышь наведена на ссылку, ее текст должен окрашиваться в более яркий фиолетовый (#c700f2). И добавьте также белый цвет фона (#fff). Получится не-много похоже на подсветку ссылок при наведении на них мыши. Применяйте те же правила стиля, если ссылки находятся в фокусе.
8. При щелчке мышью (или касании экрана сенсорного устройства) добавьте белый цвет фона и окрасьте текст ярко-фиолетовым (#ff00ff). Убедитесь, что все псевдоклассы ссылки расположены в правильном порядке.

По завершении этой работы страница должна выглядеть так, как показано на рис. ЦВ-13.17. Позже мы добавим на эту страницу фоновые изображения, поэтому, если у вас есть желание поэкспериментировать с разными цветами для различных элементов, сделайте копию этого документа и присвойте ему новое имя. В процессе работы помните о том, что палитра цветов Google — это весьма удобное место для выбора цветов и их RGB-эквивалентов.

НЕ ЗАБЫВАЙТЕ ДОБАВЛЯТЬ СИМВОЛ

Не забывайте добавлять символ # перед шестнадцатеричными значениями. Правило не будет работать без этого символа.

ФОНОВЫЕ ИЗОБРАЖЕНИЯ

Ранее было показано, каким образом изображения с помощью элемента `img` добавляются к содержимому документа, но большинство декоративных изображений с помощью CSS добавляются на страницы и в элементы в качестве фоновых. «Украшения» типа мозаичных (иногда называемых «черепичными») фоновых рисунков являются сейчас основной частью представления, а не структуры. А ведь когда-то на сайтах использовались для этого мелкие фрагменты больших графических изображений, соединенных затем с помощью таблиц (рябь).

В этом разделе мы познакомимся с набором свойств, применяемых для размещения и перемещения фоновых изображений, начиная с основного свойства: `background-image`.

Свойство `background-image` (добавление фонового изображения)

Свойство `background-image` добавляет фоновое изображение к любому элементу. Основной его задачей является предоставление местоположения для файла изображения:

`background-image`

- **Значения:** `url(местоположение изображения)` | `none`.
- **Применение** — ко всем элементам.

- По умолчанию — none.
- Наследование — нет.

ХРАНИТЕЛЬ URL

Подходящим термином для этого «хранителя URL» служит функциональная запись. Этот же термин применяется для указания десятичных значений или процентов RGB.

правило находится во встроенной таблице стилей (элемент стиля в HTML-документе), то путь в составе URL должен формироваться относительно местоположения HTML-файла. Если CSS-правило находится во внешней таблице стилей, то путь к изображению задается относительно местоположения файла с расширением `css`.

Прекрасным альтернативным вариантом может служить предоставление относительных корневых URL-адресов изображений — это гарантирует, что фоновое изображение будет обнаружено независимо от местоположения правил стиля.

Остается напомнить, что корневой каталог обозначается слэшем (косой чертой) в начале URL-адреса. Например:

```
background-image: url(/images/background.jpg);
```

Недостатком, свойственным, впрочем, всем относительным URL-адресам сайтов, является тот факт, что этот адрес нельзя проверить локально (со своего компьютера), если не настроить его в качестве сервера.

Приведенные далее примеры демонстрируют фоновые изображения, примененные для всей страницы (`body`) и к одному элементу `blockquote` с использованием отступа и границы (рис. 13.18):

**СВОЙСТВА,
СВЯЗАННЫЕ С ФОНОМ:**

```
background-color
background-image
background-repeat
background-position
background-attachment
background-clip
background-size
background
```

```
body {
    background-image: url(star.png);
}
blockquote {
    background-image: url(dot.png);
    padding: 2em;
    border: 4px dashed;
}
```

В соответствии с заданным по умолчанию поведением свойства `background-image` изображение начинается в верхнем левом углу и заполняется по горизонтали и вертикали, пока весь элемент не будет им заполнен (далее будет показано, как это поведение можно изменить). Как и фоновые цвета, мозаичные фоновые изображения заполняют как область за контентом, так и дополнительное пространство вокруг контента и расширяются до внешнего края границы (если таковая имеется). Изменить область заливки фоном можно с помощью свойства `background-clip`.

Значение `background-image` является своего рода «хранителем» интернет-адреса (URL), который содержит местоположение изображения.

Интернет-адрес (URL) указывает на текущее местоположение правила CSS. Если

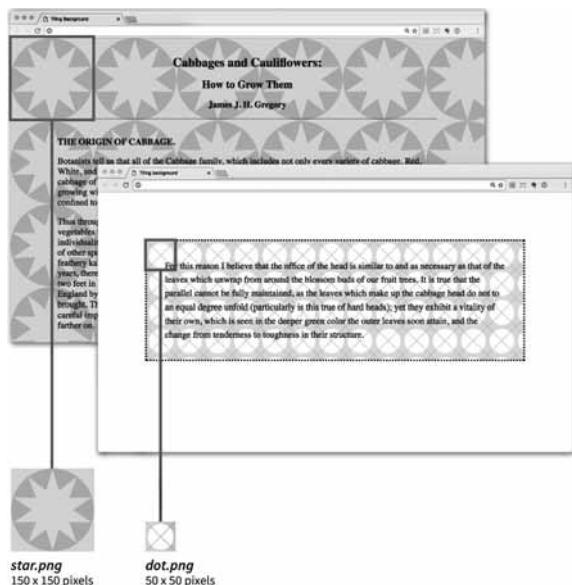


Рис. 13.18. Мозаичные фоновые изображения, добавленные с помощью свойства `background-image`

Если для элемента поддерживаются оба свойства: `background-color` и `background-image`, то изображение размещается поверх окрашенного фона. Из практических соображений настоятельно рекомендуется предоставить резервный фоновый цвет, схожий по оттенку с фоновым изображением, чтобы применить его в том случае, если изображение не сможет загрузиться.

Теперь вы можете попробовать свои силы в добавлении мозаичного фонового изображения на страницу, выполнив упражнение 13.2.

УПРАЖНЕНИЕ 13.2. ДОБАВЛЕНИЕ МОЗАИЧНОГО ФОНОВОГО ИЗОБРАЖЕНИЯ

При выполнении этого упражнения мы добавим в меню простое мозаичное фоновое изображение. Изображения, предоставленные для этого упражнения, находятся в каталоге `images`.

Добавьте к правилу стиля `body` объявление, которое сформирует с помощью изображения `bullseye.png` фоновую мозаику страницы. Укажите к файлу изображения путь относительно таблицы стилей (в нашем случае это текущий HTML-документ):

```
background-image: url(images/bullseye.png);
```

ВЫКЛАДЫВАНИЕ ФОНОВЫХ ИЗОБРАЖЕНИЙ МОЗАИКОЙ (ЧЕРЕПИЦЕЙ)

При обработке фоновых изображений учитывайте следующие рекомендации и советы:

- используйте простое изображение, которое не влияет на четкость текста над ним;
- всегда предоставляйте значение свойства `background-color`, соответствующее основному цвету фонового изображения. Если фоновое изображение почему-либо не отображается, то, по крайней мере, общий дизайн страницы не пострадает. Особенно важно, если цвет текста станет неразборчивым на заданном по умолчанию белом фоне браузера;
- как это привычно для Интернета, размер файла фоновых изображений должен быть настолько маленьким, насколько это возможно.

УКАЗЫВАЙТЕ СХОЖИЙ С ФОНОВЫМ ИЗОБРАЖЕНИЕМ ЦВЕТ ФОНА

Всегда указывайте схожий по оттенку с фоновым изображением цвет фона на тот случай, если фоновое изображение не загрузится.

Несложно, не правда ли? После сохранения и просмотра страницы в браузере, она имеет вид, показанный на рис. ЦВ-13.19.

Следует отметить, что файл `bullseye.png` представляет собой слегка прозрачную PNG-графику, поэтому она хорошо смотрится на фоне любого цвета. Попробуйте временно изменить значение свойства `background-color` для элемента `body`, добавив в стек второе объявление `background-color`, чтобы оно перекрыло предыдущее. Поэкспериментируйте с разными цветами, обращая внимание, каким образом сливаются круги. По завершении экспериментов удалите второе объявление, чтобы фон снова стал зеленым и вы могли бы использовать этот документ при выполнении следующих упражнений.

Свойство `background-repeat` **(повторяющееся фоновое изображение)**

Как мы видели на рис. 13.18, порядок расположения мозаичных изображений следующий: слева направо и сверху вниз. Это поведение можно изменить с помощью свойства `background-repeat`:

`background-repeat`

- **Значения:** `repeat` | `no-repeat` | `repeat-x` | `repeat-y` | `space` | `round`.
- **По умолчанию** — `repeat`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Если необходимо, чтобы фоновое изображение появилось только один раз, воспользуйтесь значением ключевого слова `no-repeat`:

```
body {
    background-image: url(star.png);
    background-repeat: no-repeat;
}
```

Также можно ограничить распространение мозаики только горизонтальным (`repeat-x`) или вертикальным (`repeat-y`) направлением:

```
body {
    background-image: url(star.png);
    background-repeat: repeat-x;
}
body {
    background-image: url(star.png);
    background-repeat: repeat-y;
}
```

На рис. 13.20 показаны примеры для каждого из рассмотренных значений ключевых слов. Обратите внимание, что во всех примерах мозаика начинается в верхнем левом углу элемента (или в окне браузера, если изображение применяется к элементу `body`). В следующем разделе я покажу, как изменить этот порядок.

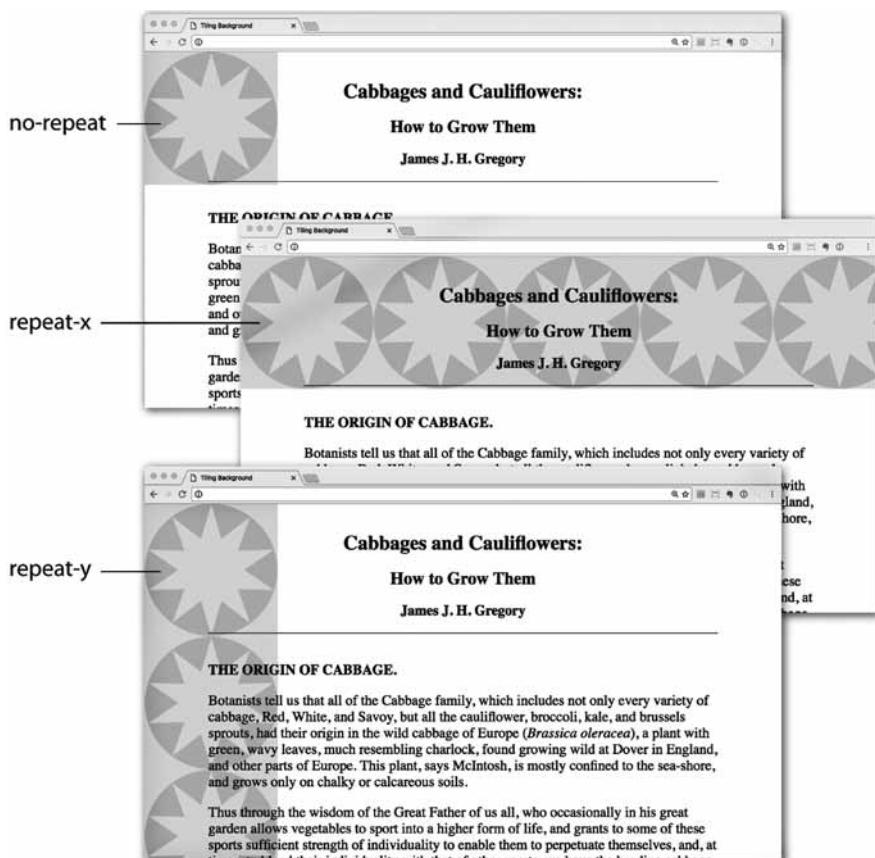


Рис. 13.20. Отключение автоматического формирования мозаики с помощью свойства `no-repeat` (вверху), формирование «черепиц» в горизонтальном направлении с помощью свойства `repeat-x` (в центре) и формирование «черепиц» в вертикальном направлении с помощью свойства `repeat-y` (внизу)

Оставшиеся значения ключевых слов: `space` и `round` — приводят к заполнению доступной области рисования фона четное число раз.

Если для параметра `background-repeat` установлено значение `space`, браузер рассчитывает, сколько фоновых изображений может поместиться по ширине и высоте области фона, затем добавляет одинаковое пространство между каждым изображением. В результате получаются четные строки и столбцы без усечения изображений (рис. 13.21, вверху).

Ключевое слово `round` заставляет браузер сжимать фоновое изображение по горизонтали и вертикали (необязательно пропорционально), что приводит к его размещению в фоновой области четное число раз (рис. 13.21, внизу).

О ПОДДЕРЖКЕ БРАУЗЕРАМИ КЛЮЧЕВЫХ СЛОВ `SPACE` И `ROUND`

Браузер Internet Explorer 8 и его ранние версии не поддерживают ключевые слова `space` и `round` для свойства `background-repeat`.



Рис. 13.21. Примеры применения ключевых слов `space` и `round` для свойства `background-repeat`. Пример для `space` был бы менее неуклюжим, если бы цвет фона соответствовал изображению, но я оставила его белым, чтобы продемонстрировать, как работает значение `space`

Рассмотрим процесс формирования некоторых повторяющихся фоновых покрытий, выполнив [упражнение 13.3](#).

УПРАЖНЕНИЕ 13.3. УПРАВЛЕНИЕ НАПРАВЛЕНИЕМ ФОРМИРОВАНИЯ МОЗАИКИ

Разместим на странице «Summer Menu» несколько более изысканную мозаику, для чего добавим мозаичный фон только вдоль верхнего края элемента `header`.

- С помощью правила `header` добавим изображение `purpledot.png` (фиолетовые кружки) и установим его повторяющимся только в горизонтальном направлении:

```
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;
}
```

- Сохраните файл и просмотрите его в окне браузера. На рис. ЦВ-13.22 показано, какой получается при этом вид. Я рекомендую изменять размер окна браузера, расширяя и сужая его, и обратить при этом внимание на положение фонового рисунка. Оно остается закрепленным слева, не так ли? Но как отрегулировать

его положение? Попробуйте изменить правило стиля таким образом, чтобы фиолетовые кружки повторялись только в вертикальном направлении, а затем сделайте, чтобы вообще не было повторов (установите значение `repeat-x` и сохраните документ по завершении работы).

3. И, наконец, попробуйте повторяющиеся значения `space` и `round` элемента `body` для фонового изображения и просмотрите, нравится ли вам полученный эффект. Обратите внимание, что «черепицы» равномерно расположены внутри основного раздела документа (`body`) документа, а не только в области просмотра, поэтому в нижней части браузера можно заметить обрезанные круги. Удалите объявление `background-repeat` — тогда документ вернется к заданному по умолчанию для последующих упражнений значению `repeat`:

```
body {  
...  
background-repeat: space;  
}
```

Свойство `background-position` (положение фонового изображения)

Свойство `background-position` задает положение фонового исходного изображения. Вы можете представить себе исходное изображение как размещенное на фоне первым, начиная от которого располагается остальная мозаика:

`background-position`

- **Значения:** `размер в длину | процентные значения | left | center | right | top | bottom`.
- **По умолчанию:** `0% 0%` (то же самое, что и `left top`)
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Расположение исходного изображения определяется описывающими его горизонтальными и вертикальными значениями. Существует множество способов сделать это:

- **позиционирование с помощью ключевых слов** — применение значений ключевых слов (`left`, `right`, `top`, `bottom` и `center`) приводит к позиционированию исходного изображения относительно внешних краев видимой области фона. Например, значение ключевого слова `left` позиционирует изображение по отношению к левому краю области фона. Заданная по умолчанию исходная позиция соответствует значениям `left` и `top`.

Ключевые слова обычно применяются парами, как в следующих примерах:

```
background-position: left bottom;  
background-position: right center;
```

Ключевые слова могут отображаться в любом порядке. Если указать только одно ключевое слово, предполагается, что отсутствующим ключевым словом будет `center`. Таким образом, применение `background-position: right` приводит к тому же эффекту, что и применение `background-position: right center`;

- размер в длину — задание положения с использованием величин длины, таких, как пиксели или em, указывает величину смещения от верхнего левого угла элемента (видимой области фона) до верхнего левого угла исходного фонового изображения. При указании значений длины горизонтальное значение всегда приводится вначале. Задание отрицательных значений разрешено и приводит к «выползанию» изображения за пределы видимой фоновой области.

В следующем примере верхний левый угол изображения располагается на расстоянии, равном 200 пикселов от левого края и 50 пикселов — вниз от верхнего края элемента (или, более точно, от края заданного по умолчанию отступа):

```
background-position: 200px 50px;
```

- процентные значения — значения, заданные в процентах, представлены горизонтальными/вертикальными парами, причем 0% 0% соответствует верхнему левому углу, а 100% 100% — нижнему правому углу. Как и в случае со значениями длины, горизонтальное значение всегда приводится первым.

Важно отметить, что процентное значение относится как к области холста (области расположения фона), *так и к самому изображению*. Так, горизонтальное значение, равное 25%, помещает точку, отстоящую на 25% от левого края изображения, в точку, которая находится на 25% от левого края области расположения фона. Вертикальное значение 100% приводит к позиционированию нижнего края изображения у нижнего края области позиционирования:

ГОРИЗОНТАЛЬНОЕ ЗНАЧЕНИЕ ВСЕГДА ПРИВОДИТСЯ ПЕРВЫМ

При предоставлении значения длины или процентного горизонтальное измерение всегда приводится первым.

```
background-position: 25% 100%;
```

Как и в случае с ключевыми словами, если указать только одно процентное значение, второе считается равным 50% (выполняется центрирование).

СМЕЩЕНИЕ ФОНОВОГО ИЗОБРАЖЕНИЯ ОТ КРАЯ ХОЛСТА

Спецификация CSS3 также предлагает для свойства `background-position` четырехчастный синтаксис, позволяющий указывать смещение (`offset`) изображения (по длине или в процентах) от определенного края области расположения фона с помощью соответствующих ключевых слов (`edge-keyword`):

```
background-position:  
  edge-keyword offset  
  edge-keyword offset
```

В следующем примере исходное изображение располагается в 50 пикселях от правого края и в 50 пикселях от нижней части области расположения фона:

```
background-position:  
  right 50px bottom 50px;
```

Такой четырехчастный синтаксис не поддерживается в IE8 и его более ранних версиях, Safari и iOS Safari 6 и их более ранних версиях, а также в Android 4.3 и его более ранних версиях.

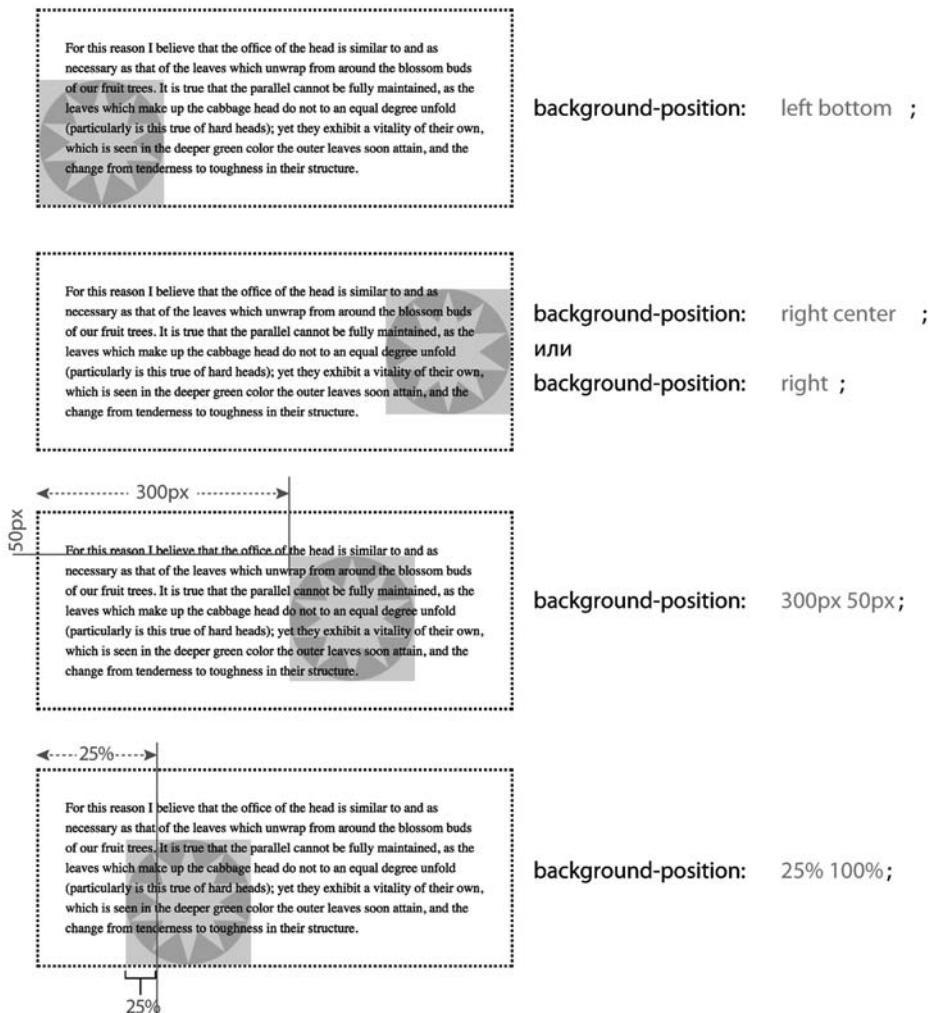


Рис. 13.23. Позиционирование неповторяющегося фонового изображения. Если бы эти фоновые изображения были заданы, как повторяющиеся (`repeat`), они бы располагались влево и вправо и/или вверх и вниз от исходных положений

На рис. 13.23 показаны результаты для каждого из рассмотренных примеров свойства `background-position` со свойством `background-repeat`, установленным в `no-repeat` для лучшего понимания происходящего. Вы можете с помощью свойства `background-position` поместить исходное изображение в нужную позицию и формировать от него мозаику либо в обоих направлениях, либо только по горизонтали или вертикали. Если изображение сформировано как мозаика, положение исходного изображения может не быть вполне очевидным, но использование свойства `background-position` дает вам возможность распространять мозаику от точки, не обязательно расположенной у левого края изображения. Подобный подход может применяться для получения центрированного и симметричного фона.

Свойство *background-origin* (начальное положение фонового изображения)

На рис. 13.23 видно, что если фоновое изображение размещено в углу области расположения фона (холста), оно помещается внутри его границы (только повторяющиеся изображения могут «залезть» под эту границу). Такая позиция задана по умолчанию, но ее можно изменить с помощью свойства *background-origin*:

background-origin

- **Значения:** `border-box` | `padding-box` | `content-box`.
- **По умолчанию** — `padding-box`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Свойство *background-origin* определяет границы области расположения фона так же, как свойство *background-clip* определяет область рисования фона. Можно установить границы как `border-box` (тогда исходное изображение помещается под внешним краем границы), как `padding-box` (вне края отступа, сразу же внутри границы) или как `content-box` (фактическая область контента элемента).

Эти моменты станут вам понятнее после ознакомления с блочной моделью (в главе 14). На рис. 13.24 показаны результаты применения каждого из ключевых слов.

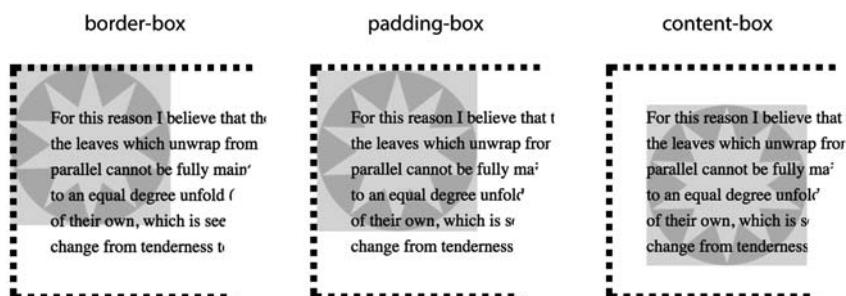


Рис. 13.24. Примеры использования ключевых слов для свойства *background-origin*

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Свойство *background-origin* не поддерживается *Internet Explorer 8* и его более ранними версиями.

Прежде чем перейти к рассмотрению оставшихся свойств фона, давайте потренируемся с размещением фоновых изображений, выполнив упражнение 13.4.

УПРАЖНЕНИЕ 13.4. РАЗМЕЩЕНИЕ ФОНОВЫХ ИЗОБРАЖЕНИЙ

В этом упражнении мы поэкспериментируем с расположением фонового изображения в нашем меню. Сначала мы кое-что подправим по части расположения уже имеющихся фоновых изображений, затем поменяем их на другие и еще немногого поэкспериментируем. Откройте документ `summer-menu.html`, где в элементах `body` и `header` должны находиться повторяющиеся элементы мозаики.

- Поскольку основные элементы меню выровнены по центру, неплохо, если бы фоновые рисунки также позиционировались по центру. Добавьте соответствующее объявление к правилам `body` и `header`, затем сохраните документ и просмотрите результат в браузере:

```
background-position: center top;
```

Разницу можно и не заметить, если снова не изменить размер окна браузера. Теперь блок мозаики расположен в центре области отображения и более или менее одинаково относительно обоих ее краев, а не только относительно правого края, как было раньше.

- Для большего эффекта измените значения свойства `background-position`, чтобы фиолетовые кружки находились вдоль нижнего края элемента `header` (значение `center bottom`). Получилось не очень хорошо, поэтому вернем значение `top`. Затем переместите файл `bullseye.png` вниз на 200 пикселов (значение `center 200px`). Обратите внимание, что рисунок все еще занимает весь экран — исходное изображение переместилось вниз, но фон по-прежнему установлен так, что мозаика распространяется по всем направлениям. На рис. ЦВ-13.25 (слева) показан результат выполненных изменений.
- Результат выглядит хорошо, но давайте пока избавимся от фона для элемента `body` и рассмотрим один трюк. В процессе разработки страницы я скрываю ненужные на тот момент стили в комментариях, но не удаляю их полностью. Тогда мне не нужно их запоминать или вводить потом снова — надо лишь удалить указатели комментариев, и стили возвращаются. Ну а уже по завершении разработки и при переходе к публикации неиспользуемые стили действительно убираю, что уменьшает размер файла.

Посмотрите, как скрываются объявления путем представления их в виде CSS-комментариев:

```
body {  
...  
    background-color: #d2dc9d;  
/* background-image: url(images/bullseye.png);  
background-position: center 200px; */  
}
```

- А теперь добавьте на фон страницы изображение `blackgoose.png` (также полупрозрачное изображение в формате PNG). Установите отсутствие повторений и центрируйте его в верхней части страницы:

```
background-image: url(images/blackgoose.png);  
background-repeat: no-repeat;  
background-position: center top;
```

Просмотрите полученный вариант в окне браузера и понаблюдайте, каким образом при прокрутке страницы фон прокручивается вверх вместе с контентом.

- Мне бы хотелось, чтобы вы почувствовали влияние на изображение различных ключевых слов и числовых значений, определяющих позиционирование. Попробуйте каждый из предлагаемых далее вариантов и посмотрите, как он выглядит в окне браузера. Обязательно прокрутите страницу и посмотрите, что происходит. Обратите внимание, что при указании процентного значения или

ключевого слова для вертикальной позиции она формируется по высоте всего документа, а не только по высоте окна браузера. Вы также можете попробовать и собственные варианты:

```
background-position: right top;
background-position: right bottom;
background-position: left 50%;
background-position: center 100px;
```

6. Оставьте изображение, размещенным как `center 100px`, чтобы подготовиться к следующему упражнению. Страница должна иметь вид, показанный на рис. ЦВ-13.25 (справа).

Свойство `background-attachment` (прикрепление фонового изображения)

В *упражнении 13.4* я предлагала вам прокрутить страницу и посмотреть, что происходит с фоновым изображением. Как и ожидалось, оно прокручивается вместе с документом от верхней части окна браузера, что является заданным по умолчанию поведением. Однако можно применить свойство `background-attachment` и освободить фон от контента, позволяя ему оставаться фиксированным в одной позиции во время прокрутки остальной части контента.

`background-attachment`

- **Значения:** `scroll` | `fixed` | `local`.
- **По умолчанию** — `scroll`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

С помощью свойства `background-attachment` можно выбрать, будет ли фоновое изображение прокручиваться с контентом или останется в фиксированном положении (прикрепленным). Если для свойства `background-attachment` выбрано значение `fixed`, изображение остается в одной позиции относительно окна просмотра браузера (в отличие от заполняемого им элемента). Вы поймете, что я имею в виду, через минуту (также попрактикуйтесь и сами, выполнив *упражнение 13.5*).

В следующем примере большое фоновое изображение помещается на заднем плане (фоне) всего документа (элемент `body`). По умолчанию при прокрутке документа изображение также прокручивается, перемещаясь вверх и за пределы страницы, как показано на рис. 13.26. Однако, если установить для свойства `background-attachment` значение `fixed`, изображение останется там, где размещалось изначально, а текст будет прокручиваться поверх него:

```
body {
    background-image: url(images/bigstar.gif);
    background-repeat: no-repeat;
    background-position: center 300px;
    background-attachment: fixed;
}
```

Значение `local`, которое появилось в CSS3, полезно применять, если элемент имеет собственный механизм прокрутки. Вместо прокрутки, выполняемой с помощью ползунка области просмотра, значение `local` приводит к фиксации фонового изображения в соответствии с контентом элемента прокрутки. Это ключевое слово не поддерживается в IE8 и более ранних версиях, а также может привести к проблемам при просмотре страницы в мобильных браузерах.



Крупное неповторяющееся фоновое изображение, находящееся в разделе `body` документа.

`background-attachment: scroll;`

По умолчанию фоновое изображение прикрепляется к элементу `body` и прокручивается вместе со страницей при прокрутке контента.

`background-attachment: fixed;`

Если свойству `background-attachment` присвоено значение `fixed`, изображение остается на своем месте относительно области просмотра браузера и не прокручивается вместе с контентом.

Рис. 13.26. Предотвращение прокрутки фонового изображения с помощью свойства `background-attachment`

УПРАЖНЕНИЕ 13.5. ФИКСИРОВАННОЕ ПОЛОЖЕНИЕ ФОНОВОГО ИЗОБРАЖЕНИЯ

В процессе последнего изменения меню веб-страницы нашего кафе в качестве фонового изображения мы применили крупное неповторяющееся изображение логотипа. Оставим его, но применим свойство `background-attachment` для сохранения местоположения изображения при прокрутке страницы:

```
body {
    background-image: url(images/
blackgoose.png);
    background-repeat: no-repeat;
    background-position: center
    100px;
    background-attachment: fixed;
}
```

Сохраните документ, откройте его в браузере и попробуйте прокрутить. Фоновое изображение остается в области просмотра браузера. Неплохо, не так ли?

Для получения дополнительной информации посмотрите, что происходит при фиксации точечного шаблона в разделе `header` (подсказка: шаблон остается на том же месте, но внутри раздела `header`. Если раздел `header` выходит из поля зрения, то же самое происходит и с фоном).

Свойство ***background-size*** (размер фонового изображения)

Итак, нам осталось рассмотреть только одно свойство фонового изображения, прежде чем перейти к описанию сокращенного свойства `background`. До сих пор представленные фоновые изображения отображались в соответствии с их собственными фактическими размерами. Размер изображения можно изменить, применяя свойство `background-size`:

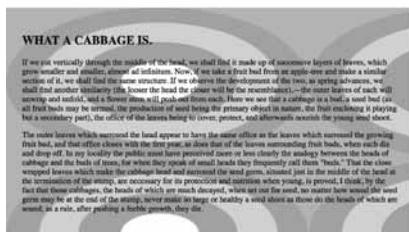
background-size

- **Значения:** `длина | процентное значение | auto | cover | contain.`
- **По умолчанию** — `auto`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

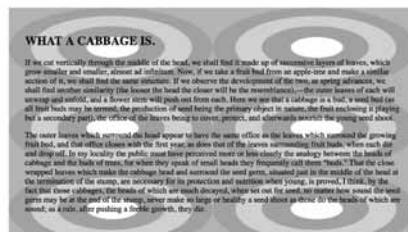
Существует несколько способов указать размер фонового изображения. Возможно, проще всего указать эти размеры в единицах длины, таких, как пиксели или `em`. Как обычно, когда заданы два значения, первым записывается горизонтальный размер. Если указать только одно значение, оно используется в качестве размера по горизонтали, а для значения по вертикали устанавливается значение `auto`.

В следующем примере размер фонового изображения `target.png`, исходный размер которого составляет 300 на 300 пикселов, изменяется (рис. 13.27):

```
header {
    background-image: url(images/target.png);
    background-size: 600px 150px;
}
```



`background-size: 600px 300px;`



`background-size: 50% 10em;`

Рис. 13.27. Изменение размера исходного фонового изображения (вверху) с помощью задания единиц длины (внизу слева) и процентных значений (внизу справа)

Значения, заданные в процентах, рассчитываются на основе размеров области расположения фона, которая по умолчанию прилежит к внутреннему краю границы, но

ее положение может быть изменено применением свойства `background-origin`, что следует иметь в виду. Таким образом, горизонтальное значение 50% не означает, что изображение будет отображено на половину его ширины, — оно просто займет 50% ширины области расположения фона (см. рис. 13.27, *внизу справа*). Напомню, что горизонтальное значение идет первым. Можно соединять значения, выраженные в процентах и в единицах длины, как показано в следующем примере:

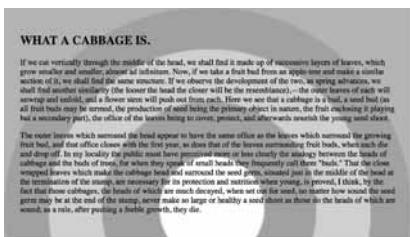
```
header {
    background-image: url(images/target.png);
    background-size: 50% 10em;
}
```

Ключевое слово `auto` изменяет размер изображения в любом направлении, необходимом для поддержания его пропорций. Растревые изображения, такие, как GIF, JPEG и PNG, характеризуются наличием внутренней пропорциональности, поэтому они всегда останутся пропорциональными, если для какого-либо одного значения их размера будет установлено значение `auto`. Некоторые изображения, такие, как изображения SVG и CSS-градиенты, не отличаются внутренними пропорциями. Тогда значение `auto` устанавливает их ширину или высоту так, чтобы они стали равны 100% ширины или высоты области расположения фона.

Ключевые слова `cover` и `contain`, появившиеся в CSS3, определяют интересные эффекты. В случае установления размера фонового изображения с помощью значения `cover` браузер изменяет его размер так, чтобы захватить им всю область расположения фона:

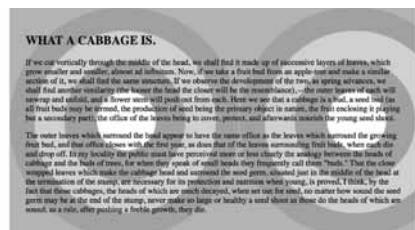
```
div#A {
    background-image: url(target.png);
    background-size: cover; }
```

При этом мы получим только одно изображение, заполняющее весь элемент, и, вполне вероятно, если пропорции изображения и области позиционирования не будут соответствовать друг другу, части изображения не совпадут с областью позиционирования (рис. 13.28, слева).



`background-size: cover;`

Фоновая область элемента заполнена изображением, при этом само изображение сохраняет свои пропорции, даже если выходит за область позиционирования.



`background-size: contain;`

Размеры изображения изменены пропорционально, поэтому оно полностью помещается в элемент. При этом может хватить места и для формирования мозаики (как здесь и показано).

Рис. 13.28. Примеры использования в качестве значений свойства `background-size` ключевых слов `cover` (слева) и `contain` (справа)

Напротив, значение `contain` задает изображению такие размеры, чтобы оно могло полностью разместиться в области позиционирования либо по ширине, либо по высоте (в зависимости от собственных пропорций):

```
div#B {
    background-image: url(target.png);
    background-size: contain; }
```

ИЗМЕНЯЙТЕ РАЗМЕРЫ РАСТРОВОГО ИЗОБРАЖЕНИЯ ОСТОРОЖНО

При изменении размера растрового изображения, такого, как GIF или PNG, вы рискуете получить размытое и пикселизированное изображение. Поэтому осторожно.

Изображение окажется полностью видимым и будет расположено внутри фоновой области (рис. 13.28, справа). Если останется место, фоновое изображение станет повторяться, если только свойству `background-repeat` не присвоено значение `no-repeat`.

Краткое свойство `background`

Для указания всех стилей фона в одном объявлении можно применять удобное свойство `background`:

`background`

- **Значения:** `background-color` `background-image` `background-repeat` `background-attachment` `background-position` `background-clip` `background-origin` `background-size`.
- **По умолчанию** — смотрите индивидуальные свойства каждого значения.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Значением свойства `background` является перечень величин, которые предоставляются для отдельных рассмотренных ранее свойств фоновых изображений. Например, одно это правило:

```
body { background: white url(star.png) no-repeat right top fixed; }
```

заменяет пять отдельных объявлений:

```
body {
    background-color: white;
    background-image: url(star.png);
    background-repeat: no-repeat;
    background-position: right top;
    background-attachment: fixed;
}
```

Все значения свойств для `background` необязательны и отображаются в произвольном порядке. Единственным ограничением является необходимость при задании координаты для свойства `background-position` сначала указывать значение «по горизонтали», а затем сразу же — значение «по вертикали». Как и с любым сокращенным свойством, помните, что если какое-либо значение не указано, оно будет

ОСТЕРЕГАЙТЕСЬ ПЕРЕОПРЕДЕЛЕНИЙ

Краткое свойство `background` весьма эффективно, но применяйте его с осторожностью. Речь об особенностях кратких (сокращенных) свойств шла и ранее, но сейчас я хочу подчеркнуть этот момент еще раз.

Поскольку свойство `background` является сокращенным, при пропуске в его перечне значений какого-либо значения оно будетброшено к значению, заданному для него по умолчанию. Будьте осторожны, чтобы случайно не переопределить с помощью сокращенного правила, которое возвратит ваши настройки к заданным по умолчанию значениям, правила стилей, заданные ранее в таблице стилей.

В следующем примере фоновое изображение `dots.gif` не применяется к элементам `h3`, поскольку во втором элементе `background` значение для `background-image` опущено. В результате значение для `background-image` устанавливается равным `none`:

```
h1, h2, h3 {  
    background: red url(dots.gif)  
    repeat-x;  
}  
h3 {  
    background: green;  
}
```

Для переопределения установленных ранее свойств лучше пользоваться не краткими, а специфицированными свойствами `background`. Например, если в приведенном примере изменяется только цвет фона для элементов `h3`, правильнее применить свойство `background-color`.

брошено до заданного по умолчанию (обратитесь к врезке «*Остерегайтесь переопределений*» для получения дополнительной информации).

При выполнении *упражнения 13.6* вы сможете с помощью свойства `background` преобразовать длинный перечень свойства фона в одно объявление.

УПРАЖНЕНИЕ 13.6. ПРЕОБРАЗОВАНИЕ С ПОМОЩЬЮ СОКРАЩЕННОГО СВОЙСТВА

Это весьма простое упражнение. Замените все связанные с фоном объявления в разделе `body` меню бистро одним объявлением 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;  
}
```

Выполните аналогичное действие для элемента `header`.

Использование нескольких фоновых изображений

В CSS3 появилась возможность прикреплять к одному элементу несколько фоновых изображений. Для этого следует поместить несколько значений свойства `background-image` в разделенный запятыми список. Дополнительные значения связанных с фоновыми изображениями свойств также попадают в разделенные запятыми списки: первое указанное значение относится к первому изображению, второе значение — ко второму и т. д.

Хотя CSS-объявления обычно работают по правилу «последний выигрывает», для нескольких фоновых изображений это не так: указанное в списке последним, располагается в самом низу, а каждое предшествующее ему в списке изображение располагается выше. Можно представить эти изображения как слои Photoshop в том смысле, что они отображаются в порядке появления в списке. Иными словами, изображение, определенное первым значением, будет идти первым, а другие выстраиваются в очередь за ним в порядке записи:

```
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, можно пользоваться преимуществом сокращенного свойства `background`, что упростит запись правила. Так, следующее свойство имеет три ряда разделенных запятыми значений:

```
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. ЦВ-13.29 показан результат выполнения этого правила: большой оранжевый кружок с цифрой «1» расположен в верхнем углу, голубой кружок с цифрой «2» — центрирован по вертикали и по горизонтали, а зеленый кружок с цифрой «3» находится в правом нижнем углу. Все три эти фоновые изображения делят область расположения фона одного элемента `body`. Попробуйте теперь самостоятельно выполнить *упражнение 13.7*.

УПРАЖНЕНИЕ 13.7. НЕСКОЛЬКО ФОНОВЫХ ИЗОБРАЖЕНИЙ

В этом упражнении мы поэкспериментируем с обработкой нескольких фоновых изображений (убедитесь, что у вас не используется старая версия IE, иначе упражнение нельзя будет выполнить).

Желательно, чтобы мозаичные группы фиолетовых кружков в разделе `header` располагались вдоль его левой и правой сторон. В нашем распоряжении также имеется изображение небольшого силуэта гуся (`goose-shadow.png`), который может милостиво прогуливаться по нижней части заголовка. Сделаем наш пример удобным для не поддерживающих несколько фоновых изображений браузеров (версии IE8 и более ранние), для чего предоставим резервное объявление только с одним изображением и отделим объявление `background-color`, чтобы избежать переопределения. Если ваш браузер отличается от IE8 в лучшую сторону, запасной вариант вам не понадобится.

Как показано в следующем примере, в одном заголовке помещаются три изображения: фиолетовые кружки слева и справа, и изображение гуся — внизу:

```
header {  
    ...  
    background: 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);  
}
```

На рис. ЦВ-13.30 показан окончательный результат нашего упражнения. Да, предыдущая версия веб-страницы мне нравилась больше, но зато вы уяснили основную идею.

ВСЕ ЦВЕТА РАДУГИ (ГРАДИЕНТЫ)

Градиент — это переход от одного цвета к другому, иногда через несколько цветов. Раньше для размещения на веб-странице градиента необходимо было в программе редактирования изображений создавать нужный градиент и добавлять полученное изображение с помощью CSS.

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Браузер *Internet Explorer 8* и более ранние его версии не поддерживают наличие нескольких фоновых изображений и полностью игнорируют любое объявление фона с более чем одним значением. Чтобы преодолеть эту проблему, выберите для элемента одно свойство `background-image` в качестве запасного варианта для браузера IE и других браузеров, не поддерживающих такую возможность, а затем укажите несколько правил `background`, которые его переопределяют:

```
body {  
    /* для не поддерживающих браузеров */  
    background: url(image_fallback.  
    png) top left no-repeat  
    /* несколько фонов */  
    background:  
        url(image1.png) left top  
    no-repeat,  
        url(image2.png) center center  
    no-repeat,  
        url(image3.png) right bottom  
    no-repeat;  
    /* фоновый цвет */  
    background-color: papayawhip;  
}
```

ГРАДИЕНТЫ

Градиенты — это изображения, которые браузеры генерируют «на лету». Вы можете использовать их в качестве фоновых изображений.

Теперь можно указывать цветовые градиенты, используя только правила CSS, дающие браузеру задание по отображению цветовых смесей. И, хотя градиенты указаны с помощью кода, они, тем не менее, являются *изображениями*. Просто они генерируются «на лету». Градиентное изображение не имеет собственного размера или пропорций — размер соответствует элементу, к которому он применяется. Градиенты могут применяться, куда может быть помещено изображение. При этом используются свойства: `background-image`, `border-image` и `list-style-image`. Все примеры этой главы основаны на использовании свойства `background-image`.

Различают два типа градиентов:

- *линейные градиенты* изменяют цвета вдоль линии — от одного края элемента к другому;
- *радиальные градиенты* начинаются в точке и распространяются от нее наружу в форме круга или эллипса.

Линейные градиенты: *linear-gradient()*

Запись `linear-gradient()` определяет угол линии градиента и одну или несколько точек вдоль той линии, где расположен чистый цвет (*стоп-цвета*). Для задания цветов можно использовать их названия или любые определяющие их числовые значения, рассмотренные в этой главе ранее, включая прозрачность. Угол линии градиента указывается в градусах (`ndeg`) или с помощью ключевых слов. При выборе значений в виде градусов величина `0deg` указывает направление вверх, и положительные углы отсчитываются от него по часовой стрелке, так что значение `90deg` указывает направление вправо. Поэтому, если необходимо, например, перейти от цвета морской волны (`aqua`) на верхнем конце линии к зеленому (`green`) на нижнем, воспользуйтесь значением `180deg`:

```
background-image: linear-gradient(180deg, aqua, green);
```

Ключевые слова описывают направление с шагом, равным 90° : `to top`, `to right`, `to bottom`, `to left`. Так, градиент, заданный значением `180deg`, можно указать с помощью ключевого слова `to bottom`. Полученный при этом результат показан на рис. ЦВ-13.31, *вверху*:

```
background-image: linear-gradient(to bottom, aqua, green);
```

Синтаксис `«to»` можно применять и для указания углов. Следующий градиент нарисован из нижнего левого угла в верхний правый угол. Результирующий угол градиента, проведенного между этими углами, определяется соотношением сторон окна:

```
background-image: linear-gradient(to top right, aqua, green);
```

В следующем примере градиент направлен слева направо (`90deg`) и включает третий цвет — оранжевый, который появляется на 25% пути по линии градиента (рис. ЦВ-13.31, *в центре*). Обратите внимание, что расположение точки стоп-цвета указано после задания значения цвета. Для указания такой точки можно применять процентные значения или любой вариант задания длины. Для первого и последнего

стоп-цветов указание позиций не требуется, поскольку они по умолчанию установлены равными на 0% и 100%, соответственно:

```
background-image: linear-gradient(90deg, yellow, orange 25%, purple);
```

Разумеется, распространение градиента не ограничено прямыми углами. Укажите любое значение угла, в соответствии с направлением которого формируется линейный градиент. Вы также можете задать столько цветов, сколько пожелаете. Если позиции стоп-цветов не указаны, цвета распределяются равномерно по длине линии градиента. Если поместить стоп-цвет на линии градиента последним (например, как синий при значении, равном 50%, в следующем примере), соответствующий ему цвет продолжится до конца линии градиента (рис. ЦВ-13.31, *внизу*):

```
background-image: linear-gradient(54deg, red, orange, yellow,  
green, blue 50%);
```

Приведенные на рис. ЦВ-13.31 примеры довольно-таки яркие, но, если выбрать другие цвета и соответствующие им стоп-цвета правильным образом, градиенты могут стать хорошим способом придания элементам небольшого затемнения и трехмерного вида. Кнопка, показанная на рис. 13.32, использует фоновый градиент для получения трехмерного изображения без использования графики:



Рис. 13.32. Изображение трехмерной кнопки, выполненной только с помощью CSS

```
a.button-like {  
    background: linear-gradient(to bottom, #e2e2e2 0%, #bdbdbd 50%,  
    #d1d1d1 51%, #fefefe 100%);  
}
```

На этом мы завершим краткий обзор линейных градиентов. Здесь я осветила лишь некоторые особенности их поведения и свойств, поэтому для более углубленного понимания связанных с ними процессов обратитесь к ресурсам, приведенным во врезке «Дополнительные источники информации». А теперь пришло время перейти к радиальным градиентам.

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ ИНФОРМАЦИИ

Наиболее подробный обзор синтаксиса CSS-градиента, который мне известен, содержится в книге Эрика Мейера (Eric Meyer) «Colors, Backgrounds, and Gradients» (O'Reilly). Схожую информацию можно найти в книге «CSS: The Definitive Guide» Эрика Мейера (Eric Meyer) и Эстель Вейль (Estelle Weyl) (также O'Reilly).

В Интернете доступны следующие обзоры и учебные пособия:

- «CSS Gradients» Криса Коудера (Chris Coyier): css-tricks.com/css3-gradients/;
- «Using CSS Gradients» — на сайте на MDN Web Docs: developer.mozilla.org/en-US/docs/Web/CSS/CSS/Images/Using_CSS_gradients;
- «CSS3 Gradients», часть электронной книги «CSS Mine» Мартина Мичалека (Martin Michalek): www.cssmine.com/ebook/css3-gradients.

Радиальные градиенты: *radial-gradient()*

Радиальные градиенты, как следует из их названия, исходят из точки и распространяются в виде круга вдоль *градиентного луча* (это как бы градиентная линия, но всегда направленная от центра наружу). Как минимум, радиальный градиент требует двух цветов, как показано в следующем примере:

```
background-image: radial-gradient(yellow, green);
```

По умолчанию градиент заполняет доступную фоновую область, а его центр располагается в центре элемента (рис. ЦВ-13.33). В результате мы получим эллипс, если содержащий градиент элемент является прямоугольником, и круг, если таким элементом служит квадрат.

Выглядит это достаточно элегантно, причем далеко не всегда нужно соглашаться на заданный по умолчанию вариант. Запись *radial-gradient()* позволяет указать форму, размер и центральное положение градиента:

- *форма* — для большинства случаев форма радиального градиента зависит от формы элемента или явного размера, который к нему применен, однако также можно указать форму, используя ключевые слова *circle* или *ellipse*. Если задать градиент в виде круга: *circle* (без иногда противоречивых спецификаций размера), он останется круглым, даже если располагается в прямоугольном элементе (рис. ЦВ-13.34, *вверху*):

```
background-image: radial-gradient(circle, yellow, green);
```

- *размер* — размер радиального градиента может задаваться в единицах длины или в процентах, применяемых к лучу градиента, или с помощью ключевых слов. Если указать только одно значение длины, оно будет воспринято как для ширины, так и для высоты, в результате чего получится круг. Если указать два значения длины: первое должно быть горизонтальным значением, а второе — вертикальным (рис. ЦВ-13.34, *в центре*). Для эллипсов также можно указывать процентные значения или смешивать процентные значения со значениями длины:

```
background-image: radial-gradient(200px 80px, aqua, green);
```

Используются и четыре ключевых слова: *closest-side*, *closest-corner*, *farthest-side* и *farthest-corner* — которые устанавливают длину градиентного луча относительно точек на содержащем его элементе;

- *расположение* — по умолчанию центр градиента располагается в точке, заданной значениями *center center*, но это можно изменить, используя синтаксис позиционирования, аналогичный синтаксису свойства *background-position*, разве что ему предшествует ключевое слово *at*, как показано в следующем примере (рис. ЦВ-13.34, *внизу*). Обратите внимание, что в пример включен дополнительный стоп-цвет для оранжевого цвета на отметке 50%:

```
background-image: radial-gradient(farthest-side at right bottom, yellow, orange 50%, purple);
```

Повторяющиеся градиенты

Для повторения градиента примите записи `repeating-linear-gradient()` или `repeating-radial-gradient()`. При этом используется такой же синтаксис, что и для отдельных градиентов, но добавление префикса `repeating-` заставляет градиент бесконечно повторять цвета в обоих направлениях. Повторяющиеся градиенты обычно применяются для формирования интересных полосатых узоров. В следующем простом примере градиент от белого цвета к серебристому (светло-серому) повторяется каждые 30 пикселов, потому что стоп-цвет для серебряного цвета установлен на 30 пикселов (рис. ЦВ-13.35, вверху):

```
background: repeating-linear-gradient(to bottom, white, silver 30px);
```

Следующий пример формирует диагональный рисунок, состоящий из оранжевых и белых полос (рис. ЦВ-13.35, внизу). Края здесь острые, поскольку белая полоса начинается точно в той же точке, где заканчивается оранжевая (при 12 пикселях) без затухания:

```
background: repeating-linear-gradient(45deg, orange, orange 12px, white 12px, white 24px);
```

Поддержка браузерами и префиксы поставщиков

Все основные браузеры начали добавлять поддержку стандартного синтаксиса градиента в период между 2012-м и 2013-м годами, и в этом плане к ним претензий уже достаточно давно нет. Однако, если требуется поддержка устаревших браузеров, можно этого добиться, применяя собственный синтаксис градиента, присущий каждому браузеру с *префиксом поставщика* (см. врезку «*Префиксы поставщиков*»). Для браузера Internet Explorer 9 и более ранних версий можно применять собственную функцию `filter`. Или идите по пути постепенного улучшения и применяйте в качестве запасного варианта сплошной цвет.

ПРИМЕНЕНИЕ ГРАДИЕНТОВ И ПРОИЗВОДИТЕЛЬНОСТЬ

Применение градиентов связано как с рядом преимуществ, так и с наличием недостатков, если рассматривать производительность устройств. С одной стороны, они не нуждаются в дополнительных обращениях к серверу и занимают, по сравнению с изображениями, меньше байтов в процессе загрузки. С другой стороны, отображение «на лету» требует времени и вычислительной мощности, что снижает производительность. Радиальные градиенты в этом смысле наихудшие. Особенно проблематично применять их на мобильных устройствах, где вычислительная мощность ограничена. Подумайте об использовании для мобильных устройств отдельной таблицы стилей без градиентов.

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Стандартный синтаксис градиента поддерживается браузерами Internet Explorer 10+, Edge, Firefox 16+, Chrome 26+, Safari 6.1+, iOS 7.1+ и Android 4.4+.

ПРЕФИКСЫ ПОСТАВЩИКОВ

Производители браузеров, как правило, начинают работать с решениями, касающимися самых современных веб-технологий, прежде чем их спецификации окажутся полностью определены. В течение многих лет они экспериментировали с ними еще до их конечной реализации, добавляя *префикс поставщика* (или *префикс браузера*) к имени свойства или функции. Префикс указывает, что реализация является проприетарной (патентованной) и все еще находится в стадии разработки. Например, когда в Safari были реализованы формы для переноса текста, использовалась ее собственная версия стандартного свойства `shape-outside` с префиксом `-webkit-`:

```
-webkit-shape-outside: url(cube.png);
```

В табл. 13.1 приведен список префиксов, применяемых основными браузерами.

Таблица 13.1. Префиксы поставщика браузера

Префикс	Организация	Наиболее популярные браузеры
<code>-ms-</code>	Microsoft	Internet Explorer
<code>-moz-</code>	Mozilla Foundation	Firefox, Camino, SeaMonkey
<code>-o-</code>	Opera Software	Opera, Opera Mini, Opera Mobile
<code>-webkit-</code>	Изначально Apple, сейчас с открытым исходным кодом	Safari, Chrome, Android, Silk, BlackBerry, WebOS, и многие другие

Префиксы поставщиков позволили разработчикам начать использование новых продвинутых функций CSS в поддерживающих их браузерах, что способствовало дальнейшему развитию веб-дизайна и спецификации. С другой стороны, вся система оказалась сложной и часто неправильно применялась. В конце концов, создатели браузеров согласились оставить префиксную систему и не пытаться больше воспроизводить проприетарные свойства.

В наши дни браузеры скрывают экспериментальные функции за «флагами» (параметрами, которые можно включать или отключать) или в отдельных предварительных выпусках технологий, к которым разработчики могут получить доступ только для тестирования. Если функция окажется стабильной, она публикуется в официальной версии браузера. Методы тестирования отдельных функций CSS будут рассмотрены в главе 19.

Следует также учесть, что имеется несколько свойств и функций CSS, прочно вошедших в моду в эпоху префиксов и которым по-прежнему требуются префиксы для работы в устаревших браузерах. Примером такой функции может служить синтаксис градиента.

Инструменты формирования префикса

Создание всех избыточных префиксных свойств не столь уж и просто, но, к счастью, имеются инструменты, генерирующие их автоматически.

Если применяется один из синтаксисов препроцессора CSS (например, Sass, LESS или Stylus), можно пользоваться их префиксом «mixins». Более подробно о препроцессорах мы поговорим в главе 19.

Если же вы создаете свою таблицу стилей CSS, используя стандартный синтаксис, после завершения работы можете пропустить эту таблицу через постпроцессор типа



- ▶ AutoPrefixer. Этот постпроцессор анализирует стили, а затем автоматически добавляет префиксы только для свойств и обозначений, которые в них нуждаются. При использовании такого инструмента сборки, как Grunt префикс становится частью «шага сборки». Хороший обзор этих инструментов приводится в работе «Autoprefixer: A Postprocessor Dealing with Vendor Prefixes in the Best Possible Way», доступной на сайте в CSS-Tricks (css-tricks.com/autoprefixer/). Более подробно подобные инструменты будут рассмотрены в главе 20.

Градиент для всех браузеров

В следующем примере приведены записи линейного градиента от желтого к зелено-му для ряда браузеров — как устаревших, так и современных, эквивалентные свойству `filter` Internet Explorer, добавленному для корректного сравнения.

Обратите внимание на различия в синтаксисе. Там, где спецификация CSS3 предусматривает использование ключевого слова `to bottom`, большинство браузеров применяют `top`. В очень старой версии, используемой в браузерах WebKit, для линейных и радиальных градиентов применялась инструкция `-webkit-gradient`, но ее быстро заменили отдельными функциями. Другое отличие, неочевидное в приведенном примере, заключается в том, что в устаревшем синтаксисе параметр `0deg` указывал на правый, а не на верхний край, как стало стандартизировано в CSS3, и углы увеличивались в направлении против часовой стрелки.

Здесь для одного градиента представлены уж слишком большие фрагменты кода, но, к счастью, теперь уже скоро ничего такого больше не будет:

```
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 Standard */  
filter: progid:DXImageTransform.Microsoft.gradient(  
startColorstr='#ffff00', endColorstr='#00ff00',GradientType=0 );  
/* IE6-9 */
```

В следующих главах, если какое-либо свойство нуждается в наличии префиксов поставщиков, это будет обязательно отмечено. В противном случае можно предполагать, что стандартная таблица CSS включает все, что вам нужно.

Создание градиентов

Как видите, приведенные только что примеры кода весьма сложны. И, даже если вынести за скобки префиксы поставщиков, задача описания градиентов все равно может оказаться весьма сложной. Вручную создавать такой код хлопотно, поэтому я рекомендую вам использовать онлайн-инструмент для разработки градиентов. Одним из приемлемых вариантов сможет служить Ultimate CSS Gradient Generator от Colorzilla (www.colorzilla.com/gradient-editor/), показанный на рис. ЦВ-13.36. Просто введите столько цветов, сколько пожелаете, и перемещайте ползунки, пока не получите желаемый вид градиента, а затем скопируйте код. Именно это я и проделала для получения только что рассмотренного примера кода. А вот еще один прекрасный вариант, который также включает поддержку повторяющихся градиентов, — инструмент CSS Gradient Generator от VirtuosoSoft (www.virtuosoft.eu/tools/css-gradient-generator/).

Рекомендую вам также полюбоваться на фоновые узоры (рис. ЦВ-13.37), сформированные с помощью градиентов и собранные Леа Веру (Lea Verou) в ее галерее шаблонов CSS3 (lea.verou.me/css3patterns). Все они очень необычны, и вы можете там же взглянуть на код, используемый для создания этих градиентов.

И, В ЗАВЕРШЕНИЕ, ВНЕШНИЕ ТАБЛИЦЫ СТИЛЕЙ

В главе 11 мы говорили о том, что имеются три способа подключения таблиц стилей к HTML-документу: встраивание с помощью атрибута `style`, встраивание с помощью элемента `style` и встраивание в виде внешнего документа `*.css`, связанного с документом или импортированного в него. В этом разделе мы, наконец-то, дойдем до третьего варианта.

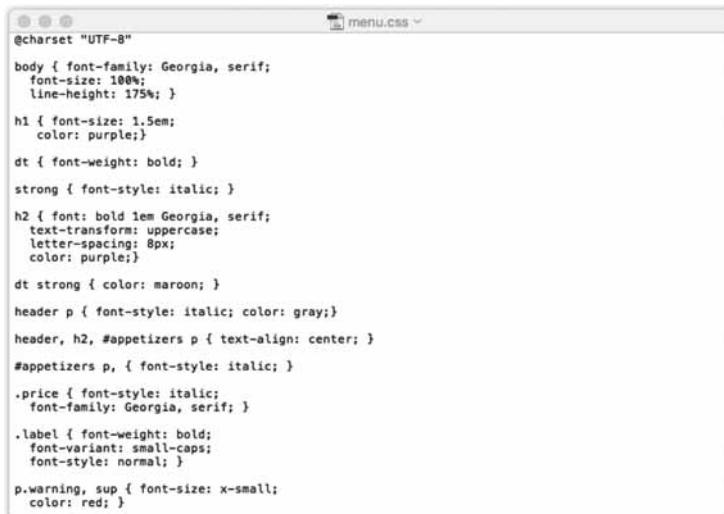
Внешние таблицы стилей — безусловно, наиболее универсальный способ применения CSS, поскольку он дает возможность вносить изменения в стиль всего сайта, изменяя один лишь документ таблицы стилей. Преимущество этого способа заключается том, что вся информация о стилях находится в одном месте и не объединяется с исходным кодом документа.

Кроме того, поскольку браузером для всего сайта загружается и кэшируется единственный документ, определяющий все его стили, то в целом для сайта загружается меньше кода, что приводит к росту его производительности.

И для начала несколько слов собственно о документе с внешней таблицей стилей. Внешняя таблица стилей — это простой текстовый документ, который включает хотя бы одно правило таблицы стилей. Он вовсе может не включать какие-либо HTML-теги (да и в любом случае нет причин их включать). Он может также содержать комментарии, но они должны использовать известный нам синтаксис CSS-комментариев:

```
/* This is the end of the section */
```

Таблица стилей должна иметь расширение `css` (существуют некоторые исключения из этого правила, но вы, как начинающий, вряд ли столкнетесь с ними сейчас). Она может начинаться с объявления кодировки символов с помощью at-правила `@charset`, хотя в реальности это надо делать только при использовании отличной от UTF-8 кодировки. Если применяется at-правило `@charset`, оно должно быть первым элементом таблицы стилей, без каких-либо символов, правил стиля или комментариев, предшествующих ему. На рис. 13.38 показано, как выглядит короткая таблица стилей в окне моего текстового редактора.



```
@charset "UTF-8"
body { font-family: Georgia, serif;
font-size: 100%;
line-height: 175%; }

h1 { font-size: 1.5em;
color: purple; }

dt { font-weight: bold; }

strong { font-style: italic; }

h2 { font: bold 1em Georgia, serif;
text-transform: uppercase;
letter-spacing: 8px;
color: purple; }

dt strong { color: maroon; }

header p { font-style: italic; color: gray; }

header, h2, #appetizers p { text-align: center; }

#appetizers p, .price { font-style: italic; }

.price { font-style: italic;
font-family: Georgia, serif; }

.label { font-weight: bold;
font-variant: small-caps;
font-style: normal; }

p.warning, sup { font-size: x-small;
color: red; }
```

Рис. 13.38. Внешняя таблица стилей представляет собой сугубо текстовый документ, включающий только CSS-правила и комментарии

Имеются два способа подключения внешней таблицы стилей: с помощью элемента `link` или правила `@import`. Рассмотрим оба эти способа.

Применение элемента `link`

Элемент `link` определяет связь между текущим документом и внешним ресурсом. Безусловно, самое популярное его применение — ссылка на таблицы стилей. Элемент `link` включается в заголовок текущего документа:

```
<head>
    <title>Titles are required.</title>
    <link rel="stylesheet" href="/path/stylesheet.css">
</head>
```

В элемент `link` следует включить два атрибута:

- `rel="stylesheet"` — определяет отношение связанного документа к текущему документу. Значением атрибута `rel` при ссылке на таблицу стилей всегда является `stylesheet`;
- `href="url-ссылка"` — указывает местоположение файла таблицы стилей (`*.css`).

Можно включить в разные таблицы стилей несколько элементов `link`, и все они будут применены к документам, к которым они привязаны. Если появятся конфликты, то с учетом порядка применения правил и каскадирования ссылка, указанная последней, переопределит предыдущие настройки.

Импорт с помощью правила `@import`

Еще один способ присоединения внешней таблицы стилей к документу — ее импорт с помощью ат-правила `@import`. Правило `@Import` — это такой тип правила, которым можно добавить таблицу стилей либо во внешнем документе таблицы стилей `*.css`, либо прямо в элементе `style`, как показано в следующем примере:

```
<head>
  <style>
    @import url("/path/stylesheet.css");
    p { font-face: Verdana; }
  </style>
  <title>Titles are required.</title>
</head>
```

В этом примере показана относительная URL-ссылка, но она также может быть и абсолютной (начинаться с `http://`). Правило `@import` приводится в начале таблицы стилей *перед любыми селекторами*. Можно импортировать более одной таблицы стилей, и все они будут применяться, но правила последней из указанных таблиц стилей имеют приоритет над предыдущими.

С помощью медиазапросов можно ограничить импорт таблицы стилей определенными типами мультимедиа (например, экран, печать или проекция) или средами просмотра (ориентация, размер экрана и т. д.). *Медиазапросы* — это метод применения стилей на основе медиа, используемого для отображения документа. Они записываются после правила `@import` через запятую. Например, если сформирована таблица стилей, которую следует импортировать и применять только при печати документа, воспользуйтесь таким правилом:

```
@import url(print_styles.css) print;
```

А для применения специальной таблицы стилей только для небольших устройств можно запросить область просмотра:

```
@import url(small_device.css) screen and (max-width: 320px);
```

Медиазапросы будут подробно рассмотрены в главе 17, но здесь они упоминаются, поскольку имеют отношение к импорту таблиц стилей.

А пока вы можете опробовать методы `link` и `@import`, выполнив *упражнение 13.8*.

УПРАЖНЕНИЕ 13.8. СОЗДАНИЕ ВНЕШНЕЙ ТАБЛИЦЫ СТИЛЕЙ

В процессе разработки веб-страницы можно, конечно, задействовать и встроенную таблицу стилей, но после завершения разработки лучше всего перейти на использование внешней таблицы стилей, которую можно применить для

форматирования сразу нескольких документов сайта. Выполним следующие действия для таблицы стилей нашего «летнего меню».

1. Откройте последнюю версию файла `summer-menu.html`. Выделите и вырежьте все правила из элемента `style`, но оставьте теги `<style>...</style>` — они нам еще понадобятся.
2. Сформируйте новый простой текстовый документ ASCII и вставьте туда все вырезанные из правила элемента `style` стили. Убедитесь, что туда случайно не попала разметка.
3. Сохраните этот документ под именем `menustyles.css` в том же каталоге, что и документ `summer-menu.html`.
4. Теперь вернитесь назад к документу `summer-menu.html` и добавьте в него правило `@import` для присоединения внешней таблицы стилей:

```
<style>
  @import url(menustyles.css);
</style>
```

Сохраните файл и перезагрузите его в браузере. Он должен выглядеть точно так же, как и со встроенной таблицей стилей. Если это не так, вернитесь в документ и убедитесь, что все соответствует примерам.

5. Удалите полностью элемент `style` и добавьте в заголовок документа таблицу стилей с помощью элемента `link`:

```
<link rel="stylesheet"
  href="menustyles.css">
```

Опять же, проверьте свою работу, сохранив документ и просмотрев его в браузере.

Использование модульных таблиц стилей

Поскольку можно компилировать информацию из нескольких внешних таблиц стилей, популярным методом управления стилями стали так называемые *модульные таблицы стилей*. Многие разработчики хранят стили, которые они часто используют, — например, типографские стили, правила макета или связанные с формой стили — в отдельных таблицах стилей, а затем комбинируют их в режиме смешивания и сопоставления с использованием правил `@import`. Напомню, что правила `@import` должны находиться перед правилами, где применяются селекторы.

Рассмотрим пример таблицы стилей, которая импортирует несколько внешних таблиц стилей:

```
/* Базовые типографские стили */
@import url("type.css");

/* Поля ввода формы */
@import url("forms.css");

/* Навигация */
@import url("list-nav.css");
```

URL БЕЗ ЗАПИСИ `URL()`

Интернет-адреса (URL) можно также включать в правила без записи `url()`:

```
@import "/path/style.css";
```

Кроме того, использование абсолютных путей, начинающихся с корня, гарантирует, что документ `*.css` будет обнаружен всегда.

```
/* Стили site-specific */
body { background: orange; }

/* Еще какие-либо правила стилей */
```

Подобную весьма удобную методику я рекомендую взять на вооружение при формировании опыта создания сайтов. Так что если у вас уже есть хорошо работающие решения, вряд ли имеет смысл заново изобретать колесо для каждого нового сайта. Впрочем, несмотря на то что модульные таблицы стилей являются неплохим средством, позволяющим экономить время и улучшать организацию труда, они могут уменьшать производительность и ухудшать кэширование.

Поэтому, если вы решите прибегнуть к использованию модульных таблиц стилей, рекомендуется скомпилировать все стили в один документ и уже потом передавать его в браузер. Но не озабочивайтесь — вам не придется делать это вручную. Существуют инструменты, которые сделают это за вас. CSS-препроцессоры LESS и Sass (которые будут официально представлены в главе 20) — это всего лишь два инструмента из многих, способных выполнять компиляцию.

ПОДВОДИМ ИТОГИ

В этой главе мы рассмотрели много базовых тем. Это и способы установки для элемента цветов переднего плана и фона при помощи различных числовых систем и названий цветов. Это и варианты настройки уровня прозрачности с помощью свойства `opacity`, а также цветовых пространств RGBa и HSLa. Мы потратили много времени на изучение различных способов добавления фоновых изображений, настройки создания мозаичных фонов, выбора места расположения исходных фоновых изображений и их размеров. Мы также увидели, каким образом можно применять в качестве фоновых изображений линейные и радиальные градиенты. Попутно мы разобрались с селекторами псевдоклассов, псевдоэлементов и атрибутов и изучили способы присоединения внешних таблиц стилей. Я думаю, что этого для одной главы достаточно! А теперь посмотрите, что вы запомнили, обратившись к следующей маленькой викторине.

КОНТРОЛЬНЫЕ ВОПРОСЫ

На этот раз я проверю ваше мастерство при выборе ответов на вопросы с несколькими вариантами ответов. Ответы на эти вопросы приведены в *приложении 1*.

1. Какие из этих заданных по умолчанию областей заполнены цветом фона?
 - a. Область позади контента
 - b. Области отступа вокруг контента
 - c. Область под границей
 - d. Пробел поля вокруг документа

- e. Все, что находится над ним
- f. a и b
- g. a, b и c
2. Какой из этих способов не позволяет указать белый цвет в CSS?
- #FFFFFF
 - #FFF
 - rgb(255, 255, 255)
 - rgb(FF, FF, FF)
 - white
 - rgb(100%, 100%, 100%)
3. Установите соответствие между псевдоклассом и элементами, которые его указывают.
- | | |
|-----------------|---|
| a. a:link | 1. Ссылки, на которые уже щелкнули. |
| b. a:visited | 2. Элемент, который выделен и подготовлен для ввода. |
| c. a:hover | 3. Элемент, который является первым дочерним элементом своего родителя. |
| d. a:active | 4. Ссылка с указателем мыши над ним. |
| e. :focus | 5. Ссылки, которые еще не посещались. |
| f. :first-child | 6. Ссылка, по которой щелкают. |
4. Сопоставьте следующие правила с их соответствующими образцами, как показано на рис. 13.39. Все образцы на рисунке применяют один исходный документ, состоящий из одного элемента абзаца, к которому применены некоторые отступы и граница.

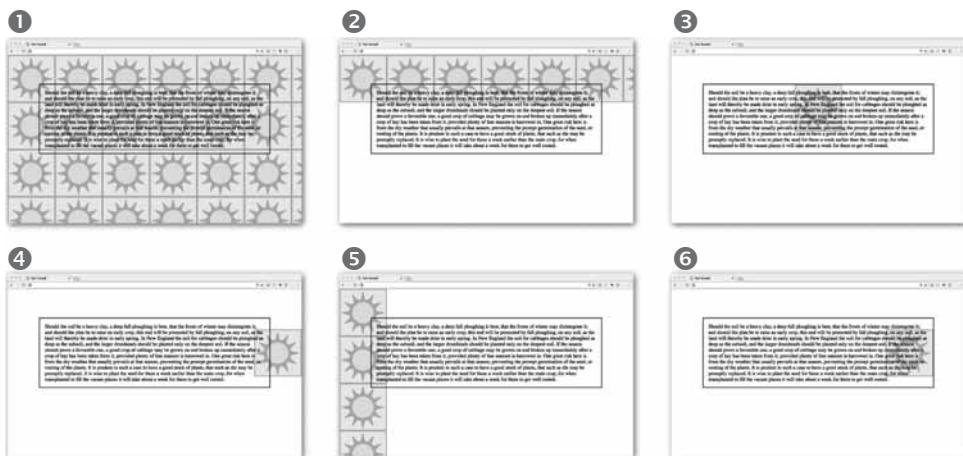


Рис. 13.39. Образцы для вопроса 4

```
a. body {
    background-image: url(graphic.gif);
}
```

```

b. p {
    background-image: url(graphic.gif);
    background-repeat: no-repeat;
    background-position: 50% 0%;
}

c. body {
    background-image: url(graphic.gif);
    background-repeat: repeat-x;
}

d. p {
    background: url(graphic.gif) no-repeat right center;
}

e. body {
    background-image: url(graphic.gif);
    background-repeat: repeat-y;
}

f. body {
    background: url(graphic.gif) no-repeat right center;
}

```

ОБЗОР CSS: СВОЙСТВА ЦВЕТА И ФОНА

В табл. 13.2 в алфавитном порядке приводятся краткие описания свойств цвета и фона.

Таблица 13.2. Свойства цвета и фона

Свойство	Описание
Background	Сокращенное свойство, объединяющее свойства фона
background-attachment	Определяет, прокручивается ли фоновое изображение или является фиксированным
background-clip	Определяет, как далеко должно распространяться фоновое изображение
background-color	Определяет цвет фона для элемента
background-image	Предоставляет расположение изображения для использования в качестве фона
background-origin	Определяет, как вычисляется значение свойства <i>background-position</i> (от края рамки, отступа или поля содержимого)
background-position	Определяет местоположение исходного фонового изображения
background-repeat	Определяет, повторяется ли и как фоновое изображение (мозаика)
background-size	Определяет размер фонового изображения
color	Определяет цвет переднего плана (текст и граница)
opacity	Определяет уровень прозрачности переднего плана и фона

ГЛАВА 14

БЛОЧНАЯ МОДЕЛЬ

В этой главе...

- ▶ *Части блока элемента*
- ▶ *Установка размеров блока*
- ▶ *Отступы*
- ▶ *Границы*
- ▶ *Контуры*
- ▶ *Поля*
- ▶ *Присваивание типов отображения*
- ▶ *Добавление теней*

В главе 11 я представила вам *блочную модель* как одну из фундаментальных концепций CSS. В соответствии с *блочной моделью* каждый элемент в документе генерирует блок, и к этому блоку применимы такие свойства, как ширина, высота, отступы, границы и поля. Как работают блоки элементов, вы, вероятно, почувствовали, когда мы добавляли к элементам фон. В этой главе рассматриваются все свойства, связанные с блоком, — сначала приводится обзор компонентов блока элементов, а затем мы подробно рассмотрим свойства собственно блока: размеры его содержимого, отступы, границы и поля.

БЛОК ЭЛЕМЕНТА

Как уже отмечалось ранее, каждый элемент в документе — будь то блочный элемент или строка — генерирует прямоугольный блок элемента (рис. ЦВ-14.1). Обратите внимание на новую терминологию — в дальнейшем она поможет вам правильно ориентироваться в материале этой главы.

Область контента — ядро блока элемента составляет собственно его контент (содержимое). На рис. ЦВ-14.1 область контента выделена белым прямоугольником.

Внутренние края — так называются края области контента. Хотя внутренние края выделены на рис. ЦВ-14.1 цветом, отличающимся от цвета собственно контента, на реальных веб-страницах край области контента невидим.

Отступы — это область между областью контента и необязательно присутствующей границей. На рис. ЦВ-14.1 область отступов обозначена желто-оранжевым цветом. Отступы также не являются обязательными.

Граница — это простая линия (или линия, помеченная стилем), окружающая элемент и отступы. Границы, как уже отмечалось, также необязательны.

Поле — необязательная область, присутствующая за *пределами* границы. На рис. ЦВ-14.1 поле обозначено светло-голубым цветом, однако на самом деле поля прозрачны, вследствие чего через них просвечивается фон родительского элемента.

ПРОСТРАНСТВО, ЗАНИМАЕМОЕ ЭЛЕМЕНТОМ НА СТРАНИЦЕ

Объем пространства, занимаемого элементом на странице, включает контент плюс отступы, границы и поля, примененные к элементу.

примененные к элементу. Внешние края на рис. ЦВ-14.1 обозначены пунктирной линией, но на веб-страницах они невидимы.

Приведенные здесь блочные компоненты содержат все элементы, однако, как будет показано далее, некоторые их свойства зависят от того, является ли элемент блоком или текстовой строкой. Некоторые из этих различий станут заметны сразу же, как только мы приступим к рассмотрению размеров блока.

Внешние края — как следует из их названия, являются внешними краями блока элементов. Они ограничивают общую площадь, которую элемент занимает на странице, включая всю область контента плюс все отступы, границы и поля,

СВОЙСТВА **WIDTH, HEIGHT И BOX-SIZING** (УКАЗАНИЕ РАЗМЕРОВ БЛОКА)

width

- **Значения:** длина | процентное соотношение | auto.
- **По умолчанию** — auto.
- **Применение** — к элементам уровня блока и заменяемым строчным элементам (таким, как изображения).
- **Наследование** — нет.

height

- **Значения:** длина | процентное соотношение | auto.
- **По умолчанию** — auto.
- **Применение** — к элементам уровня блока и заменяемым строчным элементам (таким, как изображения).
- **Наследование** — нет.

box-sizing

- **Значения:** content-box | border-box.
- **По умолчанию** — content-box.

- **Применение** — ко всем элементам.
- **Наследование** — нет.

По умолчанию значения ширины и высоты элемента блока вычисляются браузером автоматически (то есть ему присваивается заданное по умолчанию значение `auto`). Блок при этом имеет ширину, равную окну браузера или другому содержащему этот блок элементу, и высоту, необходимую для размещения контента. Однако, применив свойства `width` и `height`, можно привести область контента элемента к заданным значениям по ширине или высоте.

К сожалению, установить размеры блока значительно сложнее, чем просто внести значения ширины и высоты в таблицу стилей. Вы должны точно знать, какой части блока элементов вы эти размеры присваиваете.

Указать размеры элемента можно двумя способами. Способ по умолчанию, введенный в CSS1, применяет значения `width` и `height` к блоку контента. Тогда итоговый размер элемента складывается из указанного вами размера плюс области отступов и границ, присущие элементу. Другой метод, представленный в CSS3 как часть свойства `box-sizing`, применяет значения ширины и высоты к границе блока, которая включает контент, отступы и собственно границу. В результате применения этого способа *видимое поле элемента*, включающее отступы и границы, в точности соответствует заданным размерам. Далее мы познакомимся с обоими способами.

Независимо от выбранного способа значения ширины и высоты можно задавать только для элементов уровня блока и нетекстовых строчных элементов, таких, как изображения. Свойства `width` и `height` не применяются к строкам текста (`non-replaced`, *незаменяемым*) и игнорируются браузером. Другими словами, нельзя указать ширину и высоту якоря (`a`) или элемента `strong`.

Задание размеров блока контента (модель `content-box`)

По умолчанию (то есть если не включать в стили правило `box-sizing`) свойства `width` и `height` (ширина и высота) применяются как раз к блоку контента. Именно таким образом все современные браузеры интерпретируют значения ширины и высоты, но вы можете явно определить такое их поведение, присвоив свойству `box-sizing` значение `content-box` (`box-sizing: content-box`).

В следующем примере (рис. 14.2) простому блоку присваивается ширина 500 пикселов, высота 150 пикселов, с отступом 20 пикселов, границей 5 пикселов и полем

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Основные браузеры начали поддерживать свойство `box-sizing`, начиная с 2011–2012 годов. Для выпущенных ранее браузеров (`Chrome <10`, `Safari <5.1`, `Safari iOS <5.1` или `Android <4.3`) существует префиксная версия: `-webkit-box-size`, но в настоящее время она не считается актуальной. `Internet Explorer 6` и `7` вообще не поддерживают `box-sizing`, но эти версии браузеров сейчас почти не применяются.

СВОЙСТВО `DISPLAY`

Тем не менее существует способ применить свойства `width` и `height` к строчным элементам, таким, как якоря (`a`) — с помощью свойства `display` можно принудить их весит себя как блочные элементы (это свойство рассматривается в конце главы).

вокруг в 20 пикселов (напомню, что в заданной по умолчанию модели блока контента значения `width` и `height` применяются только к *области контента*):

```
p {
    background: #f2f5d5;
    width: 500px;
    height: 150px;
    padding: 20px;
    border: 5px solid gray;
    margin: 20px;
}
```

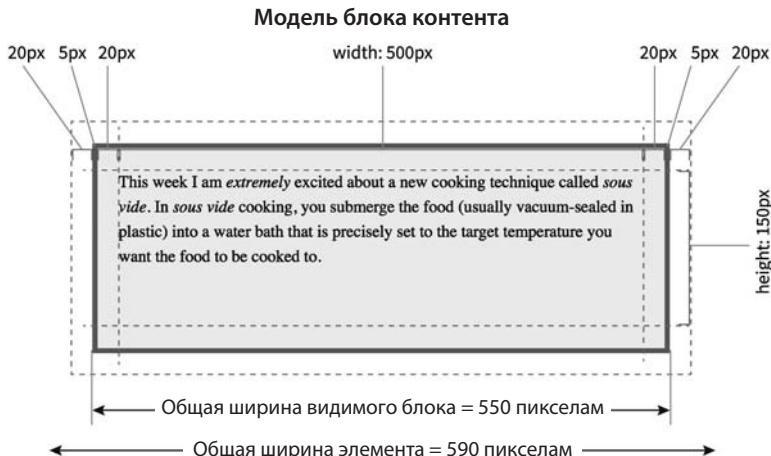


Рис. 14.2. Указание ширины и высоты с помощью модели `content-box`

Результирующая ширина *видимого блока* элементов в итоге составляет 550 пикселов: контент плюс отступы — 40 пикселов (по 20 пикселов слева и справа) и 10 пикселов границы (по 5 пикселов слева и справа):

$$\begin{aligned} \text{Видимый блок элемента} &= \\ &= 5 \text{ пикселов} + 20 \text{ пикселов} + 500 \text{ пикселов ширина} + \\ &+ 20 \text{ пикселов} + 5 \text{ пикселов} = 550 \text{ пикселям} \end{aligned}$$

После добавлении поля, равного 40 пикселов, ширина *всего* поля блока элемента составит 590 пикселов. Знание итогового размера ваших элементов имеет решающее значение — только тогда вы будете уверены, что макет поведет себя предсказуемо:

$$\begin{aligned} \text{Полная ширина блока элемента} &= \\ &= 20 \text{ пикселов} + 5 \text{ пикселов} + 20 \text{ пикселов} + 500 \text{ пикселов ширина} + \\ &+ 20 \text{ пикселов} + 5 \text{ пикселов} + 20 \text{ пикселов} = 590 \text{ пикселям} \end{aligned}$$

Задание размеров блока элемента (модель `border-box`)

Другой способ указать размер элемента — задать размеры ширины и высоты *всего* *видимого блока*, включая отступы и границу. Поскольку эти установки не входят

в заданное по умолчанию поведение браузера, в таблице стилей необходимо установить явно значение: `box-sizing: border-box`.

Вернемся к примеру из предыдущего раздела и разберемся, что произойдет, если с помощью метода `border-box` задать ширину блока элемента равной 500 пикселов (рис. 14.3). Все остальные объявления стиля для блока остаются прежними:

```
p {
    ...
    box-sizing: border-box;
    width: 500px;
    height: 150px;
}
```

Модель border-box

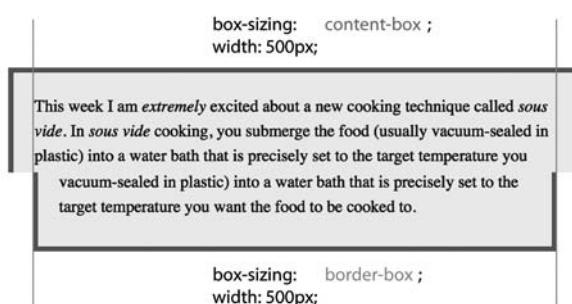


Рис. 14.3. Определение размера элемента с помощью метода `border-box` (вверху), сравнение полученных полей для каждого метода задания размеров (внизу)

Ширина видимого блока в этом случае составит 500 пикселов (сравните с 550 пикселами в модели `content-box`), а общая ширина блока элемента — 540 пикселов. Многие разработчики находят, что модель `border-box` представляет собой более интуитивно понятный способ задания размеров элементов. Особенно удобно этот метод применять для указания ширины в процентах, что является одним из краеугольных камней адаптивного дизайна. Например, можно задать двум стоящим

рядом стобцам ширину 50% и не озабочиваться более вычислением значений для отступов и ширины границ (тем не менее при этом необходимо учитывать поля).

Сейчас многие разработчики используют модель `border-box` для определения *всех* значений в документе, включая ее в корневой элемент документа (`html`), после чего все остальные элементы сами определяются с помощью наследования, например, так:

```
html {box-sizing: border-box;}  
*, *:before, *:after {box-sizing: inherit;}
```

Дополнительная информация по этой методике содержится в работе Криса Койера (Chris Coyier) «Inheriting box-sizing Probably Slightly Better Best-Practice», которая доступна на сайте: css-tricks.com/inheritingbox-sizing-probably-slightly-better-best-practice.

Свойство `height` (указание высоты блока)

МИНИМАЛЬНОЕ И МАКСИМАЛЬНОЕ ЗНАЧЕНИЯ РАЗМЕРОВ

Если вы хотите установить ограничение на размер блока элемента, примените для свойств `width` и `height` свойства `max-` и `min-`:

```
max-height, max-width,  
min-height, min-width
```

Значения: `длина | процентное соотношение | none`

Эти свойства могут применяться только для блочных элементов и заменяемых (например, изображений). Если используется модель `content-box`, значения распространяются только на область контента, и, в случае если применяются отступы, границы или поля, общий размер элемента увеличится, даже если указано свойство `max-width` или `max-height`. Обратите внимание, что браузер IE8 не поддерживает свойство `box-sizing` совместно со свойствами `max-/min-`.

ЕЩЕ О МОДЕЛИ `BORDER-BOX`

Избегайте применять свойства `max-` и `min-` вместе со свойствами `width` и `height` в модели `border-box`. Известно, что этот подход приводит к проблемам с браузером.

Свойство `height` функционирует так же, как и `width`. Обычно высоту элементов указывают редко. В зависимости от природы среды высота вычисляется автоматически, что позволяет блоку элемента изменяться в зависимости от размеров шрифта, пользовательских настроек или иных факторов. Если вы решите указать высоту для содержащего текст элемента, не забудьте рассмотреть случай, когда в этом блоке может не поместиться контент. К счастью, CSS предоставляет варианты, и я расскажу вам о них в следующем разделе.

Свойство `overflow` (обработка переполнения)

Если размеры элемента слишком малы для размещения его содержимого, с помощью свойства `overflow` можно указать, что делать с таким содержимым:

`overflow`

- **Значения:** `visible | hidden | scroll | auto.`
- **По умолчанию** — `visible`.
- **Применение** — к элементам блочного уровня и заменяемым строчным элементам (таким, как изображения).
- **Наследование** — нет.

Рассмотрим варианты значений подробнее:

- `visible` — заданное по умолчанию значение `visible` позволяет содержимому выходить за поле элемента, в результате чего оно становится видимым;
- `hidden` — если для свойства `overflow` задано значение `hidden`, содержимое, которое не помещается в блок, отрезается и не отображается за пределами области контента элемента;
- `scroll` — при указании значения `scroll` в блок элемента добавляются полосы прокрутки, разрешающие пользователю прокручивать его содержимое. Имейте в виду, что они видимы только при щелчке на элементе для прокрутки. В устаревших iOS (<4), Android (<2.3) и некоторых других устаревших мобильных браузерах возникает проблема с этим значением, поэтому желательно для мобильных устройств применять более простую альтернативу `overflow: scroll`;
- `auto` — значение `auto` позволяет браузеру решать, каким образом справиться с переполнением. В большинстве случаев полосы прокрутки добавляются, лишь если контент не помещается и они необходимы.

На рис. 14.4 показаны предопределенные значения для свойства `overflow`, примененные к элементу площадью 150 пикселов. Подсветка фона делает края области контента видимыми.

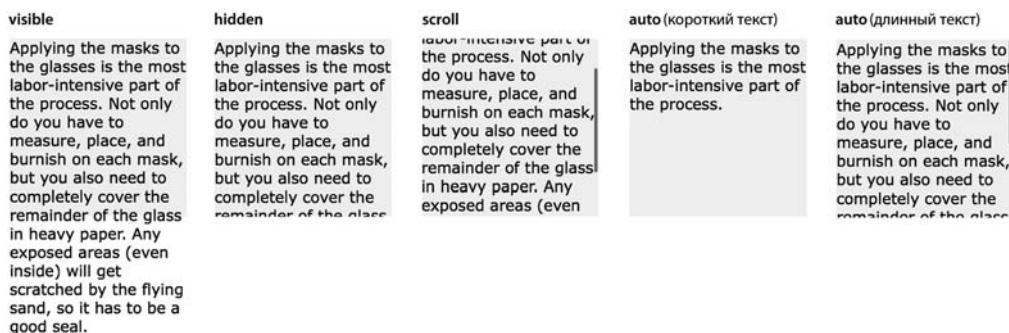


Рис. 14.4. Варианты обработки переполнения контента. Параметры `scroll` и `auto` (короткий текст) определяют узкие серые полосы прокрутки справа от текста (как это отображено в macOS)

ОТСТУПЫ

Отступ представляет собой пространство между областью содержимого (контента) и границей (или местом, где она может находиться, если не указана). Я нахожу полезным добавлять отступы к элементам, когда использую фоновый цвет или границы. Это дает содержимому возможность расположиться посвободнее, а границам или краям фона — не прижиматься вплотную к тексту.

ОТСТУПЫ

Отступы — это пространство между содержимым и границей.

Отступы можно добавлять к отдельным сторонам любого элемента (уровня блока или строчного):

padding-top, padding-right, padding-bottom, padding-left

- **Значения:** длина | процентное соотношение.
- **По умолчанию** — 0.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Существует также сокращенное свойство **padding**, позволяющее добавлять отступы одновременно со всех сторон:

padding

- **Значения:** длина | процентное соотношение.
- **По умолчанию** — 0.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Свойства **padding-top**, **padding-right**, **padding-bottom** и **padding-left** задают величину отступа для каждой стороны элемента, как показано в следующем примере (рис. 14.5). Обратите внимание, что здесь также добавлен цвет фона, что позволяет сделать видимыми внешние края области заполнения:

```
blockquote {
    padding-top: 2em;
    padding-right: 4em;
    padding-bottom: 2em;
    padding-left: 4em;
    background-color: #D098D4; /* light green */
}
```

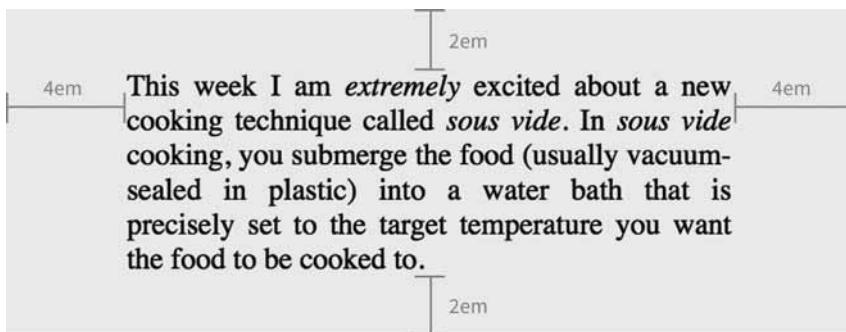


Рис. 14.5. Добавление отступов вокруг содержимого элемента

в спецификациях CSS предусмотрено указание отступа в любых единицах длины (наиболее распространены **em** и **px**) или в процентах от **ширины** родительского

элемента. Кстати, значение ширины родительского элемента используется в качестве основы и для верхнего и нижнего отступов. Если ширина родительского элемента изменяется, то изменяется значение отступа со всех сторон дочернего элемента, что усложняет отслеживание изменений процентных значений.

Сокращенное свойство *padding*

В качестве альтернативы последовательному заданию отступа для каждой из сторон контента можно воспользоваться сокращенным свойством *padding*, добавляющим отступы ко всему элементу сразу. Синтаксис этого свойства достаточно интересен: для одного свойства *padding* можно указывать четыре, три, два или одно значение. Рассмотрим, как это работает, начиная с четырех значений.

Если ввести для свойства *padding* четыре значения отступов, они применяются к каждой из сторон по часовой стрелке, начиная с верхней. Некоторые пользователи применяют мнемоническое правило «TRouBLe» (Top Right Bottom Left) для оформления контента отступами сверху, справа, снизу и слева. Такой подход применим в целом к синтаксису сокращенных значений в CSS, поэтому обратите внимание на следующую запись:

```
padding: top right bottom left;
```

Применяя свойство *padding*, можно воспроизвести отступы, указанные в предыдущем примере с помощью четырех отдельных свойств:

```
blockquote {  
    padding: 2em 4em 2em 4em;  
    background-color: #D098D4;  
}
```

Если левый и правый отступы одинаковы, можно сократить запись, указав только три значения. Значение для «справа» (второе значение в строке) будет отражено и применено и для «слева». Браузер, видя, что значение «слева» отсутствует, просто применит значение «справа» для обеих сторон. Синтаксис для трех значений выглядит так:

```
padding: top right/left bottom;
```

Следующее правило эквивалентно предыдущему примеру, поскольку отступы на левом и правом краях элемента установлены в 4em:

```
blockquote {  
    padding: 2em 4em 2em;  
    background-color: #D098D4;  
}
```

Продолжая работу с этим шаблоном, можно предоставить только два значения: первое используется для верхнего и нижнего краев, а второе — для левого и правого:

```
padding: top/bottom right/left;
```

СОКРАЩЕННЫЕ ЗНАЧЕНИЯ

Одно значение:

```
padding: 10px;
```

Применяется ко всем строкам.

Два значения:

```
padding: 10px 6px;
```

Сперва — сверху и снизу, затем — слева и справа.

Три значения:

```
padding: 10px 6px 4px;
```

Сперва сверху; затем — слева и справа; наконец — внизу.

Четыре значения:

```
padding: 10px 6px 4px 10px;
```

Последовательно применяется по часовой стрелке: сверху, справа, снизу и слева (принцип TRBL).

При этом эффекта, полученного в предыдущих двух примерах, можно достичь с помощью такого правила:

```
blockquote {
    padding: 2em 4em;
    background-color: #D098D4;
}
```

Обратите внимание, что все приведенные примеры выглядят так, как показано на рис. 14.5.

Наконец, если указать только одно значение, оно будет применено ко всем четырем сторонам элемента. Следующее объявление задает 15 пикселов отступа для всех сторон элемента div:

```
div#announcement {
    padding: 15px;
    border: 1px solid;
```

Попрактикуйтесь теперь в добавлении отступов, выполнив *упражнение 14.1*.

УПРАЖНЕНИЕ 14.1. ДОБАВЛЕНИЕ НЕБОЛЬШОГО ОТСТУПА

При выполнении этого упражнения мы начнем добавлять свойства блока для улучшения внешнего вида сайта придуманной мной пекарни «Black Goose Bakery». Я облегчила вам задачу, разметив исходный код веб-страницы в HTML-документе **bakery.html**. В отличие от веб-страниц, создаваемых при выполнении предыдущих упражнений, на веб-странице для пекарни используется внешняя таблица стилей **bakery-styles.css**. Стейлинг этого сайта на протяжении нескольких следующих глав выполняется с помощью CSS-файла, поэтому его HTML-документ вам редактировать не придется, однако для ознакомления с результатами изменений надо будет просматривать файл **bakery.html**, открывая его в браузере. Все необходимые для работы файлы доступны на сайте: learningwebdesign.com/5e/materials.

На рис. 14.6 показан вид страницы сайта до внесения изменений (слева) и после их внесения (справа). Для внесения всех изменений необходимо выполнить упражнения, приведенные в этой и следующих двух главах, и создание отступов — только начало работы. В главе 16 мы превратим малопривлекательный список навигации в симпатичную панель меню навигации (а пока, пожалуйста, не фиксируйтесь на этом) и добавим на страницу двухколоночный макет, пригодный для отображения на больших экранах.

Итак, начните с ознакомления с исходным документом. Откройте файл **bakery.html** в браузере и текстовом редакторе и посмотрите, над чем вам предстоит работать. Таблица стилей добавлена с помощью правила `@import` в элемент `style`.

Документ размечен с помощью заголовка (включая раздел `nav`), а также разделов `main`, `aside` и `footer`.

Просмотрите таблицу стилей `bakery-styles.css` в текстовом редакторе. Для организации связанных с каждым разделом стилей использованы комментарии в таблице стилей (вы даже заслужите бонусные баллы, если по мере продвижения сохраните организацию стилей!). Там содержатся стили, применяемые для форматирования текста, цвета и фона, — это все рассмотренные ранее в книге свойства, поэтому они должны быть вам знакомы. А сейчас мы добавим в файл `bakery-styles.css` несколько правил, которые создадут отступы для нужных элементов.



Рис. 14.6. Сайт «Black Goose Bakery»: до (слева) и после преобразований (справа)

- Сначала установим для свойства модели `box-sizing` значение `border-box`, применяемое для всех элементов в документе. Добавьте эти новые правила к существующему элементу `style`. Это облегчит выполнение измерений в будущем:

```
html {
    box-sizing: border-box;
}
* {
    box-sizing: inherit;
}
```

- Найдите стили для раздела `header` и задайте его высоту. По умолчанию заполняется 100% ширины страницы, поэтому о ширине беспокоиться не нужно. Для высоты выбрано значение, равное `15em`, поскольку это значение кажется достаточным, чтобы вместить контент и показать круассан на выигрышном фоне, но с изображением можно и поэкспериментировать:

```
header {
    ...
    height: 15em;
}
```

3. Раздел `main` нуждается в небольшом отступе, поэтому добавьте `1em` отступа со всех сторон. Соответствующее объявление можно добавить после имеющихся стилей `main`:

```
main {
  ...
  padding: 1em;
}
```

4. Рассмотрим теперь элемент `aside` («Hours»). К нему необходимо добавить дополнительные отступы с левой стороны, чтобы разместить мозаичное фоновое изображение. Используется несколько подходов к применению разных значений отступа для каждой из сторон, но сделать это можно так, чтобы преднамеренно переопределить более ранние объявления.

Воспользуйтесь сокращенным свойством `padding` для добавления значения отступа, равного `1em`, со всех сторон элемента `aside`. Затем запишите второе объявление, добавляющее 45 пикселов отступа только с левой стороны. Поскольку объявление `padding-left` приведено вторым, оно перезапишет левый отступ, равный `1em`, примененный с помощью сокращенного свойства:

```
aside {
  ...
  padding: 1em;
  padding-left: 45px;
}
```

5. Наконец, заметим, что нижний колонтитул смотрится некрасиво. Поэтому добавим отступы, увеличивающие высоту нижнего колонтитула и придающие его контенту некоторую свободу:

```
footer {
  ...
  padding: 1em;
```

6. Сохраните документ `bakery-styles.css` и затем откройте (или перезагрузите) документ `bakery.html` в браузере для просмотра результатов работы (рис. ЦВ-14.7). Что ж, пока улучшения еще малозаметны — показаны только изменения в отступах.

ВЕБ-ШРИФТ GOOGLE STINT

При выполнении веб-дизайна этой страницы применялся веб-шрифт Google под названием *Stint*. Для работы с ним необходимо иметь подключение к Интернету. Если же работать в автономном режиме, вместо него вы увидите *Georgia* или другой шрифт с засечками, что вполне подходит для наших целей, но страница будет иметь вид не такой, как на рисунках.

ГРАНИЦЫ

Граница — это просто линия, нарисованная вокруг области содержимого и ее (необязательного) добавления. Можно выбрать любой из восьми стилей границ, сделать их такой ширины и такого цвета, как вам понравится. Границы могут устанавливаться вокруг всего элемента или только на какой-либо его стороне или сторонах. В CSS3 были добавлены свойства для скругления углов границ или применения к ним изображений. Давайте начнем изучение границ с рассмотрения различных их стилей.

СТИЛИ ГРАНИЦ

Стиль является наиболее важным из свойств границы, поскольку согласно CSS-спецификации, если стиль границы не указан, граница отсутствует (по умолчанию применяется значение `none`). Другими словами, всегда следует объявлять стиль границы, иначе другие свойства границы игнорируются.

Стили границ можно применять к каждой из сторон поочередно:

`border-top-style`, `border-right-style`, `border-bottom-style`, `border-left-style`

- **Значения:** `none` | `solid` | `hidden` | `dotted` | `dashed` | `double` | `groove` | `ridge` | `inset` | `outset`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

или же воспользоваться сокращенным свойством `border-style`:

`border-style`

- **Значения:** `none` | `solid` | `hidden` | `dotted` | `dashed` | `double` | `groove` | `ridge` | `inset` | `outset`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Значением свойства `border-style` служит одно из 10 ключевых слов, доступных для стилей границы (рис. 14.8). Значение `hidden` эквивалентно значению `none`.

Для задания стиля какой-либо из сторон элемента примените специфицированные свойства стиля границы: `border-top-style`, `border-right-style`, `border-bottom-style` и `border-left-style`. Если не указать значение толщины (ширины) границы, по умолчанию устанавливается ее

ПРИМЕНЯЙТЕ НИЖНИЕ ГРАНИЦЫ ВМЕСТО ПОДЧЕРКИВАНИЯ

Отключение подчеркивания ссылок и замена их на пользовательскую нижнюю границу — обычный технический прием при дизайне. Вид ссылок упрощается, но при этом они выделяются из обычного текста.

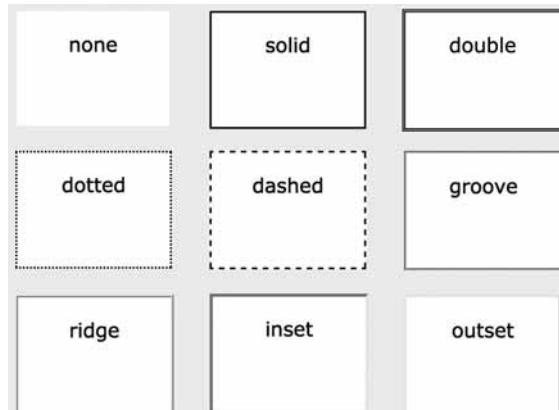


Рис. 14.8. Доступные стили границы. Показаны при заданном по умолчанию среднем (`medium`) значении их толщины

среднее (medium) значение. Если не указан ее цвет, граница окрашивается в цвет переднего плана элемента (такой же, что и текст).

В следующем примере к каждой стороне элемента я применила различные стили, чтобы показать вам специфицированные свойства стиля границы в действии (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;
}
```



Рис. 14.9. Стили границ, применяемые к отдельным сторонам элемента

Сокращенное свойство `border-style` функционирует согласно системе «по часовой стрелке» (TRouBLe), описанной ранее для отступов. Вы можете указать четыре значения границы для всех четырех сторон или меньшее число значений, если левая/правая и верхняя/нижняя границы совпадают. Эффект простеньких границ из предыдущего примера также может быть получен с помощью следующего свойства `border-style` (результат совпадает с тем, что показано на рис. 14.9):

```
border-style: solid dashed double dotted;
```

Свойства `border-width` (толщина границы)

Для указания значения толщины границы применяется одно из свойств `border-width`. Повторю еще раз: можно задать толщину границы на каждой стороне элемента с помощью специфицированного свойства:

```
border-top-width, border-right-width, border-bottom-width, border-left-width
```

- **Значения:** `длина` | `thin` | `medium` | `thick`.
- **По умолчанию:** `medium`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

или за один раз по часовой стрелке определить несколько сторон с помощью сокращенного свойства `border-width`:

`border-width`

- **Значения:** `длина` | `thin` | `medium` | `thick`.
- **По умолчанию:** `medium`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Самый распространенный способ указывать толщину границ — применять единицу измерения, выраженную в пикселях или ем. Однако вы можете указать одно из ключевых слов: `thin`, `medium` или `thick` (толстая, средняя или тонкая) — и предоставить отображение на усмотрение браузера.

В следующий пример я включила несколько значений (рис. 14.10). Обратите внимание, что свойство `border-style` также включено, поскольку, если этого не сделать, граница вовсе не будет отображаться:

```
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. Указание ширины границ

Свойства **border-color** (цвет границы)

Для указания значения цвета границы применяется одно из свойств `border-color`. Цвета границ указываются уже привычным нам способом — с помощью свойств, относящихся к сторонам границы:

`border-top-color`, `border-right-color`, `border-bottom-color`, `border-left-color`

- **Значения:** имя цвета или значение `RGB/HSL` | `transparent`.
- **По умолчанию** — значение свойства `color` для выбранного элемента.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

или при обращении к сокращенному свойству `border-color`. При указании цвета границы это значение перезапишет цвет переднего плана, установленный с помощью свойства `color` для выбранного элемента:

ЗНАЧЕНИЕ `TRANSPARENT` СВОЙСТВА `BORDER-COLOR`

Установка свойства `border-color` в значение `transparent` позволяет фону отображаться сквозь границу, но при этом сохраняет заданную толщину границы. Это полезно при формировании для границ эффектов ролловера (`:hover`), поскольку граница будет отображаться, даже если мышь не находится над элементом.

`border-color`

- **Значения:** имя цвета или значение `RGB/HSL` | `transparent`.
- **По умолчанию** — значение свойства `color` для выбранного элемента.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

О задании значений цвета вам уже все известно, в том числе и об использовании сокращенных свойств, поэтому следующий несложный пример вас не удивит (рис. ЦВ-14.11). Здесь предоставлены два значения для сокращенного свойства `border-color`, которые окрасят верх и низ элемента `div` в темнобордовый цвет (`maroon`), а левую и правую стороны — в цвет морской волны (`aqua`):

```
div#special {
    border-color: maroon aqua;
    border-style: solid;
    border-width: 6px;
    width: 300px;
    height: 100px;
}
```

ЭФФЕКТ РОЛЛОВЕРА

Эффект ролловера — активируемая каким-либо образом ссылка, обычно при помощи изменения шрифта, цвета или формы, когда посетитель сайта наводит на нее указатель мыши.

КОНТУРЫ

Контуры (`outline`) — это отличный инструмент для проверки макета страницы во время ее проектирования.

КОНТУРЫ CSS

Вокруг элемента можно нарисовать и контур. Контуры выглядят как границы, да и синтаксис их тот же, но между ними имеются существенные различия. Контуры, в отличие от границ, не рассчитываются по ширине блока элемента. Они накладываются на него сверху, ни на что не влияя. Контуры нарисованы на внешнем крае границы (если он указан) и перекрывают поля.

Поскольку контуры не оказывают влияния на макет, они служат отличным инструментом для проверки дизайна. Их подключение и отключение не изменяет ширины блока элемента, что позволяет увидеть, где и каким образом эти элементы расположены.

Свойства контура аналогичны свойствам границы с одним важным отличием: невозможно задать контуры для отдельных сторон поля элемента — или все, или ничего:

outline-style

- **Значения:** auto | solid | none | dotted | dashed | double | groove | ridge | inset | outset.
- **По умолчанию** — none.

Значения `outline-style` такие же, что и значения `border-style`, только к ним добавляется значение `auto`, позволяющее браузеру самостоятельно выбирать стиль контура в соответствии со своими установками. Также нельзя свойству `outline-style` присваивать значение `hidden`:

outline-width

- **Значения:** длина | thin | medium | thick.
- **По умолчанию** — medium.

Это то же самое, что и значения `border-width`.

outline-color

- **Значения:** название цвета или значение `RGB/HSL` | invert.
- **По умолчанию** — invert.

Заданное по умолчанию значение `invert` присваивает контуру цвет, инверсный к цвету фона, но поддержка браузерами этого свойства не слишком распространена.

outline-offset

- **Значения:** длина.
- **По умолчанию** — 0.

По умолчанию контур рисуется только за пределами края границы. Свойство `outline-offset` перемещает контур за границу на указанную величину.

outline

- **Значения:** `outline-style outline-width outline-color` (*стиль контура ширина контура цвет контура*).
- **По умолчанию:** — по умолчанию для отдельных свойств.

Сокращенное свойство `outline` комбинирует значения для `outline-style`, `outline-width` и `outline-color`. Не забывайте, что можете указывать их только сразу для всех сторон элемента:

```
div#story { outline: 2px dashed red; }
```

Комбинирование стиля, толщины и цвета границ

Разработчики CSS не экономили усилий, придумывая наиболее рациональные способы задания границ. При этом ими были созданы свойства для предоставления в одном объявлении значений стиля, толщины и цвета границы, поочередно для каждой из сторон. Так, можно указывать вид конкретных сторон:

border-top, border-right, border-bottom, border-left

- **Значения:** стиль границы толщина границы цвет границы.
- **По умолчанию** — по умолчанию для каждого свойства.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

или применять свойство `border` для одновременного задания всех четырех сторон:

border

- **Значения:** стиль границы толщина границы цвет границы.
- **По умолчанию** — по умолчанию для каждого свойства.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Значения для `border` и для свойств, относящихся к конкретным сторонам границы, могут включать значения стиля, толщины и цвета в любом порядке. Нет необходимости также объявлять все три значения, но, если не указать значение стиля границы (`border-style`), граница отображаться не будет.

Сокращенное свойство `border` работает немного иначе, чем другие рассмотренные нами сокращенные свойства, поскольку принимает один набор значений и всегда применяет их ко всем четырем сторонам элемента. Другими словами, используемая с другими сокращенными свойствами система TRBL (по часовой стрелке) в этом случае не работает.

Приведу несколько примеров определения допустимых сокращенных границ, чтобы дать вам представление о принципах работы этого свойства:

```
h1 { border-left: red .5em solid; } /* только левая граница */  
h2 { border-bottom: 1px solid; } /* только нижняя граница */  
p.example { border: 2px dotted #663; } /* для всех четырех сторон */
```

Свойства **border-radius** (скругление углов)

Может быть, вы хотите, чтобы углы блоков элементов были бы скруглены? Отлично, тогда свойство `border-radius` — это то, что вам нужно! Здесь, так же как и в предыдущих правилах, предлагаются свойства скругления конкретных углов:

border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius

- **Значения:** длина | процентное соотношение.
- **По умолчанию** — 0.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

```
p {  
width: 200px;  
height: 100px;  
background: darkorange;  
}
```

и сокращенное свойство `border-radius`:

`border-radius`

- **Значения:** 1, 2, 3 или 4 длины или процентные значения.
- **По умолчанию** — 0.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Для скругления угла элемента можно применить к нему одно из свойств `border-radius`, но вы заметите результат только в том случае, когда элемент имеет границу или цветной фон. Значения обычно устанавливаются в ем или пикселях. Не возбраняются и проценты, которые обеспечивают пропорциональное изменение скругления при изменении размеров блока элемента, но тут можно столкнуться с некоторой некорректностью поведения браузера.

Вы можете задать скругление индивидуально для каждого угла или применить сокращенное свойство `border-radius`. Если указать для свойства `border-radius` только одно значение, оно будет применено ко всем четырем углам. Четыре значения применяются по часовой стрелке, начиная с верхнего левого угла: верхний левый, верхний правый, нижний правый, нижний левый. Если ввести только два значения, первое будет использовано для верхнего левого и нижнего правого, а второе — для двух остальных углов.

Сравните различные значения `border-radius`, примененные к блокам элементов (рис. 14.12). Вы увидите, что в зависимости от того, каким образом установлены эти значения, можно получить много различных эффектов: от слегка скругленных углов до практически овальной капсулы.



`border-radius: 1em;`



`border-radius: 50px;`



`border-top-right-radius: 50px;`



```
border-top-left-radius: 1em;  
border-top-right-radius: 2em;  
border-bottom-right-radius: 1em;  
border-bottom-left: 2em;  
or  
border-radius: 1em 2em;
```

Рис. 14.12. Скругление углов блоков элементов с помощью свойств `border-radius`

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Все браузеры, начиная с 2010-го года, поддерживают свойства `border-radius` с помощью стандартного синтаксиса (то есть без применения префиксов). Существуют, правда, префиксные свойства для Firefox <3.6 и Safari <5.0, но они настолько устарели, что, вероятно, о них не стоит даже и вспоминать. Internet Explorer 8 и более ранние его версии вовсе не поддерживают `border-radius`, и восприятие вашего сайта не будет зависеть от того, скругленные в нем углы или нет, но это хорошая возможность попрактиковаться: не поддерживающие браузеры получают совершенно приемлемые прямые углы, а современные браузеры — улучшенный их вариант со скруглением.

Эллиптические углы

Все скругленные до сих пор углы представляют собой части идеальных кругов, но вы можете придать скруглению угла и эллиптическую форму, указывая два значения: первое — для горизонтального радиуса, а второе — для вертикального радиусов (рис. 14.13, А и Б):

- Ⓐ `border-top-right-radius: 100px 50px;`
- Ⓑ `border-top-right-radius: 50px 20px;`
`border-top-left-radius: 50px 20px;`

Если вы хотите применить сокращенное свойство, горизонтальные и вертикальные радиусы следует записать через слэш (в противном случае они перепутаются с различными значениями для углов). В следующем примере горизонтальный радиус для всех углов устанавливается 60 пикселов, а вертикальный радиус — 40 пикселов (рис. 14.13, С):

- Ⓒ `border-radius: 60px/40px;`

Если же нужно получить оригинальный эффект, обратите внимание на сокращенное свойство `border-radius`, с помощью которого задаются разные эллипсы для каждого из четырех углов. В следующем примере значения всех горизонтальных радиусов записаны слева от слэша по часовой стрелке (верхний левый, верхний правый, нижний правый, нижний левый), а соответствующие значения вертикальных радиусов записаны справа от него (рис. 14.13, D):

- Ⓓ `border-radius: 36px 40px 60px 20px / 12px 10px 30px 36px;`

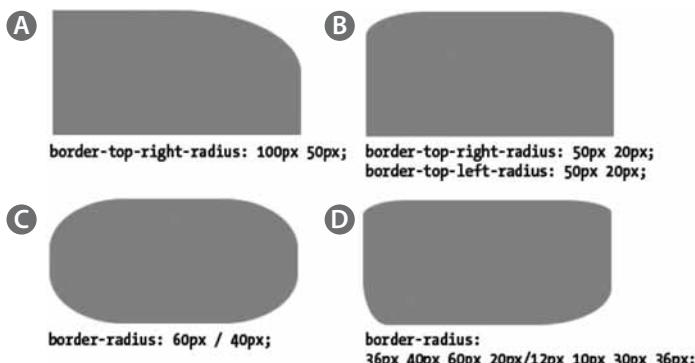


Рис. 14.13. Применение к блокам эллиптических углов

Ну вот, теперь можно попробовать свои силы в преобразовании границ. Упражнение 14.2 не только позволит вам попрактиковаться, но и познакомит с некоторыми соображениями о том, каким образом можно применять границы, чтобы сделать свои проекты более интересными.

УПРАЖНЕНИЕ 14.2. ДОБАВЛЕНИЕ ГРАНИЦ

В этом упражнении мы поэкспериментируем с границами на странице «Black Goose Bakery». Помимо задания границ вокруг разделов контента страницы, мы воспользуемся границами для выделения заголовков и — в качестве альтернативы стандартным приемам — для подчеркивания ссылок.

1. Откройте в текстовом редакторе файл `bakery-styles.css`, если вы этого еще не сделали. Начнем с основ и воспользуемся сокращенным свойством `border`, чтобы установить вокруг элемента `main` двойную желтовато-коричневую границу-рамку (`tan double rule`). Для этого добавьте в существующее правило для `main` новое объявление:

```
main {  
...  
    padding: 1em;  
    border: double 4px #EADDC4;  
}
```

2. Теперь поэкспериментируйте со свойствами `border-radius`, чтобы задать закругление определенным углам границ разделов `main` и `aside`. На мой взгляд, радиуса закругления в 25 пикселов вполне достаточно. И я предпочитаю пиксели, а не `em`, чтобы радиус не масштабировался вместе с текстом. Начните с добавления этого объявления в стили для `main`:

```
border-radius: 25px;
```

И придайте соответствующее закругление только верхнему правому углу раздела `aside`:

```
aside {  
...  
    border-top-right-radius: 25px;  
}
```

3. Чтобы еще попрактиковаться, установим декоративную границу с двух сторон заголовков, относящихся к выпечке (`h3`). Для этого найдите правило для элементов `h3` и добавьте объявление, которое установит сплошную (`solid`) границу толщиной в 1 пиксель для верхней стороны заголовка. Добавьте также еще одно правило, которое установит для левой стороны заголовка более толстую сплошную границу толщиной в 3 пикселя. Чтобы границы имели тот же цвет, что и текст, указывать цвет границ не нужно. Наконец, чтобы текст не попал на левую границу, добавьте небольшой отступ (`1em`) слева от заголовка контента:

```
h3 {  
...  
    border-top: 1px solid;  
    border-left: 3px solid;  
    padding-left: 1em;  
}
```

4. И последнее, что необходимо сделать, это заменить стандартное подчеркивание ссылок декоративной нижней границей. Начните с отключения подчеркивания для всех ссылок и добавьте такое правило в раздел `link styles` таблицы стилей:

```
a {
    text-decoration: none;
}
```

Затем добавьте туда же объявление состоящей из точек границы толщиной в 1 пиксел, устанавливаемой у нижнего края ссылок:

```
a {
    text-decoration: none;
    border-bottom: 1px dotted;
}
```

При добавлении к элементу границы часто бывает необходимо создать небольшой отступ, чтобы не допустить наложения элементов:

```
a {
    text-decoration: none;
    border-bottom: 1px dotted;
    padding-bottom: .2em;
}
```

Теперь можно сохранить таблицу стилей и перезагрузить в браузере файл `bakery.html`. На рис. 14.14 показано, какой вид получит веб-страница.

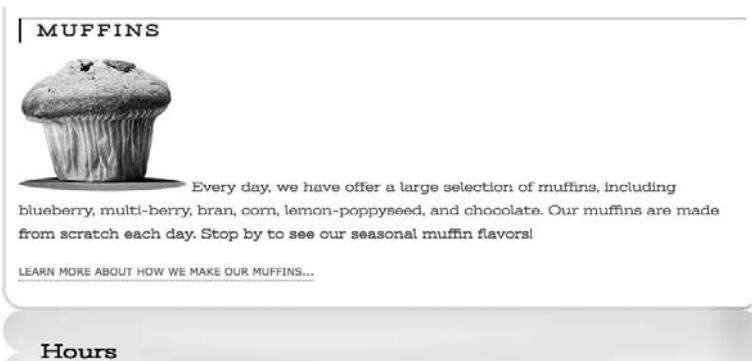


Рис. 14.14. Результаты добавления границ

ГРАНИЦЫ, СОСТАВЛЕННЫЕ ИЗ ИЗОБРАЖЕНИЙ

В CSS3 появились свойства `border-image-*`, позволяющие заполнять стороны и углы границ блока выбранными вами изображениями (рис. 14.15).

Изображения для границ применяются с помощью набора из 5 свойств:

- `border-image-source` — задает расположение изображения;
- `border-image-slice` — разделяет изображение на 9 секций с помощью задания размеров смещения;

- `border-image-width` — задает толщину границы;
- `border-image-repeat` — определяет, будет ли изображение растягиваться по сторонам или повторяться;
- `border-image-outset` — отодвигает границу от контента на заданную величину.

Имеется также сокращенное свойство `border-image`, комбинирующее отдельные свойства с помощью следующего синтаксиса:

`border-image: начальная секция / толщина / смещение повтор;`

Правила стиля для границы изображения, показанного на рис. 14.15, следующие:

```
border: 5px solid #d1214a; /* красный */
border-image: url(fancyframe.png) 55 fill / 55px /
25px stretch;
```

Сокращенное свойство `border` поддерживает для границы стиль запасного варианта на случай, если изображение не загружается или свойство `border-image` не поддерживается браузером.

Приведенное здесь правило `border-image` предписывает браузеру применить к границе изображение `fancyframe.png`, разрезав его на секции по 55 пикселов от краев и использовав центральную часть изображения для заполнения (`fill`) центральной области блока. Толщина границы при этом должна составлять 55 пикселов (`55px`), и изображение должно отстоять от краев контента на 25 пикселов (`25px`). Наконец, секции изображения, составляющие его стороны, должны быть с помощью свойства `stretch` растянуты до соответствия ширине и высоте блока.

Если вы хотите получить дополнительную информацию по этой теме, обратитесь к моей статье «[Border Images](#)». Эту статью можно загрузить с сайта: learningwebdesign.com/articles/. Более подробная информация об использовании изображений для создания границ предоставлена следующими ресурсами:

- «[The CSS Background and Borders Module Level 3](http://www.w3.org/TR/css-backgrounds-3/#the-border-image-source)» (www.w3.org/TR/css-backgrounds-3/#the-border-image-source);
- «[The `border-image` listing on CSS-Tricks](http://css-tricks.com/almanac/properties/b/border-image)» (css-tricks.com/almanac/properties/b/border-image/) — более простыми словами.



Рис. 14.15. Примеры применения изображений к границам блоков

ПОЛЯ

Поля представляют собой необязательное пространство, которое можно добавлять за пределами границы. Поля предотвращают взаимное перекрытие элементов либо «столкновение» их с краем окна браузера или области просмотра.

Специфицированные свойства для полей отдельных сторон:

`margin-top, margin-right, margin-bottom, margin-left`

- **Значения:** `длина | процентное соотношение | auto`.
- **По умолчанию** — `auto`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

и сокращенное свойство `margin`:

`margin`

- **Значения:** `длина | процентное соотношение | auto`.
- **По умолчанию** — `auto`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

функционируют так же, как уже рассмотренные свойства `padding`, однако у полей имеются некоторые особенности, о которых следует знать.

Свойства полей просты в использовании. Можно указать размер поля для отображения его на каждой стороне элемента или использовать свойство `margin` для одновременного указания всех полей.

Сокращенное свойство `margin` работает так же, как и сокращенное свойство `padding`. При вводе четырех значений они применяются к сторонам элемента по часовой стрелке (сверху, справа, снизу, слева). Если указано три значения, среднее значение применяется и к левой стороне, и к правой. Если указаны два значения, первое используется для верхнего и нижнего краев, а второе — для левого и правого. Наконец, одно значение будет применено ко всем четырем сторонам элемента.

Как и в большинстве связанных с Интернетом случаев использования размерных величин, наиболее распространенными способами задания полей являются `em`, пиксели и проценты. Однако, если указать процентное соотношение, оно рассчитывается на основе *ширины* родительского элемента. Если ширина родительского элемента изменится, будут изменены и поля для всех четырех сторон дочернего элемента (это также относится и к отступам). Ключевое слово `auto` позволяет браузеру заполнить необходимое поле или все доступное пространство (см. врезку «*Центрирование полей*»).

На рис. 14.16 показаны примеры задания полей. К элементам в этих примерах я добавила красную обводку (`2px solid red`), чтобы их границы были четче видны.

ЗАДАННЫЕ ПО УМОЛЧАНИЮ ПОЛЯ БРАУЗЕРА

Возможно, вы заметили, что вокруг заголовков, абзацев и других блоковых элементов пространство добавляется автоматически. Так функционирует стандартная таблица стилей браузера, задающая поля до и после этих элементов.

При этом следует иметь в виду, что браузер применяет для полей и отступов свои собственные значения. Эти значения и будут использоваться, если вы не переопределите их собственными правилами стиля.

Если вы работаете над дизайном и встречаете некие таинственные пустые области, которые вами не добавлялись, тому виной как раз заданные по умолчанию стили браузера. Для уточнения подобных обстоятельств я рекомендую применять средство просмотра кода браузера (Web Inspector), которое покажет источник всех примененных к элементу стилей. Или же, если вы не желаете озабочиваться стилями браузера, — сбросьте отступы и поля для всех элементов до нуля, о чем рассказывается в разд. «Сброс CSS» главы 19.

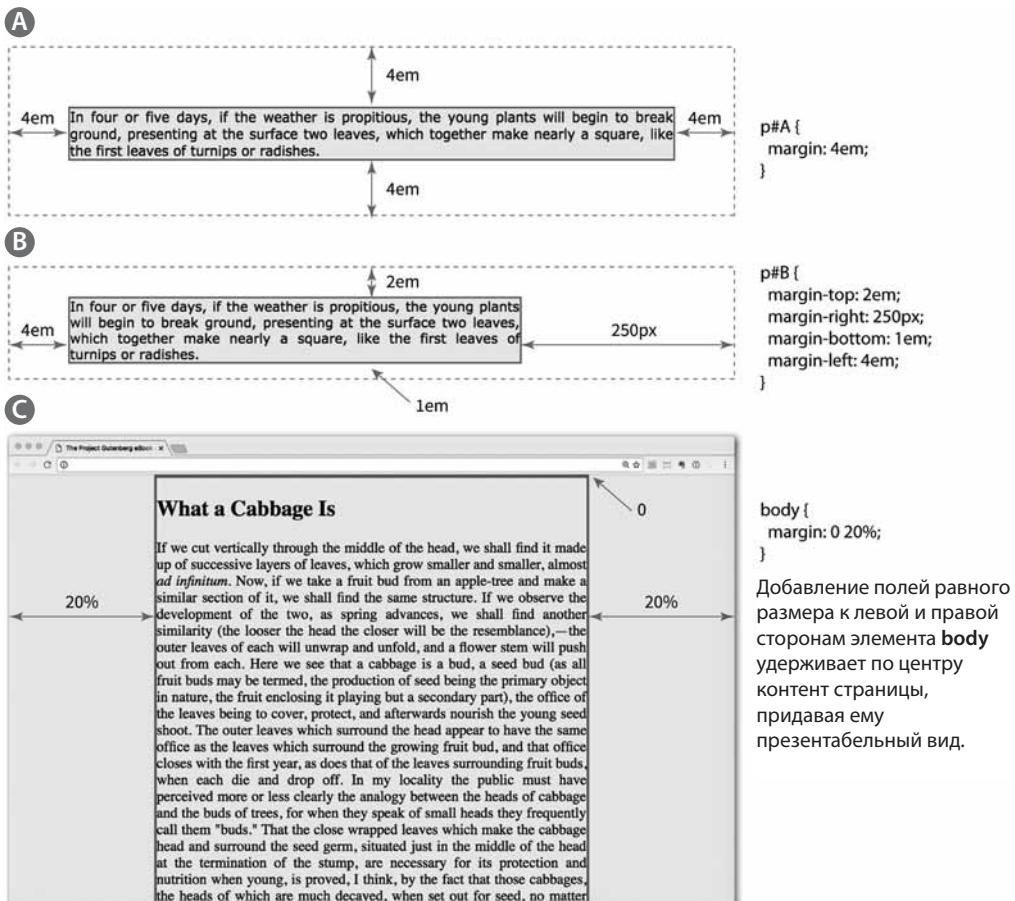


Рис. 14.16. Применение полей к разделу `body` и отдельным элементам

Пунктирные линии здесь присутствуют только для обозначения внешних краев полей и не отображаются в браузере:

```

Ⓐ p#A {
    margin: 4em;
    border: 2px solid red;
    background: #e2f3f5;
}

Ⓑ p#B {
    margin-top: 2em;
    margin-right: 250px;
    margin-bottom: 1em;
    margin-left: 4em;
    border: 2px solid red;
    background: #e2f3f5;
}

Ⓒ body {
    margin: 0 20%;
    border: 3px solid red;
    background-color: #e2f3f5;
}

```

ЦЕНТРИРОВАНИЕ ПОЛЕЙ

Выбор значения поля `auto` для левого и правого полей элемента приводит к центрированию элемента в его контейнере.

Обратите внимание на пример Ⓒ. Здесь свойство `margin` применяется к элементу `body` документа. Для этого конкретного случая я установила верхнее поле равным нулю (0), чтобы раздел `body` начинался вплотную от верхнего края окна браузера (см. рис. 14.16, Ⓓ). А добавление полей равного размера к левой и правой сторонам элемента `body` удерживает по центру контент страницы, придавая ему презентабельный вид.

ДОБАВЛЕНИЕ ПОЛЯ К РАЗДЕЛУ `BODY`

Добавление полей к разделу `body` элемента создает пространство между страницей контента и краями окна просмотра.

Поведение полей

Написать правила, создающие поля вокруг к HTML-элементов, достаточно легко, но при этом важно знать некоторые особенности поведения полей.

Коллапс поляй

Наиболее важным моментом в поведении полей, о котором следует знать, является то, что верхние и нижние смежные поля соседних элементов могут *коллапсировать*: вместо того чтобы просто соединиться, смежные поля перекрываются, и остается только то поле, которое имеет наибольшее значение.

Взглянув для примера на абзацы Ⓐ и Ⓑ из рис. 14.16, заметим, что верхний элемент Ⓐ имеет нижнее поле, равное `4em`, а нижний элемент Ⓑ имеет верхнее поле `2em`. Так вот, если бы эти абзацы следовали в тексте друг за другом, результирующее пространство для полей между ними не составило бы `6em`. Вместо этого смежные поля бы коллапсировали, и результирующее поле между абзацами составило бы `4em`, то есть наибольшее из указанных значений. Это и показано на рис. 14.17.

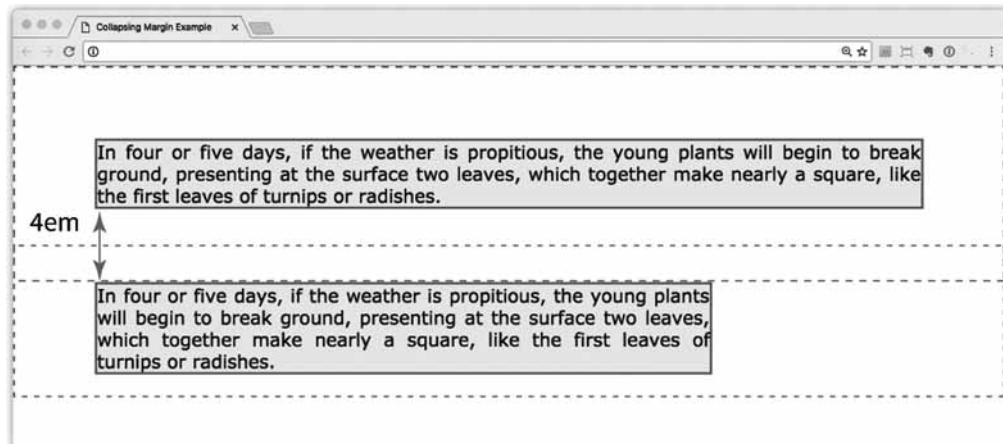


Рис. 14.17. Вертикальные поля соседних элементов коллапсируют, поэтому применяется только наибольшее значение поля из заданных соседним полям

Единственный вариант, при котором верхнее и нижнее поля не коллапсируют, это когда создаются плавающие или абсолютно позиционированные элементы (речь о них пойдет в главе 15). А вот поля слева и справа никогда не коллапсируют, поэтому проблем с ними меньше.

Поля строчных элементов

Вы можете задать верхние и нижние поля для строчных текстовых элементов (или «незаменяемых строчных элементов», применяя правильную CSS-терминологию), однако это не добавит вертикальное пространство над и под элементом, да и высота строки не изменится. А вот в случае применения к ним левого и правого полей, до и после текста в потоке элемента появится свободное пространство поля, даже если этот элемент состоит из нескольких строк (рис. 14.18, *вверху*).

Поля, примененные к заменяемым строчным элементам (таким, как изображения), отображаются со всех сторон и, следовательно, влияют на высоту строки (рис. 14.18, *внизу*).

КОЛЛАПС СМЕЖНЫХ ПОЛЕЙ

Вертикальные смежные поля коллапсируют, и применяется только значение наибольшего поля.

ЕЩЕ О КОЛЛАПСЕ ПОЛЕЙ

Если расстояние между элементами и вокруг них ведет себя непредсказуемо, иногда причиной тому может быть коллапс полей. Далее приведено несколько статей, где подробно повествуется о деструктивном поведении полей при их коллапсе. И, хотя статьи эти написаны давно, информация достоверна и поможет вам понять, что происходит «за кулисами» ваших макетов:

- «No Margin for Error» от Анди Бадда (Andy Budd): www.andybudd.com/archives/2003/11/no_margin_for_error;
- «Uncollapsing Margins» от Эрика Мейера (Eric Meyer): www.complex-spiral.com/publications/uncollapsingmargins.

```
em { margin: 2em; }
```

In four or five days, if the weather is propitious, the young plants will begin to **break ground**, presenting at the surface two leaves, which together make nearly a square, like the first leaves of turnips or radishes.

```
img { margin: 2em; }
```

In four or five days, if the weather is propitious, the young



plants will begin to break ground, **presenting at the surface two leaves, which together make nearly a square, like the first leaves of turnips or radishes.**

Рис. 14.18. Применение полей к строчным элементам: у незаменяемых элементов (вверху) отображаются только горизонтальные поля, у заменяемых элементов (таких, как изображения) поля отображаются со всех сторон

Поля с отрицательными значениями

Следует отметить, что для полей можно задавать и отрицательные значения. В таком случае содержимое, отступы и границы элемента перемещаются в противоположном направлении по сравнению с результатом для положительного значения поля.

Рассмотрим пример. На рис. ЦВ-14.19 показаны два соседних абзаца с границами, окрашенные в разные цвета. В ситуации, приведенной слева, добавление в верхний абзац *нижнего* поля величиной 3ем приводит к отодвиганию следующего абзаца на эту величину *вниз*. Если же задать верхнему абзацу отрицательное значение *нижнего* поля (-3ем), следующий за ним абзац переместится на эту же величину *вверх* и перекроет верхний абзац, имеющий отрицательное значение поля (*справа*).

Никакого смысла в таком наложении нет, и вряд ли вы на самом деле сделаете блоки текста перекрывающимися. Однако приведенный пример показывает, что, применяя поля с положительными и отрицательными значениями, можно перемещать элементы по странице. На этом как раз и основывалась устаревшая ныне методика CSS-верстки.

А сейчас давайте выполним *упражнение 14.3* и с помощью полей добавим пространство между частями веб-страницы «Black Goose Bakery».

УПРАЖНЕНИЕ 14.3. ДОБАВЛЕНИЕ ПОЛЕЙ ВОКРУГ ЭЛЕМЕНТОВ

В этом упражнении мы настроим поля вокруг элементов на странице нашей пекарни. Начнем мы с задания полей для всего документа, а потом внесем

изменения в каждый из разделов сверху вниз. Откройте для этого файл `bakery-styles.css` в текстовом редакторе.

1. Обычно поля для элемента `body` устанавливаются равными нулю, что позволяет сбросить заданные по умолчанию в браузере настройки полей. Добавьте это объявление полей к стилям `body`, затем сохраните файл и откройте его в браузере. Вы увидите, что элементы теперь расположены от самого края окна без пробелов между ними.

```
body {  
...  
    margin: 0;  
}
```

ЕЩЕ ОБ ЕДИНИЦАХ ИЗМЕРЕНИЯ

Если значение равно 0, нет необходимости указывать какую-либо конкретную единицу измерения.

2. Если вы достаточно внимательны, то, возможно, заметили, что над цветной навигационной панелью имеется немного свободного пространства. Так происходит потому, что верхнее поле списка `ul` сдвигает элемент `nav` вниз от верхнего края браузера. Исправим этот недочет, добавив новое правило стиля в разделе `nav styles` таблицы стилей:

```
nav ul {  
    margin: 0;  
}
```

3. Полями удобно пользоваться для перемещения элементов в макете. Например, если необходимо сдвинуть заголовок `h1` с логотипом немного вниз, добавим поле к его верхнему краю. Поэкспериментируем с несколькими значениями, прежде чем выбрать `1.5em` для нового правила стиля:

```
header h1 {  
    margin-top: 1.5em;  
}
```

Желательно, чтобы вступительный (`intro`) абзац в заголовке располагался ближе к логотипу, поэтому применим верхнее поле с отрицательным значением, чтобы его поднять. Добавим это объявление к существующему правилу стиля:

```
header p {  
...  
    margin-top: -12px;  
}
```

4. Добавим в раздел `main` величину поля `2.5%` для всех сторон:

```
main {  
...  
    margin: 2.5%;  
}
```

5. Добавим также дополнительное пространство над заголовками `h3` области `main`. Я выбирала значение, равное `2.5em`, но вы можете вводить и другие значения на свое усмотрение:

```
h3 {  
...  
    margin-top: 2.5em;  
}
```

6. Наконец, добавим пространство вокруг элемента `aside`, а для большего эффекта зададим при этом для каждой стороны свое значение. Примените поля: `1em` сверху, `2.5%` справа, `0` снизу и `10%` слева. Я доверяю вам сделать это самостоятельно. Можно ли выполнить все эти изменения с помощью одного объявления? Если вам необходимо проверить, все ли вы сделали правильно, вы можете сравнить свою работу с готовой версией страницы «Black Goose Bakery», доступной в комплекте материалов для упражнений этой главы.
7. Снова сохраните таблицу стилей и перезагрузите страницу в браузере. Она должна выглядеть, как показано на рис. 14.20. Дизайн не самый привлекательный, особенно когда окно браузера распахнуто во всю ширь (слева). Впрочем, если сузить окно браузера, дизайн страницы покажется не столь уж и плохим (справа) — так могла бы выглядеть версия страницы для небольшого экрана в адаптивном дизайне. (Спорим, вам не терпится дождаться, когда в главе 17, посвященной адаптивному веб-дизайну, будет показано, как это исправить!)



Рис. 14.20. Домашняя страница «Black Goose Bakery» после добавления отступов, границ и полей

ПРИСВАИВАНИЕ ТИПОВ ОТОБРАЖЕНИЯ

Раз уж мы говорим о блоках и моделях CSS-макета, самое время представить свойство `display`. Вы уже знакомы с поведением отображения для блочных и строчных элементов. Хотя HTML присваивает поведение отображения (или *тип*

отображения, применяя современный CSS-термин) определяемым элементам, имеются и другие языки на основе XML, которые используют CSS, но не выполняют такого присваивания. По этой причине — чтобы разработчики могли определить, как элементы должны вести себя в макетах, — и было создано свойство `display`:

display

- **Значения:** `inline` | `block` | `run-in` | `flex` | `grid` | `flow` | `flow-root` | `list-item` | `table` | `table-row-group` | `table-header-group` | `table-footer-group` | `table-row` | `table-cell` | `table-column-group` | `table-column` | `table-caption` | `ruby` | `ruby-base` | `ruby-text` | `ruby-base-container` | `ruby-text-container` | `inline-block` | `inline-table` | `inline-flex` | `inline-grid` | `contents` | `none`.
- **По умолчанию** — `inline`.
- **Применение** — ко всем элементам.
- **Наследование** — да.

Свойство `display` определяет тип поля элемента, которое генерирует элемент в макете. В дополнение к знакомым типам отображения `inline` и `block` можно настроить отображение элементов в виде списка или различных частей таблицы. Имеется также ряд значений для аннотации `ruby`, применяемой для восточноазиатских языков. Как можно судить по размерам списка значений, нам предлагается большое число типов отображения, но лишь некоторые из них применяются в повседневной практике.

Назначение типа отображения полезно для достижения в макете требуемых эффектов при сохранении семантики исходного HTML-кода. Например, обычной практикой является представление элементов `li` (которые обычно отображаются с характеристиками блочных элементов) в виде строчных элементов, превращающих список в горизонтальную панель навигации. Вы также можете отобразить строчный (`inline`) элемент `a` (якорь) в виде блока, чтобы придать ему определенную ширину и высоту:

```
ul.navigation li { display: inline; }
ul.navigation li a { display: block; }
```

Другим полезным значением свойства `display` является `none`, с помощью которого контент полностью удаляется из нормального потока. В отличие от значения `visibility: hidden`, которое превращает элемент в невидимый, но удерживает занимаемое им пространство пустым, свойство `display: none` удаляет контент вместе с занимаемым им пространством.

Одно из популярных применений свойства `display: none` заключается в предотвращении отображения определенного содержимого исходного документа

ЕЩЕ О CSS-СВОЙСТВЕ DISPLAY

Имейте в виду, что изменение представления HTML-элемента с помощью CSS-свойства `display` не меняет определения этого элемента в HTML как блочного или строчного. Помещение блочного элемента в строчный всегда будет недействительным, независимо от его роли в отображении.

в некоторых ситуациях — например, когда страница печатается или отображается на устройствах с небольшими экранами. Так, вы можете отображать интернет-адреса (URL) для ссылок в печатном документе, но не показывать их на экране компьютера, где ссылки являются интерактивными.

РАЗЛИЧНЫЕ МЕТОДЫ СОКРЫТИЯ ЭЛЕМЕНТОВ

В работе «Five Ways to Hide Elements in CSS» Бальджита Рати (Baljeet Rath) (www.sitepoint.com/five-ways-to-hide-elements-in-css) сравниваются различные методы сокрытия элементов, включая `display: none`.

Учтите также, что контент, для которого свойство `display` получает значение `none`, продолжает загружаться в составе документа. Настройка отображения какого-либо контента как `display:none` для устройств с маленькими экранами поможет уменьшить страницу, но никак не сократит время использования данных или время загрузки.

ТЕНИ ДЛЯ БЛОКА

Мы достигли последней остановки в турне по блочным элементам. В главе 12 было представлено свойство `text-shadow`, которое добавляло тень к тексту. Свойство `box-shadow` применяет тень ко всему видимому блоку элементов (исключая поля):

`box-shadow`

- **Значения:** '*смещение по горизонтали*' '*смещение по вертикали*' '*величина размытия*' '*величина разброса*' `color inset | none`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Вы вполне можете предугадать значения свойства `box-shadow`, поскольку ранее уже рассматривали свойство `text-shadow`: нужно указать смещение (`offset`) по горизонтали и вертикали, величину размытия тени (`blur`) и цвет. Для затенения блока можно также указать величину разброса (`spread`), которая увеличивает (или уменьшает при отрицательных значениях) размер тени. По умолчанию, цвет тени совпадает со цветом переднего плана элемента, но его можно переопределить.

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Браузерам, выпущенным до 2011-го года, требуются префиксы поставщиков для реализации свойства `box-shadow`. Браузером Internet Explorer версии 8 и более ранних затенение блоков не поддерживается вовсе. Эта ситуация вполне пригодна для постепенного улучшения — вполне вероятно, что блок без затенения в достаточной степени приемлем для пользователей, которые упорно применяют старые версии браузера.

На рис. 14.21 показаны результаты применения следующих примеров кода:

- Ⓐ `box-shadow: 6px 6px gray;`
- Ⓑ `box-shadow: 6px 6px 5px gray; /* 5 pixel blur */`
- Ⓒ `box-shadow: 6px 6px 5px 10px gray; /* 5px blur, 10px spread */`

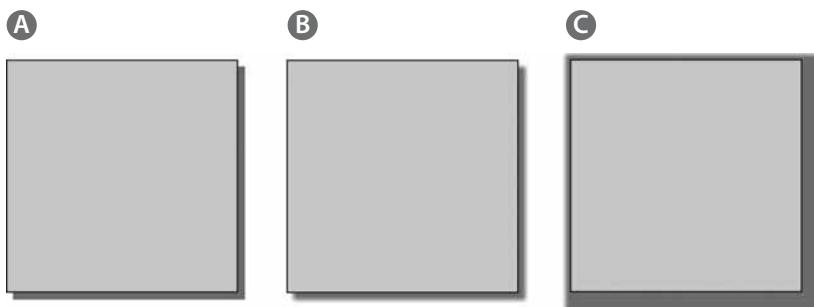


Рис. 14.21. Добавление теней вокруг элемента с помощью свойства `box-shadow`

Пример кода **A** добавляет блоку простую тень на 6 пикселов вправо и 6 пикселов вниз, без размытия или разброса. Пример **B** добавляет значение размытия в 5 пикселов, а пример **C** показывает влияние значения разброса в 10 пикселов. Затенение границ всегда применяется к области за границей элемента (или к тому месту, где она находилась бы, если граница не указана явно). Если элемент имеет прозрачный или полупрозрачный фон, затенение блока в области за элементом будет отсутствовать.

Можно сделать теневую визуализацию по краям поля видимого элемента, добавив к правилу ключевое слово `inset`. Тогда элемент будет выглядеть так, как будто вдавливается в экран (рис. 14.22):

```
box-shadow: inset 6px 6px 5px gray;
```

Со свойством `text-shadow`, как и со свойством `text-shadow` можно задавать для элемента несколько теней, указывая их значения через запятую в списке. Тени, значения для которых приведенные вначале, располагаются сверху, а последующие тени — располагаются за ними в порядке их появления в списке.

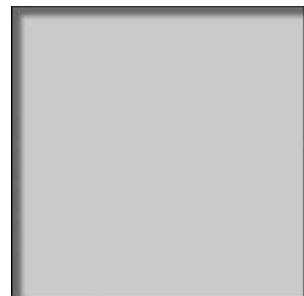


Рис. 14.22. Затенение блока с помощью ключевого слова `inset` отображается внутри блочного элемента

НЕ ЗЛОУПОТРЕБЛЯЙТЕ ТЕНЯМИ!

Затенения для блока или текста и градиенты потребляют много ресурсов процессора, потому что бремя их интерпретации и отображения переносится на браузер. Чем больше используются эти эффекты, тем ниже производительность работы, а, как известно, производительность — это очень важно для сети. Поэтому не злоупотребляйте такими эффектами.

КОНТРОЛЬНЫЕ ВОПРОСЫ

К этому моменту вы уже должны хорошо разбираться в блоках элементов и методах манипуляции пространством как внутри, так и вне их. В следующей главе мы начнем перемещать блоки по странице, но сначала в следующем далее тесте попрактикуемся в написании правил для отступов, границ и полей.

НЕКОТОРЫЕ ПОЯСНЕНИЯ К РИС. ЦВ-14.23

- Внешние края полей обозначены пунктирными синими линиями.
- Все необходимые размеры показаны синим цветом.
- Границы блоков показаны либо черным цветом, либо красным.

Итак, вам надо написать объявления, которые сформируют эффекты, показанные в каждом из примеров, приведенных на рис. ЦВ-14.23. Все имеющиеся там абзацы оформлены в соответствии с общим правилом, устанавливающим для них размеры и цвет фона. Вам нужно предоставить объявление свойства, связанного с блоком. Ответы, как всегда, приведены в *приложении 1*.

- A**
B
C
D
E
F

ОБЗОР CSS: СВОЙСТВА БЛОКА

В табл. 14.1 в алфавитном порядке приводятся краткие описания свойств блока.

Таблица 14.1. Свойства блока

Свойства	Описание
Border	Сокращенное свойство, объединяющее свойства границы
border-top border-right border-bottom border-left	Определяют свойства границ для каждой стороны элемента
border-color	Сокращенное свойство для указания цвета границ
border-top-color border-right-color border-bottom-color border-left-color	Определяют цвет границы для каждой стороны элемента
border-image	Добавляет изображение внутри границы
border-image-outset	Определяет, как далеко изображение границы должно быть расположено от границы
border-image-repeat	Способ, которым изображение заполняет стороны границы

Табл. 14.1 (продолжение)

Свойства	Описание
border-image-slice	Точки, в которых границы изображения должны быть разделены на углы и стороны
border-image-source	Расположение файла изображения, которое будет использоваться для изображения границы
border-image-width	Ширина пространства, которое должна занимать граница изображения
border-radius	Сокращенное свойство для скругления углов поля видимого элемента
border-top-left-radius border-top-right-radius border-bottom-right-radius border-bottom-left-radius	Определяют кривую радиуса для каждого отдельного угла
border-style	Сокращенное свойство для указания стиля границ
border-top-style border-right-style border-bottom-style border-left-style	Определяют стиль границы для каждой стороны элемента
border-width	Сокращенное свойство для указания ширины границ
border-top-width border-right-width border-bottom-width border-left-width	Определяют толщину границы для каждой стороны элемента
box-sizing	Указывает, применяются ли размеры ширины и высоты к блоку содержимого или к границе
box-shadow	Добавляет тень вокруг поля видимого элемента
display	Определяет тип блока элемента, который генерирует элемент
height	Определяет высоту поля содержимого элемента или границы блока
margin	Сокращенное свойство для указания пространства полей вокруг элемента
margin-top margin-right margin-bottom margin-left	Определяют поле для каждой стороны элемента
max-height	Определяет максимальную высоту элемента
max-width	Определяет максимальную толщину элемента
min-height	Определяет минимальную высоту элемента
min-width	Определяет минимальную ширину элемента

Табл. 14.1 (окончание)

Свойства	Описание
outline	Сокращенное свойство, применяемое для добавления контура вокруг элемента
outline-color	Устанавливает цвет контура
outline-offset	Устанавливает пространство между контуром и внешним краем границы
outline-style	Устанавливает стиль контура
outline-width	Устанавливает ширину контура
overflow	Определяет, как обрабатывать контент, который не помещается в области контента
padding	Сокращенное свойство для указания пространства между областью содержимого и границей
padding-top padding-right padding-bottom padding-left	Определяют количество отступов для каждой стороны элемента
width	Определяет ширину поля содержимого элемента или границы блока

ГЛАВА 15

ПЛАВАЮЩИЕ ЭЛЕМЕНТЫ И ПОЗИЦИОНИРОВАНИЕ

В этой главе...

- ▶ *Элементы, плавающие слева и справа*
- ▶ *«Очистка» плавающих элементов*
- ▶ *Удерживание плавающих элементов*
- ▶ *Создание форм обтекания текстом*
- ▶ *Относительное позиционирование*
- ▶ *Абсолютное позиционирование и содержащие блоки*
- ▶ *Фиксированное позиционирование*

К этому моменту мы рассмотрели десятки CSS-свойств, позволяющих изменять внешний вид текстовых элементов и генерированных ими блоков. Правда, до сих пор речь шла о форматировании элементов по мере их отображения в потоке документа.

В этой главе мы рассмотрим плавание и позиционирование — методы CSS, позволяющие выйти за пределы нормального потока и свободно изменять расположение элементов на странице. *Режим плавания* позволяет перемещать элемент влево или вправо и разрешает его обтекание текстом. *Позиционирование* — это способ указать местоположение элемента в любом месте страницы с точностью до пикселя.

Но, прежде чем заняться перемещением элементов, удостоверимся, что мы хорошо знакомы с тем, как они ведут себя в нормальном потоке.

НОРМАЛЬНЫЙ ПОТОК

Нормальный поток рассматривался нами в предыдущих главах, и сейчас мы лишь немножко освежим эти знания. В модели CSS-макета текстовые элементы в порядке их появления в исходном коде располагаются сверху вниз и слева направо — для языков, предполагающих чтение слева направо. Блочные элементы размещаются один над другим и заполняют доступную ширину окна браузера или другого содержащего их элемента. Строчные элементы и строки текстовых символов следуют друг за другом, заполняя элементы блока.

О ЯЗЫКАХ, ПРЕДПОЛАГАЮЩИХ ЧТЕНИЕ СПРАВА НАЛЕВО

Для языков, предполагающих чтение справа налево, — таких, как арабский и иврит, нормальный поток организуется в направлении сверху вниз и справа налево.

Объекты в нормальном потоке влияют на расположение окружающих их объектов. Такого поведения мы и привыкли ожидать на веб-страницах — элементы не перекрываются и не сливаются, а оставляют место друг для друга.

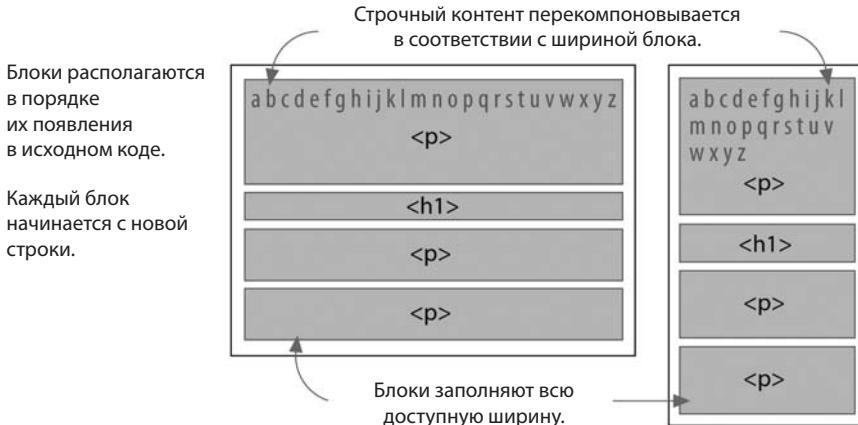


Рис. 15.1. Еще один пример поведения нормального потока

Мы уже видели все это раньше, но здесь мы сосредоточим свое внимание на том, находятся ли элементы в потоке или не связаны с ним. Плавание и позиционирование по-разному изменяют взаимосвязь элементов с нормальным потоком. Давайте сначала посмотрим на специфику поведения плавающих элементов (или «поплавков» для краткости).

ПЛАВАЮЩИЕ ЭЛЕМЕНТЫ

РЕЖИМ ПЛАВАНИЯ

Режим плавания позволяет смещать элемент влево или вправо, а также вызывает обтекание его окружающим текстом.

В общем случае свойство `float` перемещает элемент как можно дальше влево или вправо, позволяя последующему контенту его обтекать. Это уникальная и весьма полезная функция CSS:

`float`

- **Значения:** `left` | `right` | `none`.
- **По умолчанию** — `none`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Наилучший способ объяснить, что такое плавание элемента, — продемонстрировать его. В следующем примере свойство `float` применяется к элементу `img`, который переходит в плавающий режим и перемещается вправо. На рис. 15.2 показано, каким образом абзац и содержащееся в нем изображение отображаются по умолчанию (*вверху*) и какой все это примет вид после применения свойства `float` (*внизу*):

Разметка

```
<p> After the cream is  
frozen rather stiff, ...
```

Правило стиля

```
img {  
    float: right;  
}
```

Встроенное изображение
в нормальном потоке



Пространство за изображением
остается чистым

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.

Встроенное изображение
«плывет» вправо

Изображение перемещается,
и текст обтекает его

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.



Рис. 15.2. Компоновка изображения в нормальном потоке (*вверху*) и со свойством `float` (*внизу*)

Мы получили весьма любопытный эффект: от незанятого места на странице избавились, однако изображение оказалось слишком прижатым к тексту. Как вы полагаете, не надо ли слегка отодвинуть изображение от обтекающего его текста? Если вы догадались, что нужно добавить к изображению поле, вы абсолютно правы. Я добавила поля размером в `1em` со всех сторон изображения, воспользовавшись свойством `margin` (рис. 15.3). Замечаете, как свойства блока, применяемые совместно, улучшают макет страницы?

```
img {
    float: right;
    margin: 1em;
}
```

Пунктирная линия показывает край внешнего поля
(в браузере она не отображается)

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.



Рис. 15.3. Добавление поля размером 1em вокруг плавающего изображения

Предыдущие две иллюстрации демонстрируют некоторые ключевые моменты поведения плавающих элементов:

- *плавающий элемент подобен острову в потоке* — в первую очередь можно заметить, что изображение удаляется с позиции в нормальном потоке, но продолжает влиять на окружающий его контент. На рис. 15.2 весь текст абзаца перекомпонован с тем, чтобы освободить место для плавающего элемента `img`. Иногда «поплавки» сравнивают с находящимися в потоке островками — сами они не находятся в потоке, и их должен обтекать сам поток. Такое поведение плавающих элементов уникально;
- *«поплавки» остаются в области контента содержащего их элемента* — важно также отметить, что плавающее изображение размещается внутри *области контента* (внутренних краев) содержащего его абзаца и не распространяется на область отступа абзаца;
- *поддерживаются поля* — кроме того, со всех сторон плавающего изображения могут задаваться поля, как показано с помощью пунктирной линии на рис. 15.3. Другими словами, плавает (перемещается) весь блок элемента: от одного его внешнего края до другого.

Плавающие строчные и блочные элементы

До сих пор были рассмотрены только основы, теперь же мы двинемся дальше и разберемся с дополнительными особенностями плавающего поведения. Как будет показано в следующих примерах, в режим плавания можно перевести любой HTML-элемент — как строчный, так и блочный.

Плавающий строчный текстовый элемент

В предыдущем примере (см. рис. 15.2 и 15.3) в режим плавания мы отправили встроенный элемент изображения. Посмотрим, что произойдет в результате выбора

режима плавания для строчного текстового (незаменяемого) элемента — в нашем случае элемента `span` (рис. 15.4).

Разметка

```
<p><span class="tip">TIP: Make sure that your packing tub or  
bucket  
has a hole below the top of the mold so the water will drain  
off.</span>After the cream is frozen rather stiff, prepare a  
tub or  
bucket of...</p>
```

Правило стиля

```
span.tip {  
    float: right;  
    margin: 1em;  
    width: 200px;  
    color: #ffff;  
    background-color: lightseagreen;  
    padding: 1em;  
}
```

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.

TIP: Make sure that
your packing tub or
bucket has a hole below
the top of the mold so
the water will drain off.

Рис. 15.4. Плавание строчного (незаменяемого) элемента

На первый взгляд, поведение плавающего текстового элемента такое же, как поведение плавающего изображения, что и ожидалось. Однако имеют место и некоторые тонкие моменты, на которые следует обратить внимание:

- всегда указывайте ширину для плавающих текстовых элементов — прежде всего обратите внимание, что правило стиля, которое переводит в плавающий режим элемент `span`, включает свойство `width`. Ширину плавающего текстового элемента необходимо указывать обязательно, поскольку без этого (со значением `auto`) размер по ширине из-за объема контента может оказаться чрезвычайно большим. Для коротких текстов, которые легко умещаются в предполагаемое для них место, это не представляет проблему. Однако более длинный текст, помещаемый в обтекаемый блок, расширит этот блок настолько, что вызвать обтекание текста вокруг этого блока окажется невозможным. Изображения же имеют собственную ширину, поэтому ширину изображений можно не указывать (как сделано в предыдущем примере), хотя, конечно, вполне возможно и указать;

НЕ ПРЕНЕБРЕГАЙТЕ УКАЗАНИЕМ ШИРИНЫ
ШИРИНУ ДЛЯ ПЛАВАЮЩИХ ТЕКСТОВЫХ ЭЛЕМЕНТОВ НЕОБХОДИМО УКАЗЫВАТЬ ОБЯЗАТЕЛЬНО.

- *плавающие встроенные элементы ведут себя как блочные* — обратите внимание, что поле сохраняется для всех четырех сторон плавающего текстового элемента `span`, хотя верхние и нижние поля для строчных элементов обычно не отображаются (см. рис. 14.18 из предыдущей главы). Дело в том, что все плавающие элементы ведут себя как блочные. Как только строчный элемент переходит в плавающий режим, он следует правилам отображения для элементов уровня блока, и поля отображаются со всех четырех сторон этого элемента;
- *поля плавающих элементов не коллапсируют* — в нормальном потоке соприкасающиеся верхние и нижние поля элементов коллапсируют (перекрываются), но поля плавающих элементов сохраняются со всех сторон в первоначальном виде.

Плавающие блочные элементы

Рассмотрим, что происходит при задании блоку, находящемуся в нормальном потоке, свойства `float`. В следующем примере весь второй элемент абзаца «плывет» влево (рис. ЦВ-15.5):

Разметка

```
<p>If you wish to pack ice cream...</p>
<p id="float">After the ice cream is rather stiff,...</p>
<p>Make sure that your packing tub or bucket...</p>
<p>As cold water is warmer than the ordinary...</p>
```

Правило стиля

```
p {
    border: 2px red solid;
}

#float {
    float: left;
    width: 300px;
    margin: 1em;
    background: white;
}
```

На рис. ЦВ-15.5, *вверху* вокруг всех элементов `p` я добавила красную границу, чтобы их местоположение было обозначено четче. Кроме того, фону плавающего абзаца присвоен белый цвет и со всех сторон этого абзаца добавлены поля размером в `1em` (отмечены синей пунктирной линией). На рис. ЦВ-15.5, *внизу* показано, какой вид имело бы отображение этого примера на реальной странице с отключенными уточняющими компонентами.

Как показано на рис. ЦВ-15.5, выбранный абзац смещается в сторону (на этот раз влево), и последующий контент обтекает его, хотя в обычной ситуации блоки бы просто следовали друг за другом. В этом примере желательно отметить несколько моментов:

- *необходимо задать для плавающих элементов блока значение ширины* — если вы не зададите значение ширины (`width`) плавающего блока, для него будет установлено значение `auto`, при котором его содержимое заполняет доступную

ширину окна браузера или другого содержащего его элемента. Нет никакого смысла в том, чтобы получить плавающий блок во всю ширину окна, потому что по задумке нам надо, чтобы другие абзацы обтекали плавающий блок, а не просто располагались ниже его;

- элементы не «плавают» выше места, в котором расположено указание на них в разметке — плавающий блок сдвигается влево или вправо относительно места его исходного расположения, что и позволяет вызвать его обтекание следующими элементами в потоке. Он остается ниже любых элементов блока, предшествующих ему в потоке (по сути, он ими «блокируется»). Это значит, что нельзя вызвать «всплытие» элемента к верхнему углу страницы, даже если ближайший предок этого элемента является элементом `body`. Если необходимо, чтобы плавающий элемент начинался в верхней части страницы, он должен размещаться первым в исходном коде документа;
- неплавающие элементы удерживаются в нормальном потоке — красные границы на рис. ЦВ-15.5, вверху показывают, что блоки элементов, соседствующие с плавающим абзацем, занимают полную ширину нормального потока, за исключением той части их контента, которая этот плавающий абзац обтекает.

Например, добавление левого поля к окружающим «поплавок» абзацам увеличит пространство на левом краю страницы, а не между окружающим текстом и «поплавком». Если необходимо наличие пространства между плавающим элементом и обтекающим его текстом, следует применять поле к самому плавающему элементу.

Эта удобная модель оформления текстов заслуживает, чтобы на нее обратили особое внимание.

АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ

Абсолютное позиционирование — это метод CSS, с помощью которого можно размещать элементы на странице независимо от их изначального местоположения. Мы рассмотрим абсолютное позиционирование в этой главе позже. Порядок отображения элементов также можно изменить, задействовав инструменты Flexbox и Grid, о чем рассказано в главе 16.

«Очистка» плавающих элементов

Если вы собираетесь применять плавающие элементы, важно знать, каким образом можно отключать обтекание текстом и возвращаться к обычному потоку. Это выполняется путем очистки элемента, следующего за плавающим. Применение свойства `clear` к такому элементу предотвращает его размещение сразу за плавающим и вынуждает его разместиться на следующем за плавающим элементом доступном «чистом» пространстве:

`clear`

- Значения: `left` | `right` | `both` | `none`.
- По умолчанию — `none`.
- Применение — только к элементам блочного уровня.
- Наследование — нет.

Учтите, что свойство `clear` применяется к элементу, который необходимо поместить ниже плавающего элемента, а не к самому плавающему элементу. Значение `left` помещает элемент ниже любых элементов, которые были отправлены «плавать» влево. Точно так же значение `right` позволяет «очистить» элементы, следующие за элементом, плавающим на правом краю содержащего его блока. Если имеется несколько плавающих элементов и нужно убедиться в том, что элемент будет находиться ниже всех других плавающих элементов применяйте значение `both` для очистки «поплавков» с обеих сторон.

В следующем примере свойство `clear` применяется с тем, чтобы элементы `h2` находились ниже элемента, плавающего слева. На рис. 15.6 показано, что заголовок `h2` начинается со следующего доступного места, находящегося под «поплавком»:

```
img {
    float: left;
    margin-right: .5em;
}
h2 {
    clear: left;
    margin-top: 2em;
}
```

If pure raw cream is stirred rapidly, it swells and becomes frothy, like the beaten whites of eggs, and is "whipped cream." To prevent this in making Philadelphia Ice Cream, one-half the cream is scalded, and when it is very cold, the remaining half of raw cream is added. This gives the smooth, light and rich consistency which makes these creams so different from others.

USE OF FRUITS

Use fresh fruits in the summer and the best canned unsweetened fruits in the winter. If sweetened fruits must be used, cut down the given quantity of sugar. Where acid fruits are used, they should be added to the cream after it is partly frozen.

The time for freezing varies according to the quality of cream or milk or water; water ices require a longer time than ice creams. It is not well to freeze the mixtures too rapidly; they are apt to be coarse, not smooth, and if they are churned before the mixture is icy cold they will be greasy or "buttery."

Рис. 15.6. Очистка элемента, следующего за плавающим слева

Обратите внимание, что здесь к элементу `h2` применяется верхнее поле размером 2`em`, которое, тем не менее, не отображается на рис. 15.6 между плавающим изображением и заголовком. Это результат коллапса в потоке вертикальных полей. Если необходимо, чтобы между «поплавком» и последующим текстом имелся пробел, примените нижнее поле к самому плавающему элементу.

Итак, вы уже познакомились с относящимися к плавающим элементам понятиями, которых будет достаточно для выполнения *упражнения 15.1*.

УПРАЖНЕНИЕ 15.1. ПЛАВАЮЩИЕ ИЗОБРАЖЕНИЯ

В процессе выполнения упражнений этой главы мы внесем дополнительные улучшения в домашнюю страницу «Black Goose Bakery», с которой мы работали в *главе 14*. Если вы не следовали инструкциям из той главы, имейте в виду, что копия документа в самом последнем состоянии (файл `bakery_ch15.html`) находится в материалах главы 15 по адресу: learningwebdesign.com/5e/materials.

1. Откройте CSS-файл в текстовом редакторе и HTML-документ в браузере. Мы начнем здесь с удаления незаполненного вертикального пространства около изображений выпечки, переместив в плавающем режиме эти изображения влево. Для этого с помощью контекстного селектора создадим новое правило стиля, предназначенное для работы именно с изображениями, находящимися в разделе `main`:

```
main img {
    float: left;
}
```

Сохраните CSS-файл, обновите страницу в браузере, и вы увидите, что после от-правки изображений в плавание появилась необходимость в кое-какой уборке.

2. Мне бы хотелось, чтобы ссылки «Learn more» всегда отображались под изо-бражениями, причем были бы четко видны в левой части страницы. К счастью, абзацы с этими ссылками помечены классом `more`, и для них имеется правило стиля с использованием селектора класса. Так что просто очистите эти абзацы от любых «поплавков» с левого края:

```
p.more {
    ...
    clear: left;
}
```

3. Наконец, отрегулируем интервалы вокруг плавающих изображений. Присвойте для обоих изображений поле, равное `1em` с правой стороны и внизу, применяя сокращенное свойство `margin`:

```
main img {
    float: left;
    margin: 0 1em 1em 0;
}
```

4. Мне кажется, что изображению маффина надо добавить дополнительное пространство слева, чтобы лучше выровнять его с изображением хлеба. Воспользуйтесь для этого от-личным селектором атрибу-тов, пригодным для захвата любого изображения, атри-бут `src` которого включает слово `muffin` (здесь он только один):

```
img[src*="muffin"] {
    margin-left: 50px;
}
```

На рис. 15.7 показан новый и улучшенный раздел «Fresh from the Oven» нашей страницы



Рис. 15.7. Раздел продукции с плавающими изображе-ниями и обтекающим текстом теперь выглядит более компактно

Несколько плавающих элементов

Размещать на странице или даже внутри одного элемента несколько плавающих элементов оказалось весьма удобно. Фактически в течение многих лет плавающие элементы были основным методом выравнивания навигационных меню и даже создания целых макетов страниц (пожалуйста, найдите время, чтобы прочитать *врезку «Макеты на основе плавающих элементов»*).

В режиме плавания нескольких элементов действует сложная система закулисных правил отображения, которая гарантирует, что плавающие элементы не перекрываются. Вы можете обратиться к CSS-спецификации для получения подробной информации, но, если вкратце, то эту систему можно свести к правилу, гласящему, что плавающие элементы размещаются как можно левее или правее (в зависимости от того, как задано) и настолько высоко, насколько позволяет свободное пространство.

В следующем примере (рис. 15.8) показано, что произойдет, если ряд последовательных абзацев будут «отправлены в плавание» к одной из сторон:

Разметка

```
<p>[PARAGRAPH 1] ONCE upon a time...</p>
<p class="float">[P2]...</p>
```

МАКЕТЫ НА ОСНОВЕ ПЛАВАЮЩИХ ЭЛЕМЕНТОВ

Интересно, что в CSS1 или CSS2 не было предусмотрено инструментов для создания правильных макетов страниц. Однако некоторые дизайнеры догадались, что для выравнивания элементов по горизонтали можно применять особенности поведения плавающих CSS-элементов, и «поплавки» стали использоваться для превращения списков в панели навигации, а также для преобразования целых разделов документа в макеты с колонками.

Макеты с плавающими элементами преобладают в Интернете и сейчас, когда готовится эта книга, но, с той поры как прекрасные инструменты CSS-разметки: Flexbox и Grid — начали получать поддержку со стороны браузеров, популярность макетов с плавающими элементами постепенно сходит на нет. Такие макеты, в конечном итоге, исчезнут, как и макеты на основе таблиц, имевшие хождение в 1990-х годах.

Тем не менее мы сейчас находимся в переходном периоде. Не все находящиеся сегодня в эксплуатации браузеры поддерживают новые стандарты, такие, как Flexbox и Grid, поэтому, в зависимости от браузеров, которые необходимо поддерживать, все же может потребоваться предоставляемый повсеместно запасной вариант дизайна. И этот вариант обеспечивают плавающие элементы.

Если вам требуется поддержка со стороны старых браузеров, которые не понимают Flexbox и Grid, вы можете ознакомиться с моей статьей «Page Layout with Floats and Positioning» (PDF), которая доступна по адресу: learningwebdesign.com/articles/. В этой статье вы найдете уроки по созданию панели навигации с плавающими элементами, а также ряд шаблонов для создания многоколоночных макетов с плавающими элементами и позиционированием. Возможно, вам и не понадобятся эти методы, но в случае необходимости эта статья будет вам весьма полезна.

```
<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>
```

Правило стиля

```
p.float {
    float: left;
    width: 200px;
    margin: 0px;
    background: #F2F5d5;
    color: #DAEAB1;
}
```

Элементы, плавающие в направлении левого края.

Если недостаточно места, последующие элементы перемещаются вниз и как можно ближе к левому краю.

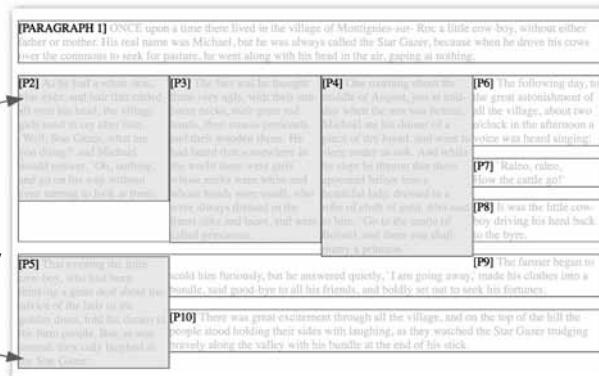


Рис. 15.8. Несколько плавающих элементов выстраиваются в ряд и не перекрываются

Первые три «поплавка» прижимаются к левому краю, но, если для четвертого (P5) не хватает места, он смещается вниз и влево, пока не столкнется с каким-либо объектом — в нашем случае с краем окна браузера. Однако, если бы один из поплавков — например (P2), оказался бы очень длинным, «поплавок» (P5) вместо этого прижался бы к краю длинного «поплавка». Обратите внимание, что следующий абзац в нормальном потоке (P6) начинает обтекание в самой высокой точке, которая ему достижима, то есть чуть ниже (P1).

Удерживание плавающих элементов

В некоторых ситуациях плавающие элементы могут вести себя странно. По умолчанию они предназначены «выпирать» из элемента, который их содержит. Это удобно, когда надо, чтобы текст просто обтекал плавающее изображение, но иногда такого поведения недостаточно.

Посмотрите на представленный на рис. ЦВ-15.9 пример. Нам желательно, чтобы граница растягивалась вокруг всего контента, однако здесь плавающее изображение «свисает» вниз.

А если перевести в плавающий режим *все* элементы содержащего их элемента, в потоке не останется элементов для удержания этого элемента открытым (рис. 15.10). Как можно видеть, элемент `#container div` содержит здесь два абзаца. В варианте нормального потока (*вверху*) у содержимого элемента `#container` имеются фоновый цвет и граница, его окружающая:

```
<div id="container">
  <p>...</p>
  <p>...</p>
</div>
#container {
  background: #f2f5d5;
  border: 2px dashed green;
}
```

Однако, если плавающими сделать оба абзаца (то есть весь контент внутри элемента `div`), как показано на рис. 15.10, *внизу*, блок элемента `#container` схлопывается до нулевой высоты, оставляя «поплавки» просто висеть под ним (вы же видите над ними пустую границу):

```
p {
  float: left;
  width: 44%;
  padding: 2%;
}
```

Так получается, потому что в нормальном потоке теперь отсутствует контент для задания высоты содержащему его элементу `div`. Это не тот эффект, который нам нужен.

К счастью, имеется несколько решений этой проблемы, причем они достаточно просты. Наиболее популярным и надежным вариантом *удерживания* плавающих элементом является метод «clearfix». Он использует псевдоэлемент `:after` для вставки после контейнера символьного пространства, устанавливая его отображение в режиме «block» и очищая его с обеих сторон. Для получения дополнительной информации по этой версии метода «clearfix» ознакомьтесь со статьей Тьери Кобленца (Thierry Koblentz) «The very latest clearfix reloaded» (cssmojo.com/the-very-latest-clearfix-reloaded). В следующем примере этот метод применяется к элементу `#container` `div`, показанному на рис. 15.10:

```
#container:after {
  content: " ";
  display: block;
  clear: both;
  background-color: #f2f5d5; /*light green*/
  border: 2px dashed green;
  padding: 1em;
}
```

Другая возможность состоит в переводе в плавающий режим самого элемента, содержащего «поплавок», при значении его ширины, равной 100%:

```
#container {
    float: left;
    width: 100%;

    ...
}
```

На рис. 15.11 показан результат применения к предыдущим примерам методов удерживания. Любой из них делает свое дело.

Итак, с основами плавания объектов я вас здесь познакомила. Но, если вы полагаете, что прямоугольное обтекание текста несколько однообразно, можно добавить разнообразия (или же просто устраниТЬ лишние пустоты), применив CSS-формы.

В нормальном потоке контейнер `div` содержит абзацы.

Etiā 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.

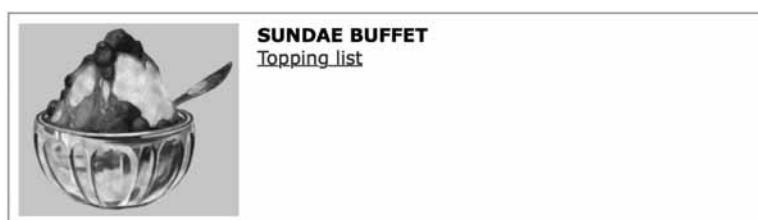
Cras id ipsum dui. Donec semper congue lectus quis vulputate. Ut felis leo, bibendum at blandit non, luctus ac lorem. Nunc vitae ligula ut neque convallis sagittis. Quisque consequat orci sed arcu tincidunt et volutpat tellus tempor. Nulla vulputate ante nec felis elementum auctor. Duis magna neque, 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 duī 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 fermentum ante, quis tempus est fringilla eu.

Если оба абзаца плавающие, контейнер вокруг них не растягивается.

Etiā 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 duī 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 fermentum ante, quis tempus est fringilla eu.

Рис. 15.10. Блок элемента, содержащего плавающий контент, полностью исчезает, если весь его контент становится плавающим



Etiā 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 duī 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 fermentum ante, quis tempus est fringilla eu.

Рис. 15.11. Наши «повисшие» плавающие теперь удерживаются

БУДУЩЕЕ, ОЖИДАЮЩЕЕ МЕТОД CLEARFIX

Новое отображаемое значение: `flow-root` — может сделать метод clearfix устаревшим раз и навсегда. Присваивание свойству `display` элемента, содержащего плавающие элементы, значения `flow-root` приводит к автоматическому растягиванию этого элемента, позволяя удерживать содержащиеся в нем плавающие изображения. На момент подготовки книги эта возможность все еще находится в фазе эксперимента, но за неё стоит следить. Потенциальный ее недостаток в том, что при ее использовании отключается коллапс полей между элементом и его первым/последним дочерним элементом, что может привести к непредсказуемым результатам. Дополнительно о методе `flow-root` можно прочитать в посте Рэйчел Эндрю (Rachel Andrew) «The end of theclearfix hack?» (rachelandrew.co.uk/archives/2017/01/24/the-end-of-theclearfix-hack).

СВОБОДНОЕ ОБТЕКАНИЕ ТЕКСТОМ CSS-ФОРМ

Рассматривая предыдущие примеры с плавающими элементами, можно заметить, что текст, обтекающий плавающее изображение или блок элемента, всегда имеет прямоугольную форму. Однако, применяя свойство `shape-outside`, можно изменять форму обтекающего текста на круглую, эллипсовидную, многоугольную или любую другую. Эта возможность, предоставляемая нам CSS, весьма перспективна, поэтому обязательно ознакомьтесь с информацией о поддержке ее браузерами.

А далее следует краткое введение в CSS-формы, которое наверняка вдохновит вас на их использование и подготовит к их самостоятельному изучению в будущем:

`shape-outside`

- **Значения:** `none | circle() | ellipse() | polygon() | url() | [margin-box | padding-box | content-box]`.
- **По умолчанию** — `none`.
- **Применение** — к плавающим элементам.
- **Наследование** — нет.

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

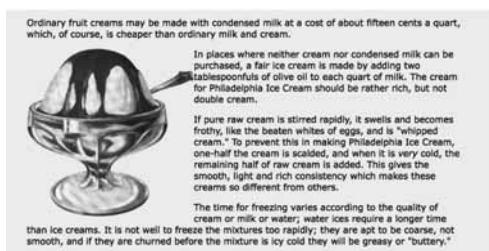
По состоянию на 2018-й год формы обтекания текстом поддерживались только в *Chrome 37+*, *Opera 24+*, *Safari 7.1+* (с префиксом, без запуска в 10.1), *iOS Safari 8+* (с префиксом, без запуска в 10.3+) и *Android 5.6+*. Эта функция рассматривается применительно к *Microsoft Edge* и внедряется в *Firefox*, поэтому к моменту издания книги ситуация с поддержкой может улучшиться. Обратитесь к сайту CanIUse.com для уточнения текущего состояния поддержки.

Тем не менее уже сейчас смело используйте это свойство в качестве постепенного улучшения для тех своих проектов, в которых прямоугольное обтекание текстом еще вполне приемлемо. Другой альтернативой служит применение функционального запроса (`@supports`) для поддержки резервного набора стилей применительно к браузерам, не поддерживающим эту возможность (функциональные запросы рассматриваются в главе 19).

На рис. 15.12 показано заданное по умолчанию обтекание текстом вокруг плавающего изображения (*слева*) и такое же обтекание с применением свойства `shape-outside` (*справа*). Именно этот эффект используется в печатных журналах, а теперь эта возможность доступна и в Интернете!

Заметим, что изменять можно форму обтекания текстом любого плавающего элемента, но здесь мы остановимся именно на изображениях, поскольку плавающие текстовые элементы, как правило, представляют собой блоки по умолчанию и так хорошо вписывающиеся в прямоугольную форму.

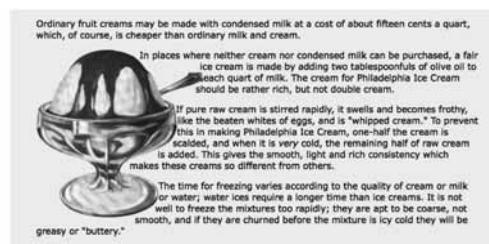
Организовать обтекание текстом изображения можно с использованием двух различных способов. Согласно одному из них, можно предоставить форму обтекания в виде плоской фигуры с помощью значений: `circle()`, `ellipse()` или `polygon()`. Другой способ состоит в применении метода `url()`, указывающего изображение с прозрачными областями (например, GIF или PNG). При использовании этого метода обтекающий текст «заплывает» в прозрачные области изображения и останавливается, достигнув его непрозрачных областей. Это метод формы показан на рис. 15.12 и описывается первым.



Обтекание изображения текстом, заданное по умолчанию

ОГРАНИЧЕНИЕ СВОЙСТВА `SHAPE-OUTSIDE`

Свойство `shape-outside` пока работает только с плавающими элементами, но считается, что подобное положение вещей изменится в будущем.



Обтекание текстом изображения с помощью свойства `shape-outside`, использующее в качестве ориентира прозрачные области изображения

Рис. 15.12. Примеры обтекания текстом изображения с помощью свойства `shape-outside`

Использование прозрачности изображений

В примере, показанном на рис. 15.12, я поместила изображение `sundae.png` в HTML-документ для отображения его на странице и это же изображение указала в правиле стиля с помощью метода `url()`, чтобы форму обтекания определяли его прозрачные области. В общем случае имеет смысл использовать одно и то же изображение и в документе, и в CSS-форме, но это не обязательно. Можно применять полученную для одного изображения форму обтекания к другому изображению на странице:

Разметка

```
<p> In places...</p>
```

Правило стиля

```
img.wrap {
    float: left;
    width: 300px;
    height: 300px;
    -webkit-shape-outside: url(sundae.png); /* prefix required
in 2018 */
    shape-outside: url(sundae.png);
```

Как можно видеть на рис. 15.12, обтекающий текст практически наезжает на изображение. Может быть, следует добавить между изображением и текстом немного свободного пространства, воспользовавшись свойством `shape-margin`?

`shape-margin`

- **Значения:** длина | процентное соотношение
- **По умолчанию** — 0.
- **Применение** — к плавающим элементам.
- **Наследование** — нет.

ПОРОГ НЕПРОЗРАЧНОСТИ

Если исходное изображение имеет несколько уровней прозрачности — например, градиентную тень, свойство `shape-image-threshold` позволяет тексту проникать в прозрачные области изображения, но прекращает это по достижении определенного уровня прозрачности. Значение этого свойства — число от 0 до 1 — соответствует проценту прозрачности. Например, если вы установите порог равным .2, текст будет обтекать в областях, прозрачность которых меньше и равна 20%, причем обтекание прекратится для областей с большими уровнями непрозрачности.

Свойство `shape-margin` определяет размер пространства между формой и обтекающим текстом. На рис. 15.13 можно увидеть эффект добавления области `1em` между непрозрачными областями изображения и обтекающими текстовыми строками, что похоже на использование поля соответствующего размера.

```
-webkit-shape-margin: 1em;
shape-margin: 1em;
```

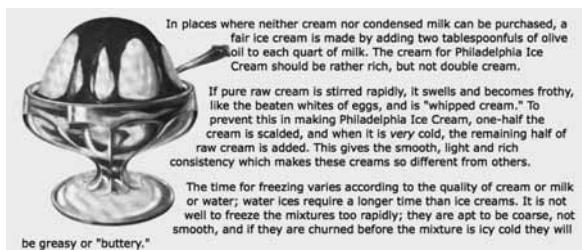


Рис. 15.13. Добавление поля между формой и обтекающим текстом

ПАРАМЕТР БЕЗОПАСНОСТИ CHROME И OPERA

В браузерах Chrome и Opera имеется параметр безопасности, который усложняет организацию обтекания текстом изображений. Не обращая внимания на установки системного администратора

стратора, браузер ограничивает применение используемого для создания CSS-формы изображения, если его файл не находится в том же домене, что и файл, инициирующий его использование. Это не является ошибкой — подобные ограничения вытекают из спецификации.

Правило также подразумевает, что совместимые браузеры не разрешают использовать изображения для CSS-формы, если файлы используются локально (то есть на вашем компьютере), — они должны загружаться для работы на сервер, что усложняет процесс дизайна, особенно для начинающих.

Если вы задаете обтекание текстом изображений и, зная, что CSS-таблица создана корректно, не видите обтекания в окне браузера, скорее всего, виновником является этот параметр безопасности, связанный с распределением ресурсов между источниками (*Cross-Origin Resource Sharing, CORS*).

Использование плоской фигуры

Другим методом задания формы обтекания текстом является применение одного из значений: `circle()`, `ellipse()` и `polygon()` — для создания соответствующей плоской фигуры.

Значение `circle()` создает для обтекающего текста форму круга. Указанная в скобках величина представляет длину радиуса круга:

```
circle(радиус)
```

В следующем примере радиус круга составляет 150 пикселов — половина ширины изображения, равного 300 пикселям. По умолчанию круг центрируется на «поплавке» по вертикали и по горизонтали:

```
img.round {  
    float: left;  
    -webkit-shape-outside: circle(150px);  
    shape-outside: circle(150px);  
}
```

На рис. ЦВ-15.14 показано применение этого правила стиля к изображениям. Обратите внимание на то, что уровень прозрачности изображения здесь не учитывается. Форма представляет собой просто наложенную на изображение фигуру, устанавливающую границы для обтекающего его текста. Таким способом любая фигура может применяться к любому изображению или другому плавающему элементу.

Самое время продемонстрировать опасное поведение форм обтекания — они позволяют тексту «заезжать» во внутрь плавающего изображения или элемента, но не образуют свободного пространства за его пределами. Так, в примере, показанном на рис. 15.15, я увеличила радиус круга со 150 до 200 пикселов:

```
img.round {  
    float: left;  
    -webkit-shape-outside: circle(200px);  
    shape-outside: circle(200px);  
}
```

Обратите внимание, что текст при этом выравнивается вдоль правого края изображения, даже если радиус круга установлен на 50 пикселов больше половины его ширины, то есть выходит за его края, и, тем не менее, круг не отталкивает текст от «поплавка». Так что, если вам нужно держать обтекающий текст подальше от внешнего края плавающего изображения или элемента, примените поле к самому элементу (конечно, тогда это будет стандартная прямоугольная форма).

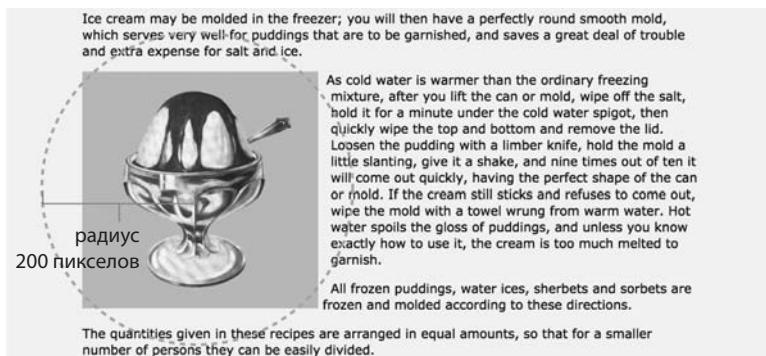


Рис. 15.15. CSS-формы позволяют тексту «заезжать» в плавающие элементы, но не образуют свободного места за его пределами

ОПАСНОЕ ПОВЕДЕНИЕ ФОРМ ОБТЕКАНИЯ

CSS-формы позволяют тексту «заезжать» в плавающие элементы, но не отодвигают текст от них в сторону.

Эллиптические формы обтекания создаются с помощью значения `ellipse()`, которое поддерживает задание горизонтального и вертикального радиусов эллипса, за которыми следует слово `at`, а после него — координаты `x` и `y` центра фигуры:

```
ellipse(rx ry at x y);
```

Координаты центра фигуры могут указываться в единицах длины или в процентах. В нашем случае эллипс имеет горизонтальный радиус, равный 100 пикселам, и вертикальный радиус, равный 150 пикселам, с центром в плавающем элементе, к которому он и применяется (рис. 15.16):



Края изображения представлены синим квадратом, а эллипс — оранжевым пунктиром

Рис. 15.16. Текст обтекает изображение по эллипсу, созданному с помощью ключевого слова `ellipse()`

```
img.round {  
    float: left;  
    -webkit-shape-outside: ellipse(200px 100px at 50% 50%);  
    shape-outside: ellipse(200px 100px at 50% 50%);  
}
```

Наконец, перейдем к значению `polygon()`, позволяющему формировать собственную фигуру обтекания за счет задания последовательности разделенных запятыми координат *x* и *y* (рис. ЦВ-15.17):

```
img.wrap {  
    float: left;  
    width: 300px;  
    height: 300px;  
    shape-outside: polygon(0px 0px, 186px 0px, 225px 34px, 300px  
34px,  
300px 66px, 255px 88px, 267px 127px, 246px 178px, 192px 211px,  
226px  
236px, 226px 273px, 209px 300px, 0px 300px);  
}
```

Ну и ну! Так много цифр, а созданная фигура весьма проста. Я хотела бы показать вам какой-нибудь подходящий инструмент для рисования и экспорта многоугольных контуров, но, к сожалению, на момент подготовки этой книги мне рекомендовать пока нечего... Для приведенного здесь примера многоугольника я получила его координаты, открыв изображение в Photoshop и отслеживая эти координаты вручную, что вполне возможно, но хлопотно.

CSS SHAPES EDITOR

В будущую версию Firefox, которая, вероятно, станет доступна в ближайшее время, должен быть включен Редактор форм CSS — *CSS Shapes Editor* (developer.mozilla.org/en-US/docs/Tools/Page_Inspector/How_to/Edit_CSS_shapes).

Ресурсы, посвященные CSS-формам

Есть еще ряд тонкостей, касающихся CSS-форм, которые я оставляю вам для дальнейшего изучения. Вот несколько ресурсов, с которых можно начать:

- CSS Shapes Module, Level 1 (www.w3.org/TR/css-shapes-1/);
- Статья Развана Калимана (Razvan Caliman) «Getting Started with CSS Shapes» (www.html5rocks.com/en/tutorials/shapes/getting-started/);

ПОИСК ПО КЛЮЧЕВЫМ СЛОВАМ

CSS SHAPES

Выполняя в Интернете поиск по ключевым словам CSS Shapes (CSS-формы), вы наверняка встретите этот же термин, применяемый в методике, которая использует CSS для рисования геометрических фигур — таких, как треугольники, стрелки, круги и так далее. Это немного сбивает с толку, хотя эти другие «CSS-формы» весьма изящны, и с ними вы, возможно, захотите повозиться. Речь о них пойдет в главе 23.

- Статья Джен Симмонс (Jen Simmons) «CSS Shapes at the Experimental Layout Lab (labs.jensimmons.com/#shapes);
- Статья Эрика Мейера (Eric Meyer) «A Redesign with CSS Shapes» (alistapart.com/article/redesign-with-css-shapes).

А теперь почему бы нам не добиться, чтобы текст обтекал изображения на странице «Black Goose Bakery» более интересным способом — хотя бы у пользователей, располагающими браузерами с соответствующей поддержкой (упражнение 15.2)?

УПРАЖНЕНИЕ 15.2. ДОБАВЛЕНИЕ ФОРМ ВОКРУГ ПЛАВАЮЩИХ ИЗОБРАЖЕНИЙ

Имеющиеся на странице «Black Goose Bakery» изображения хлеба и маффинов предоставляют нам хорошую возможность опробовать CSS-формы. Вам только необходимо поддерживать эти свойства браузер — такой, как последние версии Chrome, Safari или Opera, чтобы отобразить созданный нами эффект обтекания.

Откройте последнюю версию таблицы стилей страницы нашей пекарни и найдите раздел с пометкой `/* main "products" styles */`. Мы поместим в него стили обтекания изображений, создающие для них таблицу стилей.

Выберите в качестве цели каждое отдельное изображение, используя селектор атрибутов (уже есть один, настроенный для «muffin»). Начните с простого случая, применив обтекание текстом круга. Установите радиус круга равным 125 пикселов для изображения хлеба (`bread`) и 110 пикселов для маффинов (`muffin`):

```
img[src*="bread"] {
    -webkit-shape-outside: circle(125px);
    shape-outside: circle(125px);

}
img[src*="muffin"] {
    margin-left: 50px;
    -webkit-shape-outside: circle(110px);
    shape-outside: circle(110px);
}
```

Сохраните стили и просмотрите страницу в поддерживающем CSS-формы браузере. Круги выглядят вполне прилично, но мне кажется, что обтекание вокруг изображения хлеба улучшится, если применить эллипс. Добавьте необходимые изменения после объявлений круга, и обтекание в виде эллипса переопределит предыдущие стили (либо удалит и заменит их):

```
img[src*="bread"] {
    -webkit-shape-outside: ellipse(130px 95px at 50% 50%);
    shape-outside: ellipse(130px 95px at 50% 50%);
}
```

Если чувствуете уверенность, можете вокруг изображения маффина (`muffin`) вместо круга построить форму обтекания в виде многоугольника. Для этого вам придется воспользоваться приведенными здесь координатами или просто скопировать и вставить координаты из представленного в материалах для этой главы выполненного упражнения. Или просто оставьте круг, если вас это устраивает:

```
img[src*="muffin"] {  
...  
shape-outside: polygon(0px 0px, 197px 0px, 241px  
31px, 241px 68px, 226px 82px, 219px 131px, 250px  
142px, 250px 158px, 0px 158px);  
}
```

Окончательный результат показан на рис. 15.18. Произведенные изменения будут наиболее заметны, если окно браузера сделать настолько узким, чтобы обтекание текста продемонстрировало примененные формы. Для браузеров, которые не поддерживают формы, прямоугольная область — просто замечательное решение.



Рис. 15.18. Страница пекарни с текстом, обтекающим изображения в форме эллипса (хлеб) и в форме многоугольника (маффин) с использованием CSS-форм

Что ж, о плавающих элементах достаточно! Вы узнали, как отправить элементы плавать влево или вправо, как выполняется очистка последующих элементов, чтобы они начинались под плавающими элементами, и даже каким образом можно создавать причудливые формы для обтекания текстом. Теперь же мы перейдем к рассмотрению другого подхода к перемещению элементов на странице — позиционированию.

ОСНОВЫ ПОЗИЦИОНИРОВАНИЯ

ОБЛАСТЬ ПРОСМОТРА

При рассмотрении темы позиционирования я буду придерживаться формального термина «область просмотра», но вы имейте в виду, что это может быть окно браузера настольного компьютера, полный экран мобильного устройства или рамка элемента `iframe` применительно к веб-странице, загруженной в этот фрейм. То есть это любое пространство, которое визуально отображает веб-страницу.

В CSS поддерживается несколько методов позиционирования элементов на странице. Элементы могут позиционироваться относительно того места, где они обычно появляются в потоке, или вообще могут быть удалены из потока и размещены в определенном месте на странице. Вы также можете расположить элемент относительно области просмотра, и он останется на месте при прокрутке остальной части страницы.

Типы позиционирования

`position`

- **Значения:** `static` | `relative` | `absolute` | `fixed`.
- **По умолчанию** — `static`.
- **Применение** — ко всем элементам.
- **Наследование** — нет.

Свойство `position` указывает, что элемент должен быть позиционирован, а также определяет метод позиционирования. Кратко опишем сейчас каждое значение этого свойства, а затем рассмотрим их более подробно:

- `static` — это нормальная (статическая) схема позиционирования, при реализации которой элементы располагаются в том месте, в котором они появляются в нормальном потоке документа;
- `relative` — *относительное позиционирование* перемещает блок элемента относительно его исходного положения в потоке. Особенность относительного позиционирования состоит в том, что пространство, которое элемент занимал в нормальном потоке, сохраняется пустым;
- `absolute` — *абсолютно позиционированные* элементы полностью удаляются из потока документа и позиционируются относительно области просмотра или содержащего их элемента (мы поговорим об этом позже). В отличие от относительно расположенных элементов, пространство, которое они занимали, закрывается. Абсолютно позиционированные элементы никак не влияют на расположение окружающих их элементов;
- `fixed` — отличительная особенность *фиксированного позиционирования* заключается в том, что элемент остается в одной позиции окна просмотра даже при прокрутке документа. Фиксированные элементы удаляются из потока документов и позиционируются относительно окна просмотра, а не какого-либо другого элемента в документе;

- `sticky` — *липкое (прикрепленное) позиционирование* представляет собой комбинацию относительного и фиксированного в том смысле, что элемент ведет себя так, как если бы он был позиционирован относительно, до тех пор пока он не будет прокручен в указанное положение относительно области просмотра, и с этого момента он останется фиксированным.

Сайт MDN Web Docs содержит описание возможного варианта его применения:

Липкое позиционирование обычно используется для заголовков, расположенных в алфавитном списке. Заголовок В появится после пунктов, начинающихся с буквы А, когда они будут прокручены за пределы экрана. Но, вместо того чтобы уйти за пределы экрана с остальным содержимым, заголовок В будет оставаться зафиксированным в верхней части области просмотра, до тех пор пока не будут прокручены за пределы экрана все элементы В, после чего он будет заменен заголовком С.

Значение позиции `sticky` поддерживается текущими версиями браузеров Chrome, Firefox, Opera, MS Edge, Android, а также Safari и iOS Safari с префиксом `-webkit-`. Ни одна из версий браузера IE не поддерживает эту возможность. К счастью, прикрепленное позиционирование ничему не мешает, и если не поддерживается браузером, то позиционированный таким образом элемент остается строчным и просто прокручивается вместе с документом.

Каждый метод позиционирования имеет свое назначение, но абсолютное позиционирование является наиболее универсальным. При абсолютном позиционировании объект можно размещать в любом месте окна просмотра или внутри другого элемента. Аbsolute позиционирование ранее применялось для создания многоколоночных макетов, но сейчас чаще используется для небольших задач, таких, как позиционирование поля поиска в верхнем углу области просмотра. Это удобный инструмент, если применять его осмотрительно и экономно.

Указание позиции

После выбора метода позиционирования фактическое положение элемента указывается с помощью комбинации, состоящей из четырех свойств *смещения*:

`top, right, bottom, left`

- **Значения:** `длина | процентное соотношение | auto.`
- **По умолчанию** — `auto`.
- **Применение** — к позиционируемым элементам (когда значением свойства `position` служат: `relative`, `absolute` или `fixed`).
- **Наследование** — нет.

Значение, предоставленное для каждого свойства смещения, определяет расстояние, на которое элемент удаляется от соответствующего края. Например, значение `top` определяет расстояние, на которое верхний внешний край позиционируемого элемента смещается от верхнего края браузера или другого содержащего его элемента. Положительное значение для `top` приводит к тому, что блок элемента смещается

вниз на эту величину. Аналогично положительное значение для `left` перемещает позиционированный элемент на эту величину вправо (к центру содержащего его блока).

ОТРИЦАТЕЛЬНЫЕ ЗНАЧЕНИЯ СМЕЩЕНИЯ

Отрицательные значения допустимы и перемещают элемент в направлении, противоположном положительному значению. Например, отрицательное значение для `top` приведет к перемещению элемента вверх.

Дальнейшие объяснения и примеры свойств смещения будут предоставляться при обсуждении каждого метода позиционирования. А начнем мы рассмотрение позиционирования с достаточно простого относительного метода `relative`.

ОТНОСИТЕЛЬНОЕ ПОЗИЦИОНИРОВАНИЕ

Как упоминалось ранее, относительное позиционирование перемещает элемент относительно его исходного места в потоке. Пространство, которое он занимал, сохраняется и продолжает влиять на макет окружающего контента. Это легко понять на простом примере.

СОХРАНЕНИЕ ЗАНИМАЕМОГО ПРОСТРАНСТВА

Когда элемент относительно позиционируется, пространство, которое он занимал до этого, сохраняется.

держащего его абзаца четко представляют его местоположение. Сначала применим свойство `position`, чтобы установить метод как `relative`, затем зададим свойство смещения `top` для перемещения элемента на `2em` вниз от его первоначального положения и свойство `left` для его перемещения на `3em` вправо. Помните, что значения свойств смещения перемещают элемент от указанного края, поэтому, если вы хотите, чтобы что-то сместилось вправо, как это сделано здесь, применяется именно свойство смещения `left`:

```
em {  
    position: relative;  
    top: 2em; /* сдвигает элемент вниз */  
    left: 3em; /* сдвигает элемент вправо */  
    background-color: fuchsia;  
}
```

А теперь подробнее рассмотрим, что здесь происходит:

- *исходное пространство элемента в потоке документа сохраняется.* Вы видите пустое место там, где выделенный текст находился до того, как был позиционирован. Окружающий контент располагается так, как будто элемент все еще там находится, и поэтому говорят, что элемент продолжает «влиять» на окружающий контент;
- *происходит перекрытие* — позиционированный таким образом элемент может перекрывать другие элементы, как это показано на рис. 15.19.

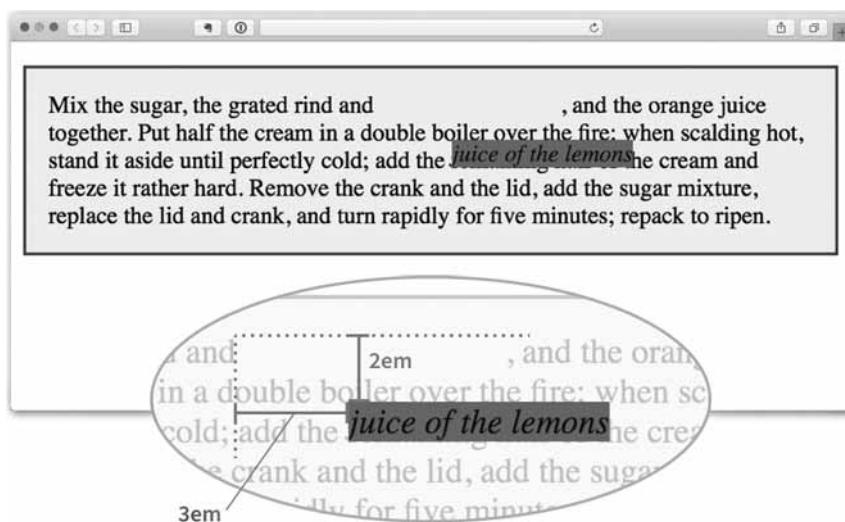


Рис. 15.19. Если элемент позиционируется с помощью метода `relative`, пространство, которое он занимал до этого, сохраняется

Пустое пространство, оставленное относительно позиционированными объектами, зачастую только мешает, из-за чего этот метод позиционирования применяется не столь часто, как абсолютное позиционирование. Тем не менее относительное позиционирование обычно применяется для формирования «контекста позиционирования» для абсолютно позиционированного элемента. Запомните термин *контекст позиционирования* — мы познакомимся с ним в следующем разделе.

АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ

Абсолютное позиционирование выполняется несколько иначе и является более гибким методом для точного размещения элементов на странице по сравнению с относительным позиционированием.

СТОЛБЦЫ ПРИ АБСОЛЮТНОМ ПОЗИЦИОНИРОВАНИИ

Как и плавающие элементы, абсолютное позиционирование может быть использовано для создания макетов из столбцов. В наши дни макеты со столбцами должны формироваться с помощью CSS Grid, но можно, как запасной вариант, применять и позиционированные столбцы, если речь идет об использовании устаревших браузеров, которые не поддерживают Grid.

Если необходимо узнать, каким образом использовать абсолютное позиционирование для полного макетирования страницы, обратитесь к инструкциям и шаблонам, которые содержатся в статье «Page Layout with Floats and Positioning» (PDF-файл), доступной по адресу: learningwebdesign.com/articles/.

Познакомившись с тем, как работает относительное позиционирование, рассмотрим тот же пример, который показан на рис. 15.19, только изменим значение свойства `position` на `absolute` (рис. 15.20):

```
em {
    position: absolute;
    top: 2em;
    left: 3em;
    background-color: fuchsia;
}
```

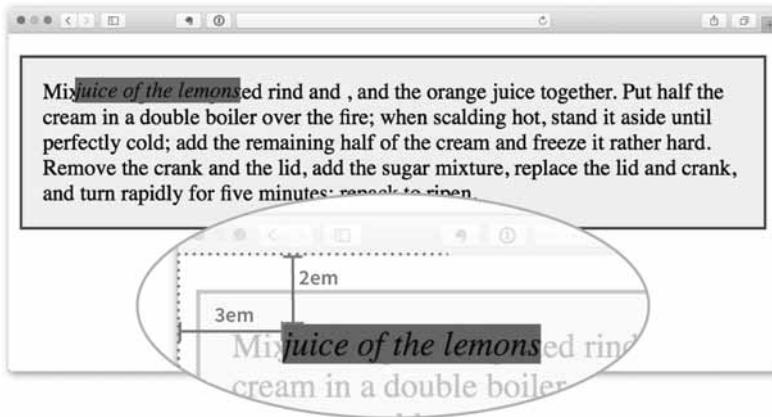


Рис. 15.20. Если элемент абсолютно позиционирован, он удаляется из потока и занимаемое им пространство исчезает

Как показано на рис. 15.20, пространство, ранее занимаемое элементом `em`, теперь исчезло, как и положено при абсолютно позиционируемом элементе, и в новой позиции блок элемента перекрывает окружающий контент, не оказывая никакого влияния на расположение окружающих элементов.

ИСЧЕЗНОВЕНИЕ ЗАНИМАЕМОГО ПРОСТРАНСТВА

Если элемент абсолютно позиционирован, пространство, которое он до этого занимал, исчезает.

глагол элемент `em` на 2ем вниз и на 3ем вправо от верхнего левого угла *области просмотра* (окна браузера).

Но для того чтобы разобраться в сути дела, понимания, что абсолютно позиционированные элементы всегда размещаются относительно области просмотра, недостаточно.

Важно учитывать, что при абсолютном позиционировании элемент позиционируется относительно его ближайшего *содержащего блока*. Просто в этот раз так получилось, что ближайший содержащий блок, показанный на рис. 15.20, является

корневым элементом (`html`), также известным как *начальный содержащий блок*, поэтому значения смещения позиционируют элемент `em` относительно всего документа.

Овладение рычагами управления концепцией содержащего блока и служит первым шагом при решении проблемы абсолютного позиционирования.

Содержащие блоки

Модуль позиционирования макета CSS уровня 3 (CSS Positioned Layout Module, Level 3) гласит: «Положение и размер блока(ов) элемента иногда рассчитываются относительно некоторого прямоугольника, называемого *содержащим блоком* элемента». Очень важно знать, в каком блоке содержится элемент, который необходимо позиционировать. Мы иногда называем это *контекстом позиционирования*.

Спецификация излагает ряд сложных правил для определения содержащего блока элемента, но, в основном, все сводится к следующему:

- если позиционированный элемент не содержится в другом позиционированном элементе, он размещается относительно исходного содержащего блока (созданного элементом `html`);
- но если элемент имеет предка (то есть содержится в элементе), у которого позиция установлена как `relative`, `absolute` или `fixed`, тогда элемент располагается относительно краев именно *этого* элемента.

На рис. 15.20 приведен пример для первого случая: элемент `p`, который содержит абсолютно позиционированный элемент `em`, сам *не позиционируется*, и отсутствуют другие позиционированные элементы, находящиеся выше него в этой иерархии. Поэтому элемент `em` располагается относительно исходного содержащего блока, что эквивалентно области окна просмотра.

Давайте преднамеренно превратим элемент `p` в содержащий блок и посмотрим, что получится. Все, что нам нужно сделать, это применить к нему свойство `position` — и никуда его не перемещать. Самый распространенный способ превратить элемент в содержащий блок — установить его свойство `position` как `relative`, но не перемещать его с помощью каких-либо значений смещения. Об этом речь шла ранее, когда говорилось, что относительное позиционирование применяется для формирования *контекста позиционирования* для абсолютно позиционированного элемента.

В следующем примере мы сохраним правило стиля для элемента `em`, но добавим к элементу `p` свойство `position`, сделав его содержащим блоком для позиционированного элемента `em` (рис. 15.21):

```
p {  
    position: relative;  
    padding: 15px;  
    background-color: #F2F5D5;  
    border: 2px solid purple;  
}
```

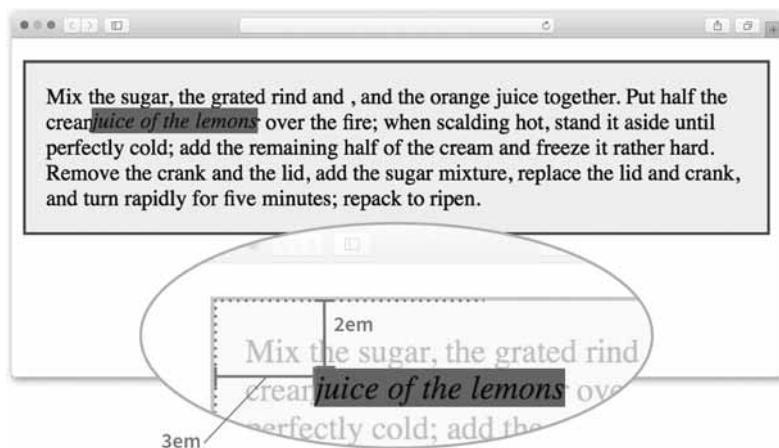


Рис. 15.21. Относительно позиционированный элемент `p` действует как содержащий блок для элемента `em`

Заметим, что элемент `em` теперь расположен на `2em` вниз и на `3em` вправо от верхнего левого угла блока абзаца, а не окна браузера. Обратите также внимание, что

ПОВЕДЕНИЕ СТРОЧНЫХ ЭЛЕМЕНТОВ

Если как содержащие блоки используются строчные элементы (и они могут таковыми быть), позиционированный элемент размещается относительно края области контента, а не края отступа.

он позиционирован относительно *края отступа абзаца* (только внутри границы), а не края области контента. Это — нормальное поведение, если блочные элементы используются как содержащие блоки.

Я собираюсь еще немного покопаться в этом, чтобы раскрыть дополнительные аспекты абсолютно позиционированных объектов. И прямо сейчас добавлю к позиционированному элементу `em` свойства `width` и `margin` (рис. 15.22):

```
em {
    width: 200px;
    margin: 25px;
    position: absolute;
    top: 2em;
    left: 3em;
    background-color: fuchsia;
}
```

Здесь мы можем отметить следующее:

- для абсолютно позиционированных элементов значения смещения применяются к внешним краям поля элементов (краю внешнего поля);
- абсолютно позиционированные элементы всегда ведут себя как элементы уровня блока. Например, поля со всех сторон сохраняются, даже если этот элемент является встроенным. Также можно установить для элемента значение ширины.

Важно отметить, что после позиционирования элемента он становится новым содержащим блоком для всех содержащихся в нем элементов. Например, узкий

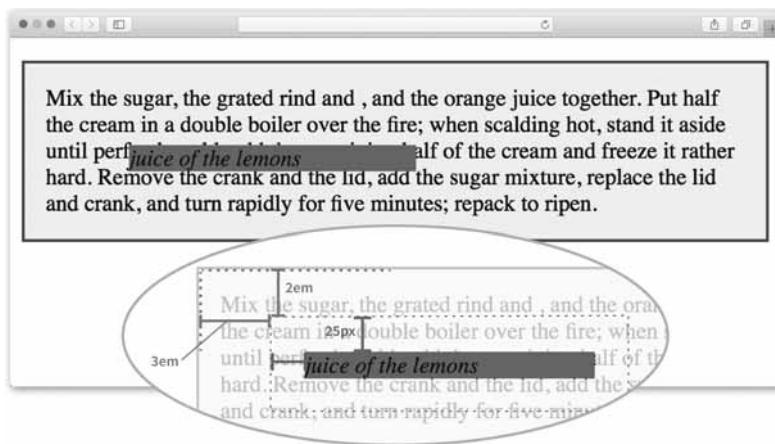


Рис. 15.22. Добавление значений ширины и полей к позиционируемому элементу

элемент `div` размещается в верхнем левом углу страницы, создавая столбец. Если необходимо позиционировать внутри этого элемента `div` изображение со значениями смещения, которые помещают его в верхний правый угол, это изображение отобразится в верхнем правом углу элемента, а не всей страницы. После позиционирования родительского элемента он действует как содержащий блок для элемента `img` и всех других содержащихся в нем элементов.

ПОВЕДЕНИЕ ОТНОСИТЕЛЬНО ПОЗИЦИОНИРОВАННЫХ ЭЛЕМЕНТОВ

Для относительно позиционированных элементов значения смещения измеряются от самого блока (а не от края внешнего поля).

Указание позиции

Теперь, когда вы стали лучше понимать концепцию содержащего блока, давайте уделим некоторое время более глубокому знакомству со свойствами смещения. Пока что мы видели только элемент, сдвинутый на несколько пунктов вниз и вправо, но это, конечно, не все, что вы можете сделать.

Пиксельные единицы измерения

Как уже упоминалось, положительные значения смещения сдвигают блок позиционируемого элемента от указанного края в направлении к центру содержащего блока. Если для стороны не задано значение, устанавливается значение `auto`, и браузер добавляет достаточно пространства, для того чтобы макет был работоспособным. В следующем примере элемент `div#B` содержится внутри элемента `div#A`, размеры которого составляют: 600 пикселов в ширину и 300 пикселов в высоту. Длины указаны в пикселях для всех четырех свойств смещения, что позволяет разместить позиционированный элемент `#B` в определенное место содержащего его элемента `#A` (рис. 15.23):

Разметка

```
<div id="A">
  <div id="B">&nbsp;</div>
</div>
```

Правило стиля

```
div#A {
  position: relative; /* создание содержащего блока */
  width: 600px;
  height: 300px;
  background-color: #C6DE89; /* зеленый */
}
div#B {
  position: absolute;
  top: 25px;
  right: 50px;
  bottom: 75px;
  left: 100px;
  background-color: steelblue;
}
```

div#A (600 пикселов ширина x 300 пикселов высота)

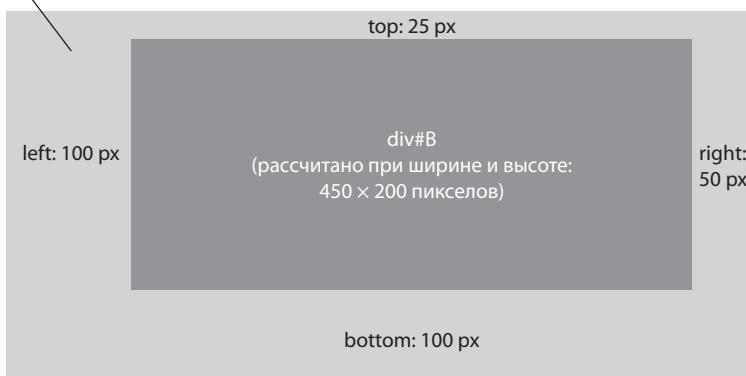


Рис. 15.23. Задание значений смещения для всех четырех сторон позиционируемого элемента

Обратите внимание, что, установив смещения для всех четырех сторон, я косвенно установила размеры позиционированного элемента div#B. Он заполняет теперь пространство 450×200 пикселов, которое остается внутри содержащего блока после применения значений смещения. Если при этом также указать ширину и другие свойства блока для div#B, повысится вероятность возникновения конфликтов — если сумма значений смещений для позиционированного блока не совпадет с доступным пространством внутри содержащего блока.

Спецификация CSS предоставляет сложный набор правил для обработки конфликтов, но вам следует проявлять осторожность, чтобы не переопределить свойства и смещения блока. В общем, значение ширины (с учетом полей, а также отступов и границ, если вы используете модель определения размеров блоков content-box)

и одно-два свойства смещения — это все, что необходимо для получения желаемого макета. Браузер позаботится об остальных расчетах сам.

Процентные значения

Вы также можете указать позиции и с помощью процентных значений. В примере, показанном на рис. 15.24, слева, изображение расположено на полпути (50%) вниз от левого края содержащего блока. В примере, показанном на рис. 15.24, справа, элемент `img` расположен таким образом, что всегда отображается в правом нижнем углу содержащего блока:

```
img#A {  
    position: absolute;  
    top: 50%;  
    left: 0%;  
}  
  
img#B {  
    position: absolute;  
    bottom: 0%;  
    right: 0%;  
}
```

Несмотря на то что в примерах указывается как вертикальное, так и горизонтальное смещение, обычно для позиционируемого элемента предоставляется только одно смещение — например, при перемещении его в поле влево или вправо с использованием свойств `left` или `right`.



Рис. 15.24. Применение процентных значений для позиционирования элемента в содержащем блоке

В процессе выполнении упражнения 15.3 добавим некоторые изменения в домашнюю страницу «Black Goose Bakery», используя возможности абсолютного позиционирования.

СИМВОЛ % ДЛЯ ЗНАЧЕНИЯ 0

Для значения 0 символ % можно опустить, поскольку это значение по существу превращает его в представление длины, равной 0, и приводит к эквивалентному результату.

ПОЗИЦИОНИРОВАНИЕ ЭЛЕМЕНТА В НИЖНЕЙ ЧАСТИ ВЕБ-СТРАНИЦЫ

Будьте осторожны при размещении элементов в нижней части исходного содержащего блока (элемента `html`). И, хотя можно ожидать, что он позиционируется в нижней части всей страницы, браузеры фактически размещают элемент в нижней части окна браузера. Результаты могут быть непредсказуемы. Если нужно разместить что-либо в нижней части страницы, поместите этот объект в элемент содержащего блока в конце исходного документа и исходите из этого.

УПРАЖНЕНИЕ 15.3. АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ

Отличные новости! Пекарня «Black Goose Bakery» получила награду Farmers' Market Award, и у нас появилась возможность представить эту награду на главной странице сайта. Чтобы добавить на страницу сайта графическое изображение награды Farmers' Market Award, мы воспользуемся в этом упражнении абсолютным позиционированием. Откройте версию сайта, которая была сохранена при выполнении *упражнения 15.2*.

- Поскольку это графическое изображение является новым контентом, нужно добавить его разметку в файл `bakery.html`. Но, так как оно не несет существенной информации, добавьте это изображение в новый элемент `div`, в нижнем колон-тизите (`footer`) документа:

```
<footer>
    <p>All content copyright © 2017, Black
    Goose Bistro.</p>
    <div id="award"></div>
</footer>
```

- Хотя изображение награды мы поместили в конец исходного документа `div`, это не означает, что оно должно отображаться именно там. Можем применить абсолютное позиционирование и разместить награду в верхнем левом углу окна просмотра, добавив в таблицу стилей, которая позиционирует `div`, новое правило, — например, таким образом (в нашем случае правило размещено в разделе `/* misc styles */`):

```
#award {
    position: absolute;
    top: 30px;
    left: 50px;
}
```

Сохраните документ и посмотрите на него (рис. 15.25). Измените размер окна браузера, сделав его очень узким, — так, чтобы позиционированное изображение награды перекрыло контент заголовка. Обратите внимание, что при прокрутке документа изображение прокручивается вместе с остальной частью страницы. Попробуйте поэкспериментировать с другими свойствами смещения, чтобы почувствовать, как меняется расположение этого изображения в области просмотра (или, точнее, в начальном содержащем блоке).



Fresh from the Oven

Рис. 15.25. Абсолютно позиционированное изображение награды

П.С. Я знаю, что панель навигации по-прежнему выглядит плохо, но ее исправлением мы займемся в *главе 16*.

Порядок наложения

Прежде чем завершить тему об абсолютном позиционировании, рассмотрим последнюю связанную с этим вопросом концепцию. Как мы видели ранее, абсолютно позиционированные элементы перекрывают другие элементы, поэтому несколько позиционированных элементов могут накладываться друг на друга.

По умолчанию элементы налагаются друг на друга в том порядке, в котором они появляются в документе, но вы можете изменить порядок наложения с помощью свойства `z-index`. Ось `z` можно представить себе в виде линии, которая проходит перпендикулярно странице, — как бы от кончика вашего носа через эту страницу и выходит с другой стороны.

СВОЙСТВО `Z-INDEX`

Свойство `z-index` полезно и для элементов в сетке, которые тоже могут перекрываться (мы рассмотрим это в главе 16).

`z-index`

- **Значения:** число | `auto`.
- **По умолчанию** — `auto`.
- **Применение** — к позиционируемым элементам.
- **Наследование** — нет.

Значением свойства `z-index` служит число (положительное или отрицательное). Чем больше это число, тем выше отображается элемент в стеке (то есть ближе к вашему «носу»). Меньшие числа и отрицательные значения перемещают элемент в стеке пониже. Рассмотрим пример, поясняющий этот момент (рис. 15.26):

Разметка

```
<p id="A"></p>
<p id="B"></p>
<p id="C"></p>
```

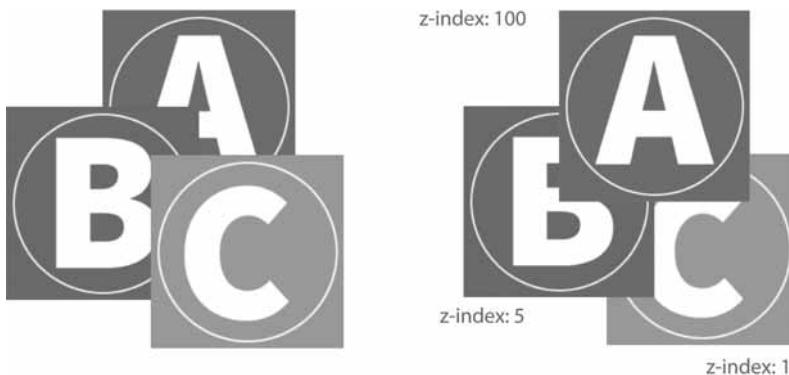
Правило стиля

```
#A {
  z-index: 100;
  position: absolute;
  top: 175px;
  left: 200px;
}
#B {
  z-index: 5;
  position: absolute;
  top: 275px;
  left: 100px;
}
#C {
  z-index: 1;
  position: absolute;
  top: 325px;
  left: 250px;
}
```

Здесь мы имеем три элемента (абзаца), каждый из которых содержит изображение буквы (A, B и C соответственно), абсолютно позиционированные так, что на странице перекрывают друг друга. По умолчанию абзац С отображается сверху, поскольку записан в источнике последним. Однако, присваивая абзацам A и B более высокие значения `z-index`, можно управлять наложением абзацев в порядке, который является предпочтительным.

Заметим, что значения `z-index` не обязаны быть последовательными, и они не относятся ни к чему конкретно. Имеет значение только то, что более высокие числовые значения приводят к размещению элемента в стеке выше.

Честно говоря, на практике свойство `z-index` при разработке макетов страниц применяется весьма редко, но знать о его существовании следует. Если необходимо гарантировать, что позиционируемый элемент всегда окажется сверху, присвойте ему высокое значение `z-index` — например 100 или 1000. Если же нужно, чтобы он находился внизу, присвойте ему отрицательное значение. Величина числа значения не имеет.



По умолчанию элементы, записанные в исходном документе позже, при наложении располагаются поверх предыдущих элементов.

Изменить порядок наложения можно с помощью свойства `z-index`. Элементы с большими значениями этого свойства располагаются поверх элементов, имеющих относительно меньшие значения.

Рис. 15.26. Изменение порядка наложения элементов с помощью свойства `z-index`

ФИКСИРОВАННОЕ ПОЗИЦИОНИРОВАНИЕ

Итак, относительное и абсолютное позиционирование мы рассмотрели, пришло время заняться фиксированным позиционированием.

По большей части фиксированное позиционирование работает так же, как абсолютное позиционирование. Существенное различие их состоит в том, что значения смещения для фиксированных элементов *всегда* относительны области просмотра, а это означает, что позиционированный элемент остается на месте при прокрутке

остальной части страницы. Возможно, вы помните (см. *упражнение 15.3*), что при прокрутке страницы «Black Goose Bakery» графическое изображение награды прокручивалось вместе с документом, хотя оно располагалось относительно начального содержащего блока (что эквивалентно окну просмотра). Иначе обстоит дело с фиксированным позиционированием, когда позиция элемента действительно *фиксирована*.

Фиксированные элементы часто используются для представления пунктов меню и остаются на одном месте вверху, внизу или сбоку экрана, поэтому всегда доступны, даже при прокрутке контента.

Обратите, кстати, внимание: если прикрепить элемент к нижней части области просмотра, в конце документа необходимо оставить достаточно места, чтобы прокручиваемый контент не скрывался за фиксированным элементом. Фиксированные элементы также проблематично применять при печати документа, поскольку они выводятся на печать на каждой странице документа без резервирования места. Поэтому при печати документа лучше отключать фиксированные элементы (настройка печати с помощью элемента `@media` будет рассмотрена в *главе 17*).

Давайте сейчас переключим изображение награды на странице «Black Goose Baker» с абсолютного позиционирования на фиксированное, чтобы удостовериться в различиях (*упражнение 15.4*).

УПРАЖНЕНИЕ 15.4. ФИКСИРОВАННОЕ ПОЗИЦИОНИРОВАНИЕ

Это упражнение весьма простое. Откройте таблицу стилей для нашей пекарни в том виде, в каком оставили ее после выполнения *упражнения 15.3*, и измените правило стиля для `#award div`, чтобы превратить его в `fixed` вместо `absolute`:

```
#award {  
    position: fixed;  
    top: 30px;  
    left: 50px;  
}
```

Сохраните таблицу стилей и откройте страницу в браузере. Начав прокручивать страницу, вы заметите, что изображение награды теперь остается на том же месте, где оно размещено в окне браузера (рис. 15.27). Обратите внимание, что элементы с фиксированным позиционированием могут скрывать контент, «заезжающий» под них при прокрутке страницы. Поэкспериментируйте, чтобы увидеть потенциальные «подводные камни» этого метода и оценить его преимущества.

ОСТЕРЕГАЙТЕСЬ УСТАРЕВШИХ ВЕРСИЙ МОБИЛЬНЫХ БРАУЗЕРОВ!

Свойство `position: fixed` приводит к ошибкам в устаревших версиях мобильных браузеров *Safari* (5, 6 и 7) и *Android* (<4.4). К счастью, эти мобильные браузеры на момент подготовки книги практически устарели, тем не менее при наличии на странице фиксированных элементов следует тщательно тестировать ее с обращением к ряду мобильных устройств.



Рис. 15.27. При прокрутке документа награда остается в том же месте — в верхнем левом углу браузера

На этом о плавании и позиционировании все... В следующей главе вы узнаете о гибких элементах и макетах на основе сетки, которые являются мощными инструментами для разработки общей структуры страницы и конкретных ее функций. Но сначала ответьте на несколько вопросов о плавании и позиционировании.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Прежде чем продолжить изложение, уделим немного времени тому, чтобы узнать, хорошо ли вы усвоили материал этой главы. Ответы на эти вопросы приведены в *приложении 1*.

1. Какие из приведенных утверждений не относятся к плавающим элементам?
 - a. Все плавающие элементы ведут себя как блочные элементы.
 - b. Поплавки расположены напротив края отступа содержащего элемента.
 - c. Контент встроенных элементов обтекает поплавок, но блок элементов не изменяется.
 - d. Следует предоставить свойство width для плавающих элементов блока.
2. Какие из этих правил стиля неверны? Почему?
 - a. img { float: left; margin: 20px; }
 - b. img { float: right; width: 120px; height: 80px; }
 - c. img { float: right; right: 30px; }
 - d. img { float: left; margin-bottom: 2em; }

3. Как убедиться, что элемент нижнего колонтитула всегда начинается на странице ниже любых всплывающих врезок?
4. Запишите название метода или методов позиционирования (статическое, относительное, абсолютное или фиксированное), которое наилучшим образом соответствует каждому из следующих описаний:
 - a. Позиционирует элемент относительно содержащего блока.
 - b. Удаляет элемент из нормального потока.
 - c. Всегда размещает элемент относительно окна просмотра.
 - d. Позиционированный элемент может перекрывать другой контент.
 - e. Позиционирует элемент в нормальном потоке.
 - f. Сохраняется пространство, которое элемент занимал в нормальном потоке.
 - g. Пространство, которое элемент занял бы в нормальном потоке, удалено.
 - h. Вы можете изменять порядок наложения с помощью свойства `z-index`.
 - i. Позиционирует элемент относительно его исходного положения в нормальном потоке.

ОБЗОР CSS: СВОЙСТВА ПЛАВАЮЩИХ ЭЛЕМЕНТОВ И ПОЗИЦИОНИРОВАНИЯ

В табл. 15.1 в алфавитном порядке приводятся краткие описания свойств плавающих элементов и позиционирования.

Таблица 15.1. Свойства плавающих элементов и позиционирования

Свойство	Описание
<code>Clear</code>	Предотвращает размещение элемента рядом с «поплавком»
<code>Float</code>	Перемещает элемент вправо или влево и позволяет последующему тексту обтекать его
<code>position</code>	Определяет метод позиционирования, который будет применен
<code>top, bottom, right, left</code>	Определяет величину смещения от каждого из соответствующих краев
<code>shape-outside</code>	Заставляет контент обтекать форму вместо ограничивающего поплавок прямоугольника
<code>shape-margin</code>	Добавляет поле для свойства <code>shape-outside</code>
<code>shape-image-threshold</code>	Определяет пороговое значение альфа-канала, используемого для создания формы обтекания
<code>z-index</code>	Определяет порядок появления в стеке перекрывающихся позиционированных элементов

ГЛАВА 16

CSS-МАКЕТИРОВАНИЕ С ПОМОЩЬЮ FLEXBOX И GRID

В этой главе...

- ▶ *Flex-элементы и flex-контейнеры*
- ▶ *Изменение направления потока*
- ▶ *Выравнивание flex-элемента*
- ▶ *Управление flex-элементами*
- ▶ *Контейнеры и элементы сетки*
- ▶ *Установка сеточного шаблона*
- ▶ *Размещение элементов в сетке*
- ▶ *Неявные средства сетки*
- ▶ *Выравнивание элементов сетки*

Приготовьтесь... начинается очередная громадная глава! И здесь вы познакомитесь с двумя универсальными инструментами CSS-макетирования веб-страниц:

- инструментом Flexbox — позволяющим осуществлять управление расположением элементов вдоль одной оси;
- инструментом Grid — служащим для создания макетов на основе сетки, подобных тем, которые десятилетиями применяли дизайнеры печати.

Каждый инструмент имеет свое специальное назначение, но вы можете совместно использовать их для создания макетов, о которых до сих пор оставалось только мечтать. Например, можно создать общую структуру страницы на основе сетки и использовать flexbox, чтобы разместить заголовок и элементы навигации. Используйте каждый инструмент для того, для чего он лучше всего подходит — не обязательно ограничиваться только одним из них.

Теперь, когда браузеры начали поддерживать эти методы, дизайнеры и разработчики получили широкие возможности для создания сложных макетов, обладающих тем уровнем гибкости, который необходим при работе с экранами различных размеров. Как только устаревшие браузеры «выйдут из игры», можно будет попрощаться с нашими прежними подходами по формированию макетов с помощью плавающих режимов работы (хотя пока еще именно они приводят к успеху).

Нетрудно заметить, что эта глава большая. *Реально* большая. Это связано с тем, что рассматриваемые спецификации содержат множество параметров и новых понятий, которые нуждаются в объяснениях и примерах. Трудно усвоить это все и сразу, поэтому я рекомендую рассматривать эту главу как две мини-главы и потратить некоторое время на то, чтобы освоить каждую технику в отдельности.

CSS FLEXBOX: СОЗДАНИЕ ГИБКИХ КОНТЕЙНЕРОВ

Модуль CSS Flexible Box Layout (CSS-модуль макета гибкого контейнера), также имеющийся в CSS, предоставляет дизайнерам и разработчикам удобный инструмент для размещения компонентов веб-страниц, таких, как строки меню, списки продуктов, галереи и многое другое.

Согласно спецификации:

Определяющим аспектом гибкого макета (flex-макета) является возможность «сгибать» flex-элементы, изменяя их ширину/высоту, чтобы заполнить доступное пространство в основном объеме.

Это означает, что такой макет позволяет элементам растягиваться или сжиматься внутри своих контейнеров, предотвращая появление как неиспользуемого пространства, так и переполнения. Благодаря такому подходу макеты могут соответствовать различным размерам области просмотра. Среди других преимуществ можно отметить следующие:

- возможность сделать все соседние предметы одинаковой высоты;
- легкое горизонтальное и вертикальное центрирование (которое также можно реализовать и с помощью устаревших CSS-методов);
- возможность изменять порядок отображения элементов, независимо от заданного в источнике.

Модель макета Flexbox имеет высокую надежность, но, поскольку она разрабатывалась с учетом обеспечения максимальной степени гибкости, для ознакомления с нею потребуется определенное время (по крайней мере, мне пришлось достаточно позиться). В процессе изучения материала мне помогло следующее представление: все дочерние элементы того элемента, которому предписано превратиться в «гибкий контейнер» (flexbox), выстраиваются на одной оси, как «бусы на нитке». «Нитка» может быть натянута горизонтально, висеть вертикально и даже складываться в несколько раз, но «бусинки» всегда располагаются на одной «нитке» (или, если использовать правильный термин, на одной оси). Если вы захотите выстроить объекты и по горизонтали, и по вертикали, то для этого следует использовать инструмент CSS Grid, о котором я расскажу в следующем разделе этой главы .

Прежде чем мы углубимся в изучение материала, я хочу дать вам краткую справку о поддержке браузерами этих новых технологий. Все текущие версии браузеров

поддерживают новейшую спецификацию W3C Flexible Box Layout Module, но при использовании устаревших браузеров вам потребуется указывать префиксы и применять другие устаревшие свойства и значения. В целях упрощения подхода, применяемого в самом начале обучения, я придерживалась текущих стандартных свойств, но создание действенных таблиц стилей из-за необходимости учитывать возможность устаревших браузеров может потребовать знакомства с большим объемом кода. В разд. «Браузеры, поддерживающие Flexbox», завершающем раздел, посвященный Flexbox, приводится подробная информация о поддержке браузерами этой технологии.

МНОГОКОЛОНОЧНЫЙ МАКЕТ

Третий инструмент макетирования CSS3, который широко используется на практике, — многоколоночный макет. Модуль Multi-column Layout Module (модуль многоколоночного макета), доступный на сайте по адресу: w3.org/TR/css-multicol-1, предоставляет инструменты, применяемые для разбивки текстового контента на несколько столбцов, подобно тому как это происходит при верстке газет (рис. 16.1). Этот модуль разработан в целях увеличения степени гибкости. Благодаря ему можно выстроить столбцы таким образом, что их ширина и количество будут автоматически соответствовать доступному пространству.

Поскольку эта глава слишком велика, чтобы охватить и эту технологию тоже, соответствующий урок я вынесла в статью «Multicolumn Layout», формат PDF), доступную по адресу: learningwebdesign.com/articles/.

Creams and Milks	Cream should be rather rich, but not double cream. If pure raw cream is stirred rapidly, it swells and becomes frothy, like the beaten whites of eggs, and is "whipped cream." To prevent this in making Philadelphia Ice Cream, one-half the cream is scalded, and when it is very cold, the remaining half of raw cream is added. This gives the smooth, light and rich consistency which makes these creams so different from others.	The time for freezing varies according to the quality of cream or milk or water; water ices require a longer time than ice creams. It is not well to freeze the mixtures too rapidly; they are apt to be coarse, not smooth, and if they are churned before the mixture is icy cold they will be greasy or "buttery." The average time for freezing two quarts of cream should be ten minutes; it takes but a minute or two longer for larger quantities.
Use of Fruits Ordinary fruit creams may be made with condensed milk at a cost of about fifteen cents a quart, which, of course, is cheaper than ordinary milk and cream. In places where neither cream nor condensed milk can be purchased, a fair ice cream is made by adding two tablespoonsfuls of olive oil to each quart of milk. The cream for Philadelphia Ice	Use fresh fruits in the summer and the best canned unsweetened fruits in the winter. If sweetened fruits must be used, cut down the given quantity of sugar. When acid fruits are used, they should be added to the cream after it is partly frozen.	Pound the ice in a large bag with a mallet, or use an ordinary ice shaver. The finer the ice, the less time it takes to freeze the cream. A four quart freezer will require ten pounds of ice, and a quart and a pint of coarse rock salt. You may pack the freezer with a layer of ice three inches thick, then a layer of salt one inch thick, or mix the ice and salt in the tub and shovel it around the freezer.

Рис. 16.1. Пример текста, оформленного с помощью свойств многоколоночного макета

Настройка flex-контейнера

Ранее нами уже рассматривался режим блочной разметки, применяемый для размещения элементов в обычном потоке, и режим строчной разметки, используемый для отображения контента внутри потока по горизонтали. Flexbox — это еще один

ЗНАЧЕНИЕ `INLINE-FLEX`

Значение `inline-flex` генерирует линейный уровень flex-контейнера. В этой главе мы сосредоточимся на наиболее используемом значении `flex`.

После этого элемент превращается во flex-контейнер, а все его непосредственные дочерние элементы (элементы `div`, пункты списков, абзацы и т. д.) автоматически становятся flex-элементами этого контейнера и выравниваются в нем по flex-линиям (как бусины по ниткам).

На рис. ЦВ-16.2 показан эффект простого добавления свойства `display: flex` к элементу `div`, в результате чего и осуществляется переход в режим Flexbox. Я добавила к контейнеру синюю границу, чтобы выделить его. Для экономии места чисто косметические стили, такие, как цвета и шрифты, здесь просто не показаны:

Разметка

```
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
```

Правило стиля

```
#container {
  display: flex;
}
```

ЕЩЕ КОЕ-ЧТО О FLEXBOX

Отметим несколько моментов, которые следует учитывать при работе с Flexbox и контроле над поведением flex-элементов:

- свойства `float`, `clear`, многоколоночный макет и свойство `vertical-align` не работают с элементами в режиме flexbox;
- в режиме flexbox поля не колapsируют. Края полей размещаются в начале или в конце flex-линии и не перекрывают отступ контейнера. Поля соседних элементов складываются;
- спецификация рекомендует избегать процентных значений для полей и отступов flex-элементов вследствие непредсказуемости получаемых результатов.

режим макетирования, которому присуще собственное поведение. Для перевода элемента в этот режим присвойте его свойству `display` значение `flex` или `inline-flex`.

Как можно видеть, элементы выстроены в ряд слева направо, что соответствует заданному по умолчанию поведению Flexbox, когда страница написана на английском или другом языке, в котором текст читается слева направо. Надо отметить, что заданное по умолчанию направление flex-контейнера соответствует направлению языка, на котором написана страница. По умолчанию для иврита и арабского языка отображение выполняется в направлении справа налево, а при использовании столбцов — в вертикальном направлении. Поскольку по умолчанию задано не одно направление, терминология, указывающая направления, немного абстрактна. Вы поймете, что я имею в виду, когда мы в следующем разделе станем рассматривать «поток».

Следует учесть, что любой flex-элемент можно превратить во flex-контейнер, присвоив его свойству `display` значение `flex`, в результате чего мы получим *вложенный flex-контейнер* — вы сами попробуете это в ближайшем упражнении. Для некоторых Flexbox-решений вложенные flex-контейнеры применяются при нескольких уровнях вложенности.

РЕСУРСЫ FLEXBOX

В этой главе детально описано устройство контейнера Flexbox, но если вам все же нужны дополнительные подробности, воспользуйтесь Интернетом для получения доступа к нескольким перспективным и практическим руководствам. Когда будете выполнять поиск информации в Интернете, убедитесь в том, что нашли публикации 2015 года и более новые, поскольку информация, основанная на ранних версиях спецификаций, стремительно устаревает. А далее приведены некоторые сайты, которые я считаю наиболее полезными или наиболее интересными:

- «*A Complete Guide to Flexbox*» (css-tricks.com/snippets/css/a-guide-to-flexbox/) — это одна из наиболее популярных ссылок на Flexbox, под которой скрыт обзор функций Flexbox от Криса Койера (Chris Coyier). Многие разработчики, использующие технологию Flexbox, постоянно обращаются к этому обзору;
- «*Flexbox Froggy*» (flexboxfroggy.com/) — не упустите возможности поиграть в эту онлайн-игру, которая поможет вам освоить Flexbox. Вы научитесь многому, помогая разноцветным лягушкам вернуться к своим лилиям;
- «*What the Flexbox?!*» (flexbox.io/) — здесь Уэс Бос (Wes Bos) знакомит вас со свойствами Flexbox, а также предлагает несколько проектов кода, которые включены в бесплатную серию, состоящую из 20 видеороликов;
- «*Flexbox Playground*» (codepen.io/enxaneta/full/adLPwv/) — как следует из ее названия, страница от Габи (Gabi) позволяет поэкспериментировать со всеми свойствами и значениями Flexbox и быстро ознакомиться с результатами. Это хорошее место для ознакомления с возможностями Flexbox;
- «*Flexy Boxes*» (the-echoplex.net/flexyboxes/) — еще одна «детская площадка» Flexbox и генератор кода.

Управление «потоком» в контейнере

Как только вы превратите элемент во flex-контейнер, вы сможете задать для него несколько свойств, позволяющих управлять тем, как элементы перемещаются внутри него. Понятие «поток» относится к направлению, в котором располагаются flex-элементы, а также определяет, разрешено ли их нанизывать на дополнительные линии.

Свойство `flex-direction` (задание направления потока)

В общем случае можно удовлетвориться выстроеннымми в ряд элементами (как показано на рис. ЦВ-16.2), а можно обратиться к некоторым возможностям, управляемым свойством `flex-direction`:

flex-direction

- **Значения:** row | column | row-reverse | column-reverse.
- **По умолчанию — row.**
- **Применение —** к flex-контейнерам.
- **Наследование —** нет.

НАПРАВЛЕНИЕ СТРОК И СТОЛБЦОВ

В системах письма с горизонтальным направлением строк текста ключевое слово `row` размещает элементы по горизонтали, подобно тому как европейцы и американцы обычно представляют «строки». Для вертикально ориентированных языков ключевое слово `row` выравнивает элементы по вертикали в соответствии с заданным по умолчанию направлением системы их письма. Точно так же применение ключевого слова `column` приведет к горизонтально выровненным элементам для вертикально ориентированных языков.

Тем не менее поскольку в книге описывается создание сайтов на горизонтально ориентированных языках, я придерживаюсь соглашений, заключающихся в том, что `row` = «горизонтальный», а `column` = «вертикальный».

ГЛАВНАЯ И ПОПЕРЕЧНАЯ ОСИ

Главная ось — направление потока, которое указано для flex-контейнера. Поперечная ось перпендикулярна главной оси.

тальной, задание `column` — что она будет вертикальной (опять же, направление строк и столбцов зависит от языка, как отмечается во врезке «Направление строк и столбцов»). Поперечной осью служит любое направление, перпендикулярное главной оси: вертикальное — для `row`, горизонтальное — для `column`.

СОХРАНЕНИЕ ПРЯМЫМИ ГЛАВНОЙ И ПОПЕРЕЧНОЙ ОСЕЙ

Сохранение прямыми главной и поперечной осей при переключении между рядами и столбцами может служить своего рода умственной гимнастикой и является одним из самых нетривиальных умений при использовании Flexbox. По мере приобретения определенного практического навыка вы вполне с этим справитесь.

Заданным по умолчанию значением является `row`, показанное в предыдущем примере на рис. ЦВ-16.2 (см. также врезку «Направление строк и столбцов»). С помощью значения `column` можно указать, что элементы (пункты) выравниваются вертикально. Свойства `row-reverse` и `column-reverse` располагают элементы в указанных направлениях, но начинаются они от конца, и заполнение идет в противоположном направлении. На рис. ЦВ-16.3 показано влияние каждого значения в применении к приведенному простому примеру.

Ознакомив вас с Flexbox в действии, я хочу уточнить формальную терминологию Flexbox. Поскольку система не имеет средств явного указания направлений, в значениях свойств не упоминаются варианты: «левый», «правый», «верхний» или «нижний». Вместо этого речь ведется о *главной оси* и *поперечной оси*. Главная ось соответствует направлению потока, заданному для flex-контейнера. Для горизонтально ориентированных языков задание `row` означает, что главная ось окажется горизон-

Получив представление об осях, познакомьтесь также и с другими компонентами системы Flexbox, что значительно облегчит вам изучение ее свойств (рис. ЦВ-16.4). Как главная, так и поперечная оси имеют начало и конец — в зависимости от направления потока элементов. *Главный размер* — это ширина (или высота, если это столбец) контейнера вдоль главной оси, *поперечный размер* — высота (или ширина, если это столбец) вдоль поперечной оси.

Свойство **flex-wrap** (разбивка группы элементов на несколько линий)

Если в контейнере содержится большое или неизвестное количество flex-элементов и нежелательно, чтобы все они ютились в доступном пространстве, можно разрешить разбивку их группы на дополнительные линии с помощью свойства **flex-wrap**:

flex-wrap

- **Значения:** nowrap | wrap | wrap-reverse.
- **По умолчанию** — nowrap.
- **Применение** — к flex-контейнерам.
- **Наследование** — нет.

По умолчанию группа элементов не разбивается на дополнительные линии (nowrap), ключевое же слово wrap подключает возможность ее разбивки на несколько линий в направлении от начала поперечной оси к ее концу. Например, если направление row, линии располагаются сверху вниз.

Значение wrap-reverse разбивает группу элементов на несколько строк, но располагает их в противоположном направлении: от конца поперечной оси к ее началу (то есть снизу вверх). И, хотя такой вариант представляется несколько экзотическим, трудно исключить, что не появится повод для использования этого значения.

Я добавила несколько элементов div в наш пронумерованный пример flexbox и придала flex-элементам значение ширины, равное 25%, поэтому только четыре элемента соответствуют ширине контейнера. На рис. ЦВ-16.5 показан результат сравнения различных значений разбиения для случая, когда значением свойства **flex-direction** является row — как и принято для него по умолчанию:

Разметка

```
<div id="container">
  <div class="box box1">1</div>
  <!-- more boxes here -->
  <div class="box box10">10</div>
</div> }
```

Правила стиля

```
#container {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}
.box {
  width: 25%;
}
```

По умолчанию, если свойству `flex-direction` присвоено значение `column`, контейнер растягивается для включения элементов по высоте. Чтобы заметить разбиение, установите высоту контейнера, как это сделано в следующем примере. На рис. ЦВ-16.6 показано, каким образом выполняется разбиение для каждого из ключевых слов `flex-wrap`. Обратите внимание, что элементы по-прежнему имеют ширину родительского контейнера, равную 25%, поэтому между столбцами остается пространство:

```
#container {  
    display: flex;  
    height: 350px;  
    flex-direction: column;  
    flex-wrap: wrap;  
}  
.box {  
    width: 25%;  
}
```

Свойство `flex-flow` (совмещение свойств `flex-direction` и `flex-wrap`)

Сокращенное свойство `flex-flow` делает задание свойств `flex-direction` и `flex-wrap` коротким и приятным. Если одно из них не задать, будет принято его значение по умолчанию, а это означает, что вы можете использовать `flex-flow` для задания одного или обоих направлений и разбивки:

flex-flow

- **Значения:** `flex-direction flex-wrap`.
- **По умолчанию** — `row nowrap`.
- **Применение** — к flex-контейнерам.
- **Наследование** — нет.

Применяя свойство `flex-flow`, можно следующим образом сократить предыдущий пример (см. рис. ЦВ-16.6):

```
#container {  
    display: flex;  
    height: 350px;  
    flex-flow: column wrap;  
}
```

Вы только лишь прикоснулись к Flexbox, но у вас уже есть все необходимое, чтобы привести в порядок уродливое навигационное меню на странице нашей пекарни, выполнив упражнение 16.1.

УПРАЖНЕНИЕ 16.1. ФОРМИРОВАНИЕ ПАНЕЛИ НАВИГАЦИИ С ПОМОЩЬЮ FLEXBOX

Откройте в текстовом редакторе последнюю версию таблицы стилей для домашней страницы «Black Goose Bakery». Если вы не следовали инструкциям из преды-

дущей главы, загрузите обновленную копию файла `bakery-styles.css` из материалов главы 16.

1. Итак, файл `bakery-styles.css` уже открыт у вас в текстовом редакторе. Начните с создания самого простого элемента `ul` в элементе `nav`:

```
nav ul {  
    margin: 0;  
    padding: 0;  
    list-style-type: none;  
}
```

ВНИМАНИЕ!

Обязательно задействуйте один из браузеров, поддерживающих Flexbox (см. далее разд. «Браузеры, поддерживающие Flexbox»).

Превратите элемент `ul` во flex-контейнер, присвоив его свойству `display` значение `flex`. В результате все элементы `li` станут flex-элементами. Поскольку нам нужны строки, а не разбиение, значения по умолчанию для `flex-direction` и `flex-wrap` подойдут, и эти свойства можно опустить:

```
nav ul {  
    ...  
    display: flex;  
}
```

Сохраните документ и просмотрите страницу в браузере. Обратите внимание, что ссылки в ряд уже выстроены, и это может восприниматься как улучшение вида, но работы осталось еще много.

2. А теперь поработаем над отображением ссылок. Начните с отображения элементов `a` в элементах списка `nav`: они будут отображаться как блочные, а не строчные элементы. Придайте им скругленные границы величиной `1px`, задайте отступы внутри границ (`.5em` сверху и снизу, `1em` слева и справа) и поля размером `.5em`, что внесет дополнительное пространство и создаст коричневую панель навигации:

```
nav ul li a {  
    display: block;  
    border: 1px solid;  
    border-radius: .5em;  
    padding: .5em 1em;  
    margin: .5em;  
}
```

3. Нам нужно, чтобы панель меню навигации была центрирована по ширине раздела `nav`. Здесь я немного забегаю вперед, поскольку мы еще не знакомы со свойствами выравнивания, но на самом деле они вполне понятны на интуитивном уровне. Расценивайте эту информацию, как предварительное введение в следующий раздел, и добавьте следующее объявление элемента `nav ul`:

```
nav ul {  
    ...  
    display: flex;  
    justify-content:  
    center;  
}
```

ВАЖНО!

Эта версия сайта пекарни служит отправной точкой для упражнения 16.6, поэтому сохраните ее для дальнейшего применения.

На рис. 16.7 показано, какой вид приобретет меню навигации по завершении этого этапа.



Рис. 16.7. Список ссылок стилизован в виде горизонтальной панели меню

Управление выравниванием flex-элементов в контейнере

К этому моменту нам уже известно, каким образом перейти в режим flexbox, превратив элемент в flex-контейнер, а его дочерние элементы — во flex-элементы. Мы также разобрались, как изменить направление потока и как организовать разбиение группы элементов, чтобы они расположились в несколько линий. Оставшийся нерассмотренным набор свойств контейнера оказывает влияние на выравнивание элементов вдоль главной (justify-content) и поперечной осей (align-items и align-content).

Свойство **justify-content** (выравнивание вдоль главной оси)

По умолчанию flex-элементы имеют ширину, ограниченную содержащимся в них контентом, а это означает, что на flex-линии контейнера может оставаться свободное место (см. рис. ЦВ-16.2). Также по умолчанию элементы следуют друг за другом в направлении слева направо от «главного старта» (это зависит от направления чтения языка страницы и направления flex-линии).

Свойство justify-content указывает, каким образом распределить дополнительное пространство вокруг или между элементами, которые не являются гибкими или достигли своего максимального размера:

justify-content

- **Значения:** flex-start | flex-end | center | space-between | space-around.
- **По умолчанию** — flex-start.
- **Применение** — к flex-контейнерам.
- **Наследование** — нет.

РАСПРЕДЕЛЕНИЕ ДОПОЛНИТЕЛЬНОГО ПРОСТРАНСТВА

Распределить дополнительное пространство вдоль главной оси также можно, увеличив ширину самих flex-элементов для заполнения доступного пространства. Эта часть работы со свойствами flex будет рассмотрена позже.

Поскольку свойство justify-content управляет величиной пространства в самом контейнере, применим это свойство к элементу flex-контейнера:

```
#container {
    display: flex;
    justify-content: flex-start;
}
```

На рис. ЦВ-16.8 показано, каким образом выравниваются элементы с помощью каждого из значений свойства `justify-content`. Как и можно было ожидать, значения `flex-start` и `flex-end` располагают линию элементов по направлению от начала и от конца главной оси соответственно, а значение `center` приводит к ее центрированию.

Применение значений `space-between` и `space-around` требует дополнительного пояснения. Если свойству `justify-content` присвоить значение `space-between`, первый элемент располагается в начальной точке, последний элемент переходит в конечную точку, а оставшееся пространство равномерно распределяется между оставшимися элементами. Свойство `space-around` добавляет одинаковое количество пространства слева и справа от каждого элемента, что приводит к удвоению пространства между соседними элементами.

Если выбран вариант столбца с вертикальной главной осью, применяемые значения оказывают аналогичный эффект. В этом случае для просмотра эффекта нужно, чтобы была явно указана высота контейнера с оставшимся дополнительным пространством. Я изменила размер текста и установила высоту элемента контейнера для рис. ЦВ-16.9 так, чтобы продемонстрировать применение аналогичных ключевых слов по отношению к главной вертикальной оси.

ПРОВЕРЬТЕ СПЕЦИФИКАЦИЮ FLEXBOX НА НАЛИЧИЕ ОБНОВЛЕНИЙ

Поскольку новые ключевые слова, применяемые для выравнивания, добавлены в спецификацию `Grid Layout`, они также доступны и для `Flexbox`, однако поскольку они внесены недавно, то поддерживаются не столь успешно. Обязательно проверьте спецификацию `Flexbox` на наличие обновлений.

ЗАДАНИЕ СВОЙСТВА `JUSTIFY-CONTENT`

Задание свойства `justify-content` должно осуществляться только после того, когда для элементов уже рассчитаны поля и учтен способ, которым элементы установлены как `«flex»`. Если значение `flex` для элементов позволяет им увеличиваться в размерах для заполнения контейнера по ширине, задание `justify-content` не производится.

Свойство `align-items` (выравнивание вдоль поперечной оси)

В предыдущем разделе мы рассмотрели расположении объектов на главной оси, но вы также можете поиграть с выравниванием по поперечной оси: вверх и вниз, когда направление `row` (линия), влево и вправо, если направление `column` (столбец). Выравнивание и распределение по поперечной оси осуществляется с помощью свойства `align-items`:

`align-items`

- **Значения:** `flex-start` | `flex-end` | `center` | `baseline` | `stretch`.
- **По умолчанию** — `stretch`.
- **Применение** — к flex-контейнерам.
- **Наследование** — нет.

На рис. ЦВ-16.10 показаны различные значения свойства `align-items`, применительно к линиям. Чтобы увидеть эффект от применения этих значений, следует

указать высоту контейнера — в противном случае он расширяется для размещения контента без дополнительного пространства. Поэтому я и придала контейнеру высоту — чтобы показать, каким образом элементы располагаются на поперечной оси.

Подобно свойству `justify-content`, свойство `align-items` применяется именно к flex-контейнеру (что может вызвать путаницу, поскольку в названии свойства фигурирует слово «items»):

```
#container {
    display: flex;
    flex-direction: row;
    height: 200px;
    align-items: flex-start;
}
```

Значения `flex-start`, `flex-end` и `center` вам уже знакомы, только в этом случае они относятся к началу, концу и центру поперечной оси. Значение `baseline` приводит к выравниванию базовых линий первых строк текста независимо от их размера. Этот вариант весьма удобен для выравнивания элементов с разными размерами текста — таких, как заголовки и абзацы нескольких элементов. Наконец, значение `stretch`, заданное по умолчанию, приводит к тому, что элементы растягиваются для заполнения поперечной оси.

Если для направления `flex container` выбрано значение `column`, свойство `align-items` выравнивает элементы влево и вправо. Вернитесь к рис. ЦВ-16.2 и ЦВ-16.9 и обратите внимание, что при установке элементов в столбце без предоставления дополнительной информации по выравниванию каждый элемент растягивается на всю ширину поперечной оси, поскольку `stretch` является заданным по умолчанию значением.

Если необходимо для одного или нескольких элементов отменить установку `cross-axis`, примените для таких элементов свойство `align-self`. Это свойство является первым из свойств, применяемых к элементу, а не к самому контейнеру. Свойство `align-self` обладает значениями, аналогичными значениям свойства `align-items`, только в этом случае каждый элемент обрабатывается поочередно:

`align-self`

- **Значения:** `flex-start` | `flex-end` | `center` | `baseline` | `stretch`.
- **По умолчанию** — `stretch`.
- **Применение** — к flex-элементам.
- **Наследование** — нет.

В следующем примере (рис. ЦВ-16.11) показано, что четвертый блок настроен на выравнивание с учетом конца поперечной оси, а другие характеризуются (имеют поведение), связанное с заданным по умолчанию расположением:

```
.box4 {
    align-self: flex-end;
}
```

Свойство **align-content** (выравнивание нескольких линий)

Последнее применяемое для выравнивания свойство — `align-content` — оказывает влияние на то, каким образом вдоль поперечной оси растягиваются несколько flex-линий. Это свойство используется только в том случае, когда свойство `flex-wrap` установлено равным `wrap` или `wrap-reverse` и для выравнивания имеется несколько строк. Если элементы расположены в одной строке, выравнивания не происходит:

align-content

- **Значения:** `flex-start` | `flex-end` | `center` | `space-around` | `space-between` | `stretch`.
- **По умолчанию** — `stretch`.
- **Применение** — к многострочным flex-контейнерам.
- **Наследование** — нет.

Все значения, представленные в списке свойства, должны быть вам знакомы, и работают они ожидаемым образом. Только распределяют они в этот раз дополнительное пространство вдоль поперечной оси именно вокруг нескольких линий.

В следующем примере свойство `align-content` применяется к элементам flex-контейнера (рис. ЦВ-16.12). Для этого контейнера также нужно задать высоту, поскольку иначе контейнер получил бы такую высоту, которая позволила бы вместить весь контент и свободного места не осталось бы:

```
#container {  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap;  
    height: 350px;  
    align-items: flex-start;  
}  
  
box {  
    width: 25%;  
}
```

ОГРАНИЧЕНИЕ СВОЙСТВА

ALIGN-CONTENT

Свойство `align-content` применяется только при наличии для растягивания нескольких flex-линий.

Выравнивание элементов с помощью полей

Поскольку речь у нас идет о выравнивании, я хотела бы показать вам еще один полезный прием, используемый при размещении компонентов с помощью Flexbox.

Строки и панели меню в Интернете применяются повсеместно, и обычно один элемент панели — типа логотипа или поля поиска — визуально отделяется от других. Так вот, для получения в контейнере дополнительного пространства на указанной стороне или на сторонах какого-либо flex-элемента можно использовать поля, тем самым отделяя этот элемент от прочих. Наверняка, все станет более понятно на примере.

Панель меню, показанная на рис. 16.13, состоит из логотипа и четыре опций. Мне желательно, чтобы логотип оставался в левом краю панели, а предлагаемые опции меню находились всегда справа, независимо от ширины области просмотра:

Разметка

```
<ul>
  <li class="logo"></li>
  <li>About</li>
  <li>Blog</li>
  <li>Shop</li>
  <li>Contact</li>
</ul>
```

Правила стиля

```
ul {
  display: flex;
  align-items: center;
  background-color: #00af8f;
  list-style: none; /* removes bullets */
  padding: .5em;
  margin: 0;
}
li {
  margin: 0 1em;
}
li.logo {
  margin-right: auto;
}
```

ИСПОЛЬЗУЙТЕ ПОЛЯ

Для того чтобы добавить пространство по сторонам отдельных элементов, удобно за-действовать поля.



Рис. 16.13. Применение поля для задания пространства вокруг flex-элементов: правое поле элемента логотипа сдвигает оставшиеся элементы вправо

Неупорядоченный список (`ul`) я превратила здесь во flex-контейнер, поэтому элементы его списка (`li`) стали flex-элементами. По умолчанию все элементы контейнера расположатся в начале главной оси (слева), оставляя справа свободное пространство. Установка правого поля элемента логотипа в значение `auto` перемещает это пространство вправо от логотипа, отталкивая все оставшиеся элементы вправо (к «главному концу»).

Этот прием можно использовать для ряда сценариев. Если вы хотите, чтобы последний элемент отображался справа, установите значение `auto` для его левого поля. Необходимо получить одинаковое по величине пространство вокруг центрального элемента списка? Установите в значение `auto` и левое, и правое поля. Нужно переместить кнопку в нижнюю часть столбца? Установите в значение `auto` верхнее поле для

последнего элемента. Дополнительное пространство в контейнере перейдет в поле и оттолкнет соседний элемент.

Итак, мы рассмотрели достаточно много тем, и сейчас самое время применить Flexbox на практике, выполнив *упражнение 16.2*.

ЕЩЕ О СВОЙСТВЕ `JUSTIFY-CONTENT`

При использовании для flex-элемента свойства `margin: auto`, свойство `justify-content` перестает давать видимый эффект, поскольку место для дополнительного пространства на главной оси вы назначили вручную.

УПРАЖНЕНИЕ 16.2. ГИБКОЕ ВСТРОЕННОЕ МЕНЮ

При выполнении этого упражнения мы поработаем со свойствами Flexbox, используя контент, несколько более сложный, чем ссылки в строке меню, и сформируем простое онлайн-меню, включающее несколько пунктов. Как всегда, материалы этого упражнения доступны на сайте по адресу: learningwebdesign.com/5e/materials.

Откройте файл **flex-menu.html** в текстовом редакторе. В этом файле уже имеется подготовленный контент, а также внутренняя таблица стилей со стилями для представления косметических нюансов меню: цветов, шрифтов, границ, интервалов и т. п. Откройте файл в браузере — пункты меню отобразятся в столбце, поскольку являются блочными элементами. Чтобы элементы `div` оболочки `#menu` были лучше видны, я задала им границы.

- Сначала, приложив минимальные усилия для получения максимального эффекта, превратим оболочку `#menu` элемента `div` во flex-контейнер. Для этого в существующее правило для `#menu` добавим следующее объявление:

```
#menu {
    border: 3px solid #783F27;
    display: flex;
}
```

Сохраните файл и перезагрузите страницу в браузере — вы увидите, что flex-элементы расположились в строке (в ряд). Поскольку другие flex-свойства не добавлялись, здесь демонстрируется flexbox-поведение по умолчанию (рис. 16.14):

- каждый элемент (определенный элементом `section`) имеет полную высоту контейнера `#menu`, независимо от своего контента;
- ширина элементов-секций установлена равной 240 пикселам, и этот размер сохраняется (как и предусмотрено по умолчанию). В зависимости от ширины окна браузера можно видеть контент, выходящий за пределы контейнера и, вследствие этого, обрезанный (см. рис. 16.14).

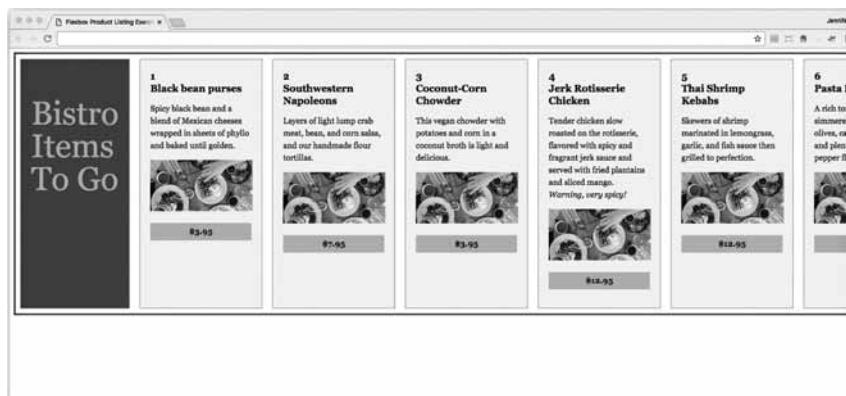


Рис. 16.14. Меню bistro в заданном по умолчанию режиме flexbox: элементы — хотя для них недостаточно места — расположены в одну строку, и контент обрезан

2. По умолчанию flex-элементы отображаются в направлении письма (строки в наших языках следуют в направлении слева направо) — эти элементы пронумерованы для явного отображения их порядка. Добавьте свойство `flex-direction` к имеющемуся правилу `#menu` для просмотра результатов применения некоторых других значений: `row-reverse`, `column`, `column-reverse`:

```
flex-direction: row-reverse;
```

3. Установите свойство `flex-direction` опять равным `row` и поэкспериментируйте с выравниванием вдоль поперечной оси, применяя свойство `align-items` (попробуйте для свойства `align-items` значения: `flex-start`, `flex-end`, `center`, `baseline` и `stretch` — чтобы посмотреть на результаты их применения), после чего установите значение этого свойства, равным `flex-start`:

```
align-items: flex-start;
```

Сохраните результат и выполните перезагрузку — вы увидите, что все элементы располагаются в начале поперечной оси — в нашем случае сверху (рис. 16.15).

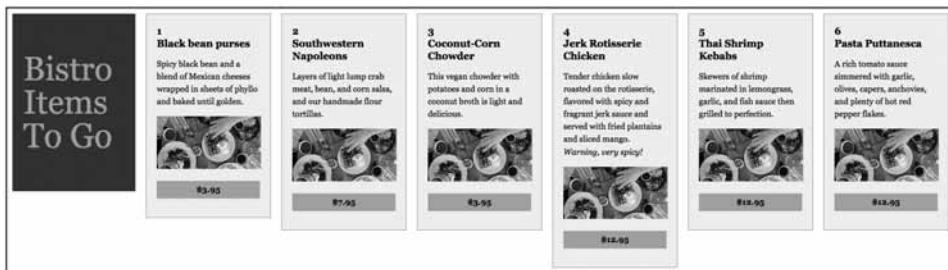


Рис. 16.15. Применение свойства `align-items` для выравнивания элементов от начала поперечной оси (`flex-start`)

4. Для пробы установите значение свойства `align-items`, снова равным `stretch`. Вместо размещения всех элементов в одной строке и их обрезки по краям браузера растянем (перенесем) их на несколько строк, применяя свойство `flex-wrap` для контейнера `#menu`:

```
flex-wrap: wrap;
```

Сохраните файл и рассмотрите его в окне браузера (рис. 16.16). Измените размер окна браузера и понаблюдайте за поведением растяжения строк. Обратите внимание, что каждая flex-линия имеет высоту, равную высоте наиболее высокого ее элемента, но сами линии могут иметь различную высоту в зависимости от контента элементов.

5. Если хотите, можете заменить объявления `flex-direction` и `flex-wrap` одним объявлением `flex-flow`:

```
flex-flow: row wrap;
```

6. По умолчанию элементы на каждой flex-линии выровнены по направлению от начала главной оси (слева). Попробуйте изменить выравнивание элементов вдоль главной оси с помощью свойства `justify-content` (снова воспользуйтесь правилом flex-контейнера `#menu`):

```
justify-content: center;
```

Заметьте, что элементы в контейнере хорошо центрированы, но попробуйте также и другие значения: `flex-start`, `flex-end`, `space-between`, `space-around`.

- И, наконец, сделаем так, чтобы кнопки с ценами располагались в нижней части каждого пункта меню, что вполне возможно, если каждый элемент также является flex-контейнером. Здесь я превращаю каждый элемент во вложенный flex-контейнер путем установки свойств `display` к значению `flex` и указания направления `column`, чтобы элементы продолжали стоять вертикально. Теперь элементы `h2` и `p` становятся flex-элементами flex-контейнера `section`:

```
section {
  ...
  display: flex;
  flex-direction: column;
}
```

При перезагрузке страницы в окне браузере она примет вид, примерно такой же, как и в случае, если секции были бы составлены из блочных элементов. Небольшое различие состоит в том, что соседние поля между элементами складываются, а не коллапсируют.

Теперь перенесите содержащие цены абзацы в нижнюю часть элементов, применяя свойство `margin: auto`. Добавьте это объявление к существующему правилу стиля для элементов с именем класса `price`:

```
.price {
  ...
  margin-top: auto;
}
```

На рис. 16.17 показано окончательное состояние меню «Bistro Items to Go» сложенными flex-контейнерами. Позднее, после изучения свойств, относящихся уже к отдельным элементам, мы продолжим работу над этим файлом, поэтому сохраните его для дальнейшего применения.

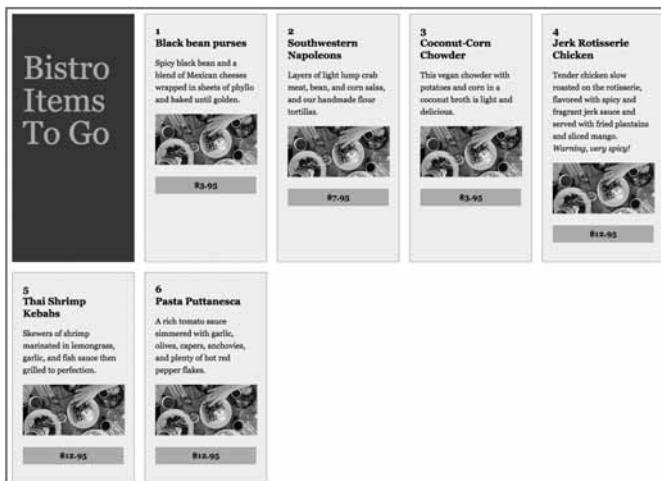


Рис. 16.16. Меню с подключенным растяжением



Рис. 16.17. Меню с выровненными flex-элементами и значениями цен

Свойство **flex** (управление «гибкостью» элементов в контейнере)

Одним из существенных преимуществ модели flexbox является возможность изменять размеры элементов, или «изгибать» их — если воспользоваться формальным термином *flex*, что позволяет им вполне соответствовать доступному пространству. Именно гибкость делает Flexbox столь универсальным инструментом для выполнения разработок с учетом широкого спектра размеров экранов и окон браузеров, с которым сталкиваются веб-дизайнеры. Вся прелесть в том, что браузер вычисляет размеры элементов «на лету», а это значит меньше математики для нас! Итак, приступаем к знакомству со свойствами этой «гибкости».

Ранее мы рассмотрели свойство *justify-content*, которое распределяет свободное пространство в контейнере между и вокруг находящихся в нем элементов вдоль главной оси. Концепция гибкости (*flex*) определяет то, каким образом пространство распределяется внутри элементов, увеличиваются или уменьшаются сами элементы при их подгонке.

Гибкость управляется свойством *flex*, которое указывает, насколько элемент может увеличиваться и уменьшаться, и задает его начальный размер. На самом деле *flex* — это сокращенное свойство для отдельных свойств: *flex-grow*, *flex-shrink* и *flex-basis*, причем спецификация настоятельно рекомендует разработчикам использовать сокращенное свойство *flex*, а не указывать эти отдельные свойства, чтобы избежать противоречивых значений по умолчанию и гарантировать, что авторы для каждого случая применения свойства *flex* имели в виду все три его компонента:

flex

- **Значения:** none | 'flex-grow flex-shrink flex-basis'.
- **По умолчанию** — 0 1 auto.

- **Применение** — к flex-элементам.
- **Наследование** — нет.

Значением свойства `flex`, как правило, служат три flex-свойства, указанные в следующем порядке:

```
flex: flex-grow flex-shrink flex-basis;
```

Для свойств `flex-grow` и `flex-shrink` значения 1 и 0 выполняют роль переключателей «on/off» (вкл/выкл), причем 1 — это «подключение», которое позволяет элементу растягиваться (`grow`) либо сжиматься (`shrink`), а значение 0 этому препятствует. Свойство `flex-basis` устанавливает начальные размеры: либо как определенный размер, либо в виде размера, основанного на контенте.

В следующем небольшом примере элемент списка стартует от значения ширины, равной 200 пикселам, ему разрешено растягиваться для заполнения свободного пространства (1), но не разрешено сжиматься (0), становясь уже, чем исходные 200 пикселов:

```
li {  
    flex: 1 0 200px;  
}
```

ОГРАНИЧЕНИЕ FLEX-СВОЙСТВ

Flex-свойства применяются к flex-элементам, а не к flex-контейнеру.

Вы сейчас получили только общее представление о гибкости. Далее мы подробно рассмотрим увеличение, сжатие и базовый размер в указанном порядке.

Но сначала важно отметить, что свойство `flex` и его компоненты относятся исключительно к flex-элементам, а не к контейнерам. Ясное понимание того, какие свойства относятся к контейнеру, а какие — к элементам, является одним из непременных условий успешного применения Flexbox (см. врезку «Flex-свойства», где приведен удобный список свойств, разбитых по категориям).

FLEX-СВОЙСТВА

После ознакомления со всеми свойствами модуля Flexible Box Module желательно сразу отделить свойства, применяемые к контейнерам, от свойств, применяемых для flex-элементов:

- **Свойства контейнера** — примените следующие свойства к flex-контейнеру:

```
display  
flex-flow  
flex-direction  
flex-wrap  
justify-content  
align-items  
align-content
```

- **Свойства flex-элементов** — примените следующие свойства к flex-элементам:

```
align-self  
flex  
flex-grow  
flex-shrink  
flex-basis  
order
```

Свойство **flex-grow** (растягивающиеся элементы)

ИСПОЛЬЗУЙТЕ ВМЕСТО СВОЙСТВА **FLEX-GROW** СОКРАЩЕННОЕ СВОЙСТВО **FLEX**

Свойство **flex-grow** — это отдельное свойство, определяющее, каким образом элемент можно растягивать. Разработчикам рекомендуется использовать вместо него сокращенное свойство **flex**.

Вот что пишет в документации W3C о свойстве **flex-grow**: «Если у вас есть элемент с **flex-grow: 1**, то он будет растягиваться на все свободное пространство контейнера». Но это не совсем так. Рассмотрим пример.

flex-grow

- **Значения:** число.
- **По умолчанию** — 0.
- **Применение** — к flex-элементам.
- **Наследование** — нет.

Если установить значение **flex-grow** для всех элементов контейнера равным 1, браузер занимает любое свободное пространство, доступное вдоль главной оси, и применяет его в равной степени к каждому элементу, растягивая их на одинаковую величину.

Обратимся к простому примеру, уже рассмотренному в этой главе ранее, и разберемся, как работает это свойство с различными flex-настройками. На рис. ЦВ-16.18 показано, что произойдет, если свойство **flex-grow** установить в значение 1 для всех элементов контейнера (свойства **flex-shrink** и **flex-basis** остаются со своими заданными по умолчанию значениями). Сравните результат с аналогичным примером, когда свойство **flex-grow** установлено в заданное по умолчанию значение 0 (аналогичное поведение наблюдалось в примере, представленном на рис. ЦВ-16.2):

Разметка

```
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
```

Правило стиля

```
.box {
  ...
  flex: 1 1 auto;
}
```

Первым значением в свойстве **flex** указывается, может ли элемент растянуться больше (и в какой пропорции), — другими словами, это значение **flex-grow**.

По умолчанию для свойства **flex-grow** установлено значение 0, означающее, что элемент не может стать шире, чем размер его контента или указанное для него значение ширины. Поскольку элементы по умолчанию не растягиваются, на них могут оказывать влияние свойства выравнивания. Если же свободного пространства внутри элементов нет, не будет выполняться и выравнивание:

Если указать для элемента более высокое значение свойства `flex-grow`, оно подействует как коэффициент, который позволит ему занять внутри контейнера соответственно больше места. Например, придав элементу `box4` значение `flex-grow: 3`, мы дадим ему возможность занять в три раза больше места, чем остальные элементы, имеющие значения `flex-grow: 1` (рис. ЦВ-16.19):

```
.box4 {  
  flex: 3 1 auto;  
}
```

Обратите внимание, что полученный элемент не стал в три раза шире остальных элементов — к нему просто добавилось в три раза больше места.

Если места на линии недостаточно, есть вероятность, что добавленного к элементам пространства окажется мало и разницу вообще трудно будет заметить. В таком случае, возможно, вам надо будет поиграть со значениями `flex-grow` и шириной окна браузера, пока вы не получите желаемый эффект.

Теперь, когда вы познакомились со свойством растягивания, свойство сжатия не должно вызвать у вас непонимание, поскольку основано на тех же принципах.

Свойство `flex-shrink` (сжатие элементов)

Второе значение свойства `flex` — свойство `flex-shrink`, применяется, если контейнер недостаточно широк и не может полностью вместить все свои элементы. По сути, свойство `flex-shrink` удаляет некоторое пространство внутри элементов, что приводит к сжатию их до нужного размера в соответствии с заданным соотношением:

`flex-shrink`

- **Значения:** число.
- **По умолчанию** — 0.
- **Применение** — к flex-элементам.
- **Наследование** — нет.

По умолчанию значение свойства `flex-shrink` установлено равным 1 — то есть, если не предпринимать никаких действий, элементы сжимаются в одинаковой пропорции. Если свойство `flex-shrink` установлено в 0, элементам не позволено сжиматься, и они могут выступать из своего контейнера и за пределы области просмотра. Наконец, как и в случае со свойством `flex-grow`, более высокое значение выступает в роли коэффициента — элемент, имеющий значение свойства `flex-shrink`, равное 2, сжимается вдвое сильнее по сравнению с элементом, имеющим значение, равное 1. В общем случае нет никакой надобности указывать значение коэффициента сжатия — достаточно только подключить сжатие (1) или отключить его (0).

ИСПОЛЬЗУЙТЕ ВМЕСТО СВОЙСТВА `FLEX-SHRINK` СОКРАЩЕННОЕ СВОЙСТВО `FLEX`

Свойство `flex-shrink` — это отдельное свойство, определяющее, каким образом элемент можно сжимать. Разработчикам рекомендуется использовать вместо него сокращенное свойство `flex`.

ПОВЕДЕНИЕ СВОЙСТВА `FLEX-SHRINK` ПО УМОЛЧАНИЮ

По умолчанию (значение `flex-shrink: 1`) — если контейнер недостаточно широк, элементы будут сжиматься, чтобы уместиться в имеющееся пространство.

Flex-элементы прекращают сжиматься, как только достигают своего минимального размера (определенного отношением `min-width/min-height`). По умолчанию (если значения `min-width/min-height` установлены в `auto`) этот минимум основан на минимальном размере контента (`min-content`). Можно легко установить его как ноль или `12em`, или как любое другое значение длины, которое покажется удобным. Следите за этим эффектом, если глубоко вложенные элементы заставляют flex-элемент становиться шире, чем ожидалось.

Действие свойства `flex-shrink` показано на рис. ЦВ-16.20 в следующем разделе.

Свойство `flex-basis` (поддержка начального размера)

Третье значение свойства `flex` — `flex-basis` определяет начальный размер элемента до его выравнивания, растягивания или сжатия. Это свойство можно применять для flex-элементов вместо свойства `width` (или свойства `height` — для столбцов):

flex-basis

- **Значения:** `длина | процентное соотношение | content | auto`.
- **По умолчанию** — `auto`.
- **Применение** — к flex-элементам.
- **Наследование** — нет.

В следующем примере свойство `flex-basis` для элементов контейнера установлено равным 100 пикселям (рис. ЦВ-16.20). При этом элементы могут ужиматься, что позволит им уместиться в доступном пространстве (`flex-shrink: 1`), но им не разрешается увеличиваться в ширину (`flex-grow: 0`) больше чем на 100 пикселов, что ведет к образованию в контейнере свободного пространства:

```
box {
  flex: 0 1 100px;
}
```

ИСПОЛЬЗУЙТЕ ВМЕСТО СВОЙСТВА `FLEX-BASIS` СОКРАЩЕННОЕ СВОЙСТВО `FLEX`

Свойство `flex-basis` — это отдельное свойство, которое устанавливает начальный размер элемента. Разработчикам рекомендуется использовать вместо него сокращенное свойство `flex`.

также можете точно задавать ширину контента, задав значение свойства `flex-basis` с помощью ключевого слова `content`. Однако эта возможность пока еще не имеет удовлетворительной поддержки браузерами.

В следующем примере начальное значение ширины для всех элементов установлено в 100 пикселов, поскольку значение `auto` свойства `flex-basis` применяет значение,

По умолчанию для свойства `flex-basis` выбирается значение `auto` — тогда для размера элемента применяются заданные для него значения свойства `width/height`. Если свойство для главного размера элемента (`width` или `height`) не установлено или установлено как `auto` (по умолчанию), свойство `flex-basis` ориентируется на ширину контента. Вы

установленное для свойства `width`. При этом элементам разрешено растягиваться, занимая свободное пространство контейнера, но не разрешается сжиматься:

```
box {  
    width: 100px;  
    flex: 1 0 auto;  
}
```

Когда браузер выбирает размер flex-элемента, он обращается к значению `flex-basis`, сравнивает его с доступным пространством вдоль оси, а затем добавляет или удаляет пространство из элемента в соответствии с настройками по растягиванию и сжатию (`grow` или `shrink`). Важно отметить, что, когда элемент подобным образом растягивается или сжимается, он может оказаться уже или шире, чем значение его длины, обеспечиваемое свойствами `flex-basis` или `width`.

О ПРИОРИТЕТЕ FLEX-НАСТРОЕК

Flex-настройки переопределяют заданные для flex-элементов значения ширины/высоты.

Удобные сокращенные flex-значения

Преимущество применения свойства `flex` состоит в том, что оно предусматривает несколько удобных сокращений, которые поддерживают типичные сценарии Flexbox. Любопытно, что некоторые из таких сокращенных значений переопределяют заданные по умолчанию значения для отдельных свойств, что поначалу немного смущает, но, в итоге, способствует более предсказуемому поведению веб-страницы. При создании flexbox-компонента посмотрите, может быть, вам поможет одна из следующих удобных настроек:

- `flex: initial;` — все равно, что `flex: 0 1 auto`. Препятствует растягиванию flex-элемента даже при наличии свободного пространства, но позволяет ему сжиматься для размещения в контейнере. Размер основан на заданных свойствах `width/height`, что, по умолчанию, соответствует размеру контента. Вместе со значением `initial` вы можете применять свойство `justify-content` для выравнивания по горизонтали;
- `flex: auto;` — все равно, что `flex: 1 1 auto`. Позволяет элементам стать полностью гибкими — увеличиваться или уменьшаться по мере необходимости. Размер основан на заданных свойствах `width/height`;
- `flex: none;` — эквивалентно указанию `flex: 0 0 auto`. Формирует полностью негибкий flex-элемент с размерами, основанными на свойствах `width` и `height`. Здесь также вместе со значением `none` для выравнивания можно применить свойство `justify-content`;
- `flex: целое число;` — аналогично указанию `flex: целое число 1 0px`. В результате получается flex-элемент с flex-размером, равным 0, что означает его перевод в абсолютное flex-значение (см. врезку «[Абсолютные и относительные flex-значения](#)»), и свободное пространство распределяется между элементами в соответствии с соотношениями их flex-значений.

ПОЛЬЗУЙТЕСЬ УДОБНЫМИ СОКРАЩЕНИЯМИ

При формировании flexbox-компонентов обратите внимание на преимущества удобных в использовании сокращенных flex-значений.

АБСОЛЮТНЫЕ И ОТНОСИТЕЛЬНЫЕ FLEX-ЗНАЧЕНИЯ

На рис. ЦВ-16.19 показано, каким образом свободное пространство распределяется между элементами с учетом соотношения их flex-значений. Речь при этом идет об *относительных flex-значениях* — именно таким образом свободное пространство пересчитывается всякий раз, когда свойство `flex-basis` устанавливается в любое значение, отличное от нуля (0), — такое, как некое значение `width/height` или `auto`.

Однако, если установить значение `flex-basis` в 0, произойдет нечто примечательное — элементы приобретут размеры, пропорциональные соотношениям своих flex-значений, которые известны как *абсолютные flex-значения*. То есть при `flex-basis: 0` элемент со значением свойства `flex-grow`, равным 2, *обязательно* будет вдвое шире, чем элемент, для которого этот параметр равен 1. Напомню еще раз, что такой подход реализуется, только если `flex-basis` установлен в 0.

На практике рекомендуется всегда после нуля (0) указывать единицу измерения, то есть записывать `0px` или, предпочтительно, `0%`.

В следующем примере абсолютного flex-значения для первого блока свойство `flex-grow` установлено в значение 2, а для четвертого блока свойство `flex-grow` установлено в значение 3 с помощью сокращенного свойства `flex`: целое число, речь о котором шла ранее. На рис. ЦВ-16.21 показан итоговый общий размер блоков, пропорциональный значениям `flex-grow`, поскольку свойство `flex-basis` установлено равным 0.

```
.box {
    /* применяется ко всем блокам */
    flex: 1 0 0%;
}
.box1 {
    flex: 2; /* сокращенное значение для flex: 2 1 0px */
}
.box4 {
    flex: 3; /* сокращенное значение для flex: 3 1 0px */
}
```

ВЗГЛЯД В БУДУЩЕЕ

В разд. «Стайлинг форм» главы 19 я применяю Flexbox для создания адаптивной формы. Свойства Flex позволяют полям формы приспосабливаться к доступной ширине, в то время как метки всегда остаются одинаковыми. А выравнивание позволяет полям формы на меньших экранах перемещаться ниже своих меток. Вы, сейчас, конечно заняты Flexbox, однако, может быть, имеет смысл немножко заглянуть в будущее.

Как вам этот материал о Flexbox? Естественно, трудно воспринять все сразу. Имеется одно свойство элемента Flexbox, с которым желательно познакомиться, перед тем как перейти к экспериментам.

Свойство `order` (изменение порядка следования flex-элементов)

Одной из примечательных особенностей Flexbox является возможность отображения элементов в порядке, отличном от порядка, в котором они записаны в исходном документе (см. врезку «Когда следует изменять порядок (и когда — нет)?»). Это

означает, что можно изменять порядок расположения элементов, используя только CSS, — у нас появился мощный инструмент для адаптивного дизайна, позволяющий переносить контент документа на меньшие экраны.

Для изменения порядка элементов примените свойство `order` к тем элементам, которые необходимо переместить:

`order`

- **Значения:** целое число.
- **По умолчанию** — 0.
- **Применение** — к flex-элементам и абсолютно позиционируемым дочерним элементам flex-контейнеров.
- **Наследование** — нет.

Значением свойства `order` является положительное или отрицательное число, с помощью которого оказывается влияние на размещение элемента вдоль flex-линии. Подобный подход аналогичен свойству `z-index` тем, что конкретное числовое значение не имеет значения — важно лишь, как оно связано с другими значениями.

По умолчанию у всех элементов устанавливается значение свойства `order`, равное нулю (0). Когда элементы имеют одинаковые значения `order`, они располагаются в том же порядке, в каком записаны в исходном HTML-коде. Если же им задать *различные* значения `order`, они располагаются от самого меньшего значения `order` к самому большему.

Возвращаясь к нашему примеру раскрашенного и пронумерованного блока, придалим элементу `box3` значение `order`, равное 1. Получив более высокое значение `order`, чему у остальных элементов, которые установлены по умолчанию в значение 0, он после них всех и отобразится (рис. ЦВ-16.22):

```
.box3 {  
    order: 1;  
}
```

КОГДА СЛЕДУЕТ ИЗМЕНЯТЬ ПОРЯДОК (И КОГДА — НЕТ)?

Имейте в виду, что, несмотря на кажущееся удобство, переупорядочивание является лишь визуальным отображением ловкости рук и должно применяться с осторожностью. Обратите внимание на некоторые моменты, о которых следует помнить:

- несмотря на то что визуальными браузерами элементы отобразятся в заданном свойством `order` порядке, альтернативные средства — такие, как программы чтения с экрана, могут, тем не менее, считывать контент в порядке его записи в источнике (хотя однозначно это утверждать тоже сложно);
- изменяйте исходный порядок, если для изменения порядка имеется логическая (а не визуальная) причина;
- не обращайтесь к свойству `order` только потому, что так удобнее;
- применяйте свойство `order`, если и логический, и визуальный порядки требуют такого переупорядочения.

Если несколько элементов имеют одно и то же значение `order`, они объединяются в группу (называемую *порядковой группой*) и отображаются в исходном порядке. Что произойдет, если элемент `box2` из предыдущего примера также получит значение `order`, равное 1? Тогда и `box2`, и `box3` будут иметь значение `order`, равное 1 (что делает их порядковой группой), и они отобразятся в исходном порядке, но после всех элементов с более низким значением `order`, равным 0 (рис. ЦВ-16.23):

```
.box2, .box3 {
    order: 1
}
```

Свойство `order` может принимать также и отрицательные значения. Продолжая наш пример, приадим элементу `box5` значение свойства `order`, равное -1. Обратите внимание на рис. ЦВ-16.24, где элемент `box5` не просто перемещается назад на одну позицию — он встает перед всеми элементами, которые имеют значение `order`, равное 0, что, как известно, является большей величиной по сравнению с -1:

```
.box5 {
    order: -1
}
```

В этих примерах я использовала простые величины значений, равные 1 и -1, но можно взять 10008 или -649 и результат будет таким же — влияние свойства `order` устанавливается от наименьшего его значения к наибольшему. При этом числовые значения не обязательно должны иметь последовательный порядок.

Посмотрим, каким образом можно применять свойство `order` для иных целей, нежели простое перемещение небольших блоков вдоль линии. В качестве следующего примера мы возьмем небольшой документ с верхним колонтитулом (`header`), основным разделом (`main`), куда входят элемент `article` и два элемента `aside`, а также нижним колонтитулом (`footer`):

```
<header>...</header>
<main>
    <article><h2>Where It's At</h2></article>
    <aside id="news"><h2>News</h2></aside>
    <aside id="contact"><h2>Contact</h2></aside>
</main>
<footer>...</footer>
```

В последующем коде CSS я сделала элемент `main` flexbox-контейнером, поэтому элементы `article` и `aside` выстраиваются в ряд, формируя три столбца (рис. 16.25). Для каждого элемента я установила значение `flex`, которое позволяет им расширяться и сжиматься, а ширину задала с помощью свойства `flex-basis`. Наконец, для указания порядка следования элементов я определила соответствующие значения свойства `order`. Обратите внимание, что раздел **Contact** теперь идет первым, хотя он и записан последним в исходном документе. И, в качестве дополнительного бонуса, все столбцы заполняют основной контейнер по высоте, несмотря на количество содержащегося в них контента:

```
main {
    display: flex;
}
```

```

article {
  flex: 1 1 50%;
  order: 2;
}
#news {
  flex: 1 1 25%;
  order: 3;
}
#contact {
  flex: 1 1 25%;
  order: 1;
}

```



Рис. 16.25. Создание многоколоночного макета с помощью Flexbox

FLEXBOX ИЛИ GRID LAYOUT?

Хотя полностраничный макет можно создать с помощью Flexbox, эту задачу лучше решать с помощью модуля Grid Layout, речь о котором пойдет далее. Следует, тем не менее, учесть, что Flexbox имеет лучшую поддержку со стороны браузеров по сравнению с модулем Grid Layout, поэтому Flexbox может служить для Grid Layout подходящим резервным вариантом. Flexbox лучше подходит для формирования на странице отдельных компонентов — таких, как панели навигации, карты описаний продуктов или каких-либо информационных блоков, которые желательно помещать вдоль линии.

На этом завершается наш тур по свойствам Flexbox! При выполнении *упражнения 16.3* мы задействуем некоторые свойства уровня элемента к меню известного вам бистро. По окончании работы над упражнением обратите внимание на советы по работе с различными браузерами, приведенные в следующем разделе.

УПРАЖНЕНИЕ 16.3. НАСТРОЙКА FLEX-КОНТЕЙНЕРА И ПРИМЕНЕНИЕ СВОЙСТВА *ORDER*

Меню нашего бистро выглядит неплохо, но можно добавить к нему несколько заключительных штрихов. Для этого откройте файл **flex-menu.html** в том виде, в каком оставили его после выполнения *упражнения 16.2*, и внесите в него рекомендованные далее изменения.

- Чтобы не оставлять внутри контейнера много пустого пространства, сделаем так, чтобы элементы заполнили доступное пространство. Поскольку желательно, чтобы элементы были полностью гибкими, можно присвоить значение `auto` свойству `flex` (что аналогично использованию свойства `flex: 1 1 auto;`). Для этого добавьте к правилу `section` следующее объявление для подключения поведения растягивания:

```

section {
  ...
  flex: auto;
}

```

2. А теперь самый последний штрих — добавим фотографии в верхнюю часть каждого пункта меню. Поскольку они являются flex-контейнерами, для перемещения их элементов можно применить свойство `order`. Поэтому выберите абзацы с именем класса `photo` и присвойте им значения меньшие, чем заданное по умолчанию значение 0 — тогда фотография отобразится в столбце первой (рис. 16.26):

```
.photo {
    order: -1;
}
```

Если вы хотите еще более улучшить результат, установите ширину элементов `img` равной 100% — тогда они всегда будут заполнять контейнер по ширине. Созданная нами веб-страница выглядит несколько размытой при отображении на большом экране, поэтому воспользуйтесь рассмотренной в главе 7 методикой адаптивного изображения.

Эта страница вряд ли является лучшей в мире веб-страницей, но она предоставила нам возможность опробовать ряд свойств Flexbox.

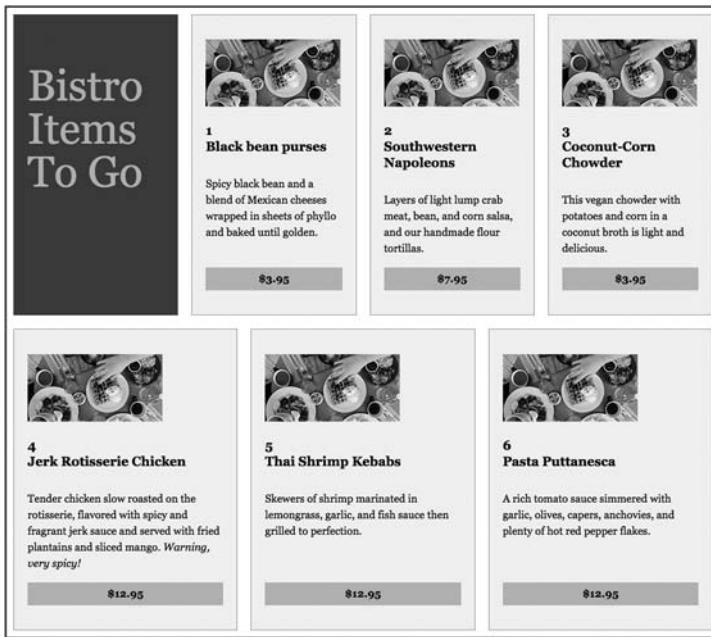


Рис. 16.26. Завершенное меню бистро с flex-элементами, заполняющими свободное пространство, и фотографиями, перемещенными в верхнюю часть каждого пункта

Браузеры, поддерживающие Flexbox

Текущая версия Flexible Box Layout Module (модуля макета гибкого контейнера) получила статус стабильной Рекомендации кандидата (Candidate Recommendation) в 2012-м году (www.w3.org/TR/css-flexbox-1/). Все основные настольные и мобильные браузеры поддерживают этот стандарт с 2015-го года, а некоторые — еще с 2013-го

года. Перечень таких браузеров (на момент подготовки этой книги) охватывает, согласно данным [CanIUse.com](#), примерно 80–90% пользователей.

Спецификация Flexbox претерпела на своем пути к стабилизации большое число изменений, и некоторые устаревшие браузеры реализовали в свое время эти устаревшие спецификации. Среди них три основных выпуска:

- *текущая версия (2012):*

Пример синтаксиса: `display: flex;`

Поддерживается следующими браузерами: IE11+, Edge 12+, Chrome 21–28 (`-webkit-`), Chrome 29+, Firefox 22–27 (`-moz-`, выравнивание отсутствует), Firefox 28+, Safari 6–8 (`-webkit-`), Safari 9+, Opera 17+, Android 4.4+, iOS 7–8.4 (`-webkit-`), iOS 9.2+

- *«промежуточная» версия (2011):*

Пример синтаксиса: `display: flexbox;`

Поддерживается: IE10

- *устаревшая версия (2009):*

Пример синтаксиса: `display: box;`

Поддерживается: Chrome <21, Safari 3.1–6, Firefox 2–21, iOS 3.2–6.1, Android 2.1–4.3

Вы не найдете в этих списках браузер Internet Explorer 9 и более ранние его версии, поскольку они вовсе не поддерживают Flexbox.

Для обеспечения поддержки Flexbox максимальным количеством браузеров необходимо сложный набор префиксов и альтернативных свойств, детали которых слишком сложны, чтобы в них углубляться прямо сейчас. Вряд ли вы захотите писать вручную такой код, однако, к счастью, есть варианты, обеспечивающие выход из этой ситуации.

Для автоматического генерирования сложного стека вы можете воспользоваться инструментом Autoprefixer. А в процессе изучения CSS и совершенствования своих практических навыков можно конвертировать стили прямо на сайте [autoprefixer.github.io](#) — надо лишь вставить там свои стили, и вы получите код, который можно добавлять в свою таблицу стилей (рис. 16.27).



Рис. 16.27. Сайт Autoprefixer преобразует стандартные стили Flexbox в стили, необходимые для полной поддержки всех возможных браузеров

Когда же вы сберетесь вывести свой рабочий процесс на профессиональный уровень, то сможете включить Autoprefixer в качестве «этапа сборки», что позволит вам автоматизировать большую часть процесса разработки. А если вы используете CSS-препроцессор, такой, как Sass, то сможете для управления всеми этими утомительными префиксами использовать и так называемые *миксины* (mixins). Впрочем, инструменты сборки и препроцессоры мы рассмотрим в главе 20.

ТЕСТИРУЙТЕ СГЕНЕРИРОВАННЫЕ ПРЕФИКСЫ НА ВСЕХ ЦЕЛЕВЫХ БРАУЗЕРАХ

Имейте в виду, что, хотя Autoprefixer облегчает добавление префиксов, бесперебойная работа контейнеров flexbox на всех браузерах не гарантируется. Всегда могут проявиться вполне непредсказуемые различия в их поведении, поэтому обязательно тестируйте сгенерированные Autoprefixer префиксы на всех целевых браузерах.

таблиц. Тогда можно применять метод обнаружения функций, позволяющий уточнить, поддерживает ли браузер контейнеры Flexbox. Если браузер не проходит тест, ему предоставляется запасной набор стилей, в то время как поддерживающие браузеры получают полный спектр возможностей Flexbox. Обнаружение функций мы рассмотрим в главе 19.

FLEXBUGS — ОШИБОЧНЫЕ РЕАЛИЗАЦИИ FLEXBOX

Существует несколько ошибочных реализаций Flexbox. К счастью для нас, Филипп Уолтон (Philip Walton) собрал все эти ошибки в репозитории GitHub под названием Flexbugs. Чтобы ознакомиться с ошибками и путями их обхода, посетите сайт: github.com/philipwalton/flexbugs.



Рис. 16.28. Эта очаровательная красная панда не имеет ничего общего с CSS-макетом, но мне кажется, что вам необходима передышка, прежде чем перейти к модулю Grid Layout. Фото Тери Финна (Teri Finn)

Итак, мы рассмотрели одну обширную методику верстки, и пора перейти к рассмотрению другой методики верстки — столь же обширной! Читатель, вы все еще со мной? К этому моменту нами изучено большое число мельчайших деталей, и, если вы хоть капельку похожи на меня, наверное, ваша голова уже идет кругом. Поэтому посмотрите на рис. 16.28. Здесь не идет речь о создании CSS-макетов — просто мне захотелось чуть-чуть развеяться. И, действительно, почему бы вам не отложить эту книгу и немножко не отвлечься, прежде чем приступить к изучению макетирования с помощью Grid?

МОДУЛЬ CSS GRID LAYOUT

Наконец-то у веб-дизайнеров и разработчиков появился CSS-модуль, использующий базовую сетку для формирования реального макета страницы, причем прошло всего лишь каких-то 25 лет! Модуль CSS Grid Layout предоставляет нам систему для размещения элементов в строках и (!) столбцах (вспомните, Flexbox размещает элементы только вдоль одной оси), причем элементы могут оставаться полностью гибкими и поддерживать соответствие с экранами разных размеров или имитировать макет печатной страницы. Сетки можно применять для создания таких типов макетов веб-страниц, которые уже известны на сегодня, или же изощряться с приемами типографики и использованием свободного пространства, как это делает Джен Симмонс (Jen Simmons) в своих Лабораторных демонстрациях (рис. 16.29). Сетку также можно применять и для оформления только части страницы — например, галереи изображений или продуктов.



Воссоздание печатного джазового постера с использованием сетки



Воссоздание приглашения на лекцию Die Neue Typographie (1927) с использованием сетки



Эксперимент с наложением фоторабот Доротеи Ланж

Рис. 16.29. Примеры разработок с применением сеток со страницы Джен Симмонс «Experimental Layout Lab» (labs.jensimmons.com)

В этом разделе я предоставлю вам полную информацию об использовании модуля Grid Layout. Тем не менее следует отметить, что у вас все равно могут остаться несколько недостаточно проясненных моментов, которые вы сможете исследовать самостоятельно.

Модуль Grid Layout — одна из наиболее сложных спецификаций в CSS, особенностям которой можно посвятить целую книгу. И эта книга — «Grid Layout in CSS» (издательство O'Reilly) — действительно написана Эриком Мейером (Eric Meyer). Именно с ее помощью я смогла понять дремучий язык самой спецификации (которая также доступна на сайте по адресу: www.w3.org/TR/css-grid-1/). Я также рекомендую книгу эксперта по Grid Рэчел Эндрю (Rachel Andrew) «The New CSS Layout», способную создать у читателя исчерпывающее представление о том, как создавать и применять сеточные макеты.

И ЕЩЕ О ДОПОЛНИТЕЛЬНЫХ ИСТОЧНИКАХ

Книга «CSS: The Definitive Guide», 4-е издание (издательство O'Reilly), написанная Эриком А. Мейером (Eric A. Meyer) и Эстель Вейль (Estelle Weyl), — это свод всех сведений о CSS. Она включает в качестве главы полное изложение возможностей CSS-модуля Grid Layout.

В Интернете также доступно большое число отличных Grid-ресурсов (речь о них пойдет в конце этого раздела).

Необходимое отступление по поводу браузерной поддержки

По поводу поддержки браузерами модуля Grid Layout имеются новости как хорошие, так и не очень. Хорошие новости состоят в том, что с марта 2017-го года браузеры Chrome 57+, Opera, Firefox 52+, Safari 10+ и iOS Safari 10+ начали поддер-

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

В браузерах Internet Explorer версий 10 и 11 и MS Edge (до версии 15) был реализован ранний проект модуля Grid Layout, большая часть которого уже устарела. Если идет речь о стандартных стилях сетки, описанных в этой главе, их следует рассматривать как стили, которые не поддерживаются упомянутыми браузерами. Однако, если эти браузеры Microsoft используются вашей целевой аудиторией, вероятно, стоит выбрать альтернативную версию макета, написанную с применением более старого синтаксиса сетки, который этими браузерами поддерживается.

не поддерживают возможности Grid-макетов, применяя Flexbox или устаревшие плавающие элементы (либо устаревшую спецификацию Grid для IE и Edge <15), что полностью зависит от браузеров, на которые вы ориентированы. Чтобы спокойно работать над макетами на основе Grid для браузеров, способных их обрабатывать, рекомендую воспользоваться возможностями инструмента CSS Feature Query, который проверяет поддержку Grid и предоставляет соответствующий набор стилей. Способы обнаружения функций подробно обсуждаются в главе 19.

Для получения обновленной информации о браузерной поддержке не забудьте посетить сайт CanIUse.com. Другим хорошим ресурсом может служить страница поддержки браузеров на сайте «Grid by Example», созданная Рэйчел Эндрю (Rachel Andrew) (gridbyexample.com/browsers), где она публикует новости о браузерной поддержке, а также описывает распространенные ошибки.

Основы работы с модулем Grid Layout

Процесс использования модуля CSS Grid Layout несложен:

1. **Воспользуйтесь свойством `display` для превращения элемента в grid-контейнер** — его дочерние элементы автоматически превратятся в grid-элементы.
2. **Настройте для сетки столбцы и строки** — вы можете задать их явно и/или указать, каким образом строки и столбцы должны формироваться «на лету».
3. **Присвойте каждому элементу сетки соответствующую область в сетке** — если не присвоить элементы явно, они разместятся по ячейкам сетки последовательно.

живать стандарт Grid — свободно и без браузерных префиксов. Поддержка Grid в 2017-м году была добавлена и в 16-ю версию браузера Microsoft Edge.

Не очень хорошая новость заключается в том, что помимо устаревших версий этих браузеров текущий стандарт Grid не поддерживает ни одна версия браузера Internet Explorer.

Таким образом, вам необходимо иметь возможность формировать альтернативные макеты для браузеров, которые

Сильно усложняет Grid Layout то, что спецификация предоставляет слишком много возможностей для определения каждой мелочи. Ими хорошо пользоваться в процессе профессиональной деятельности, но не когда вы впервые приступаете к изучению работы с сетками. В этой главе я подготовлю вас к работе с достаточно мощным набором инструментов Grid, который вы сможете дополнять самостоятельно по мере необходимости.

Терминология Grid

Прежде чем углубиться в изучение конкретных свойств, следует ознакомиться с основными составными частями и словарем Grid-системы.

Начиная с разметки, элемент, к которому применено свойство `display: grid`, становится *grid-контейнером* и определяет контекст для форматирования сетки. Все его прямые дочерние элементы автоматически становятся *grid-элементами*, которые, в конечном итоге, и оказываются в сетке. Если вы ознакомились с разд. «CSS Flexbox: создание гибких контейнеров» этой главы, то излагаемая здесь схема «помотки становятся элементами» вам знакома.

Ключевой фразой в предыдущем абзаце являются слова «прямые дочерние элементы», поскольку только эти элементы становятся grid-элементами. Это не относится к элементам, вложенным в эти элементы, поэтому их нельзя размещать в сетке. Однако можно вкладывать одну сетку в другую, если вы собираетесь создать сетку более глубокого уровня.

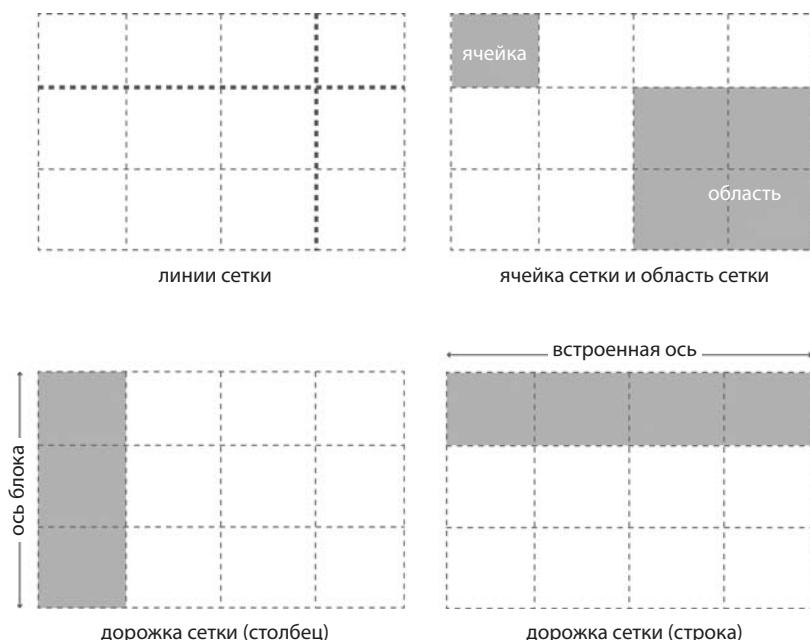


Рис. 16.30. Компоненты CSS сетки

Сетка сама по себе состоит из ряда компонентов (рис. 16.30):

- *линии сетки* — горизонтальная и вертикальная разделяющие линии сетки;
- *ячейка сетки* — наименьшая единица сетки, граничащая с четырьмя соседними линиями сетки и через которую не проходят какие-либо другие линии сетки;
- *область сетки* — прямоугольная область, образованная одной или большим количеством ячеек сетки;
- *дорожка сетки* — пространство между двумя соседними линиями сетки, представляющее собой *столбец сетки* или *строку сетки*. Столбцы сетки выстроены параллельно вертикальной оси блока (поскольку элементы блока расположены друг над другом) — для языков, в которых используется горизонтальное письмо. Строки сетки расположены параллельно строчной (горизонтальной) оси.

Следует отметить, что созданная для сетки структура не зависит от количества grid-элементов в контейнере. Можно поместить 4 grid-элемента в сетку из 12 ячеек, оставив 8 ячеек сетки в качестве свободного пространства. В этом вся прелесть сетки. Вы также можете сформировать сетку с количеством ячеек, меньшим, чем количество элементов сетки, — тогда браузер добавит в сетку ячейки для их размещения. Вот какая это удивительно гибкая система!

Ну а теперь без лишних церемоний перейдем к написанию кода.

Объявление отображения сетки

Для превращения элемента в grid-контейнер присвоим его свойству `display` значение `grid` или `inline-grid`.

ЗНАЧЕНИЕ `INLINE-GRID`

Линейные сетки (`inline-grid`) функционируют так же, как и блочные (`grid`), но могут применяться в потоке контента. В этом разделе рассматриваются только сетки блочного уровня.

Во время подготовки книги началась работа над проектом модуля макета CSS 2-го уровня (*Working Draft of CSS Grid Layout Module Level 2*), который предусматривает режим «подсетки», позволяющий вложенной сетке наследовать ее сеточную структуру от родительской сетки.

В следующем несложном примере элемент `#layout` `div` становится grid-контейнером, а каждый из его дочерних элементов (`#one`, `#two`, `#three`, `#four` и `#five`) — соответственно, grid-элементом:

Разметка

```
<div id="layout">
  <div id="one">One</div>
  <div id="two">Two</div>
  <div id="three">Three</div>
  <div id="four">Four</div>
  <div id="five">Five</div>
</div>
```

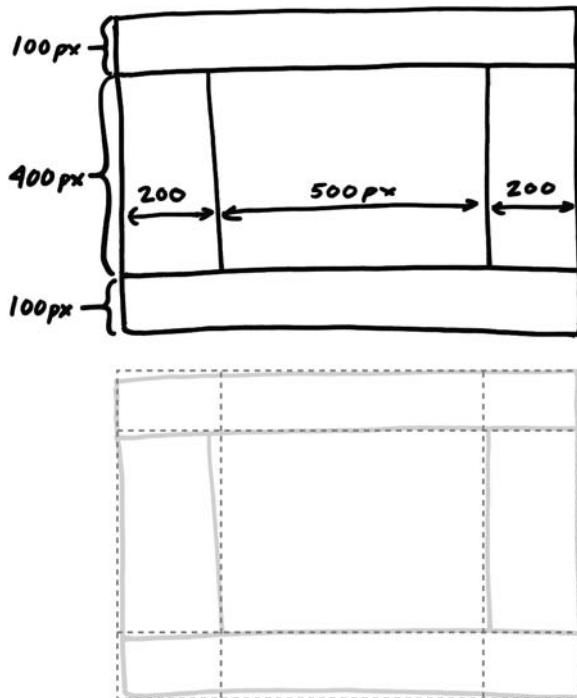
Правило стиля

```
#layout {
    display: grid;
}
```

Так формируется сцена (или, если использовать более точный термин, — *контекст*) для сетки. Теперь нам нужно задать ей необходимое количество строк и столбцов, а также их ширину.

Настройка сетки

Поскольку у меня нет никакого желания забивать себе голову вычислением ячеек и областей, я сделала быстрый эскизный набросок, дающий общее представление о том, какую окончательную сетку мне хотелось бы получить (рис. 16.31).



ЭТО ВАМ НИЧЕГО НЕ НАПОМИНАЕТ?

Возможно, вы заметили, что макет страницы с его верхним и нижним колонититулами и тремя колонками выглядит так же, как и созданный с помощью Flexbox (см. рис. 16.25). И это действительно так! Дело в том, что для достижения намеченного результата может быть использовано несколько решений. Как только модуль Grid Layout получит надежную поддержку, он займет главенствующую роль в создании подобных этому гибких макетов полноценных страниц.

Рис. 16.31. Весьма приблизительный эскиз макета страницы на основе сетки. Пунктирные линии на нижнем изображении показывают, сколько строк и столбцов необходимо сетке для формирования задуманной структуры макета

Эскиз — хороший первый шаг для работы с сетками. Из моего эскиза видно, что для макета нужны три дорожки-строки и три дорожки-столбца, хотя некоторые области контента охватывают более одной ячейки. Это вполне стандартная схема для веб-страницы, и хотя я включила в контент всего одно слово, чтобы сосредоточиться на структуре, здесь можно представить себе и более обширный текстовый контент, заполняющий каждую область сетки.

Определение дорожек сетки

Для настройки сетки с помощью CSS укажите высоту каждой строки (дорожки) и ширину каждого столбца с помощью шаблонов свойств `grid-template-rows` и `grid-template-columns`, которые применяются к элементу контейнера:

`grid-template-rows`
`grid-template-columns`

- **Значения:** `none` | список размеров дорожек и (опционально) названий линий.
- **По умолчанию** — `none`.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Значениями шаблона свойств `grid-template-row` является список *высот* для каждой дорожки-строки сетки. Значения шаблона свойств `grid-template-columns` представляют список значений *ширины* для каждой дорожки-столбца. Количество размеров дорожек определяет количество строк или столбцов. Например, если вы укажете четыре значения длины для шаблона свойств столбцов сетки (`grid-template-columns`), то и получите сетку, которая изначально разделена на четыре столбца.

ЕЩЕ О НАПРАВЛЕНИИ СТРОК И СТОЛБЦОВ

Как и в модуле Flexbox, в модуле Grid Layout направление строк и столбцов зависит от направления языка, на котором написана страница. Я использую в книге терминологию сетки для обозначения направления письма слева направо, сверху вниз.

Вы также можете задавать имена для линий сетки между дорожками, о чем речь пойдет далее, но а начнем мы с простых вещей.

Размеры дорожки сетки

В следующем примере я определила свойства шаблона для разделения контейнера `#layout` на три столбца и три строки, имеющие размеры, приведенные в исходном эскизе (см. рис. 16.31):

```
#layout {  
    display: grid;  
    grid-template-rows: 100px 400px 100px;  
    grid-template-columns: 200px 500px 200px;  
}
```

Давайте посмотрим, как эта сетка будет выглядеть в браузере. На рис. ЦВ-16.32 показано, что по умолчанию элементы сетки размещаются в доступных ячейках сетки по порядку следования. Я добавила для элементов сетки фоновые цвета, чтобы их границы четко выделялись, а для показа всей структуры сетки (*справа*) использовала инструмент Firefox CSS Grid Inspector.

Поскольку элемент `div #layout` имеет только пять дочерних элементов, соответственно заполняются лишь первые пять ячеек. Это автоматическое поведение потока, и хотя для нашего примера сетки подобный вариант заполнения не важен, но он

полезен для случаев последовательного добавления контента в сетку — например, при создании галереи изображений. Вскоре мы займемся целенаправленным размещением в сетке каждого нашего grid-элемента, но сначала рассмотрим значения шаблонов свойств.

Номера и названия линий сетки

Когда браузер формирует сетку, каждой линии сетки автоматически присваивается номер, и на него можно ссылаться при позиционировании элементов. Линия сетки в начале дорожки сетки равна 1, затем линии нумеруются последовательно. На рис. 16.33 показана нумерация линий для сетки нашего примера.



Рис. 16.33. Номера линиям сетки присваиваются автоматически

Кроме этого, линии, начиная с `-1`, нумеруются также и с конца дорожек, откуда и ведут обратный отсчет: `-2`, `-3` и т. д. (как показано «серыми» цифрами на рис. 16.33). Возможность нацеливаться на конец строки или столбца без подсчета строк (или даже представления о количестве строк или столбцов) весьма удобна, и вы оцените ее со временем.

Но, если вы не хотите отслеживать номера линий, можно назначать им имена, что интуитивно даже более понятно. В следующем примере я присваиваю линиям имена, которые соответствуют тому, как я стану использовать сетку на последней странице. Названия линий добавляются в шаблоны в квадратных скобках, размещаемых соответственно их отображению относительно дорожек:

```
#layout {  
    display: grid;  
    grid-template-rows: [header-start] 100px [content-start] 400px  
    [footer-start] 100px;  
    grid-template-columns: [ads] 200px [main] 500px [links] 200px;  
}
```

Исходя из приведенного примера, линия в верхней части сетки может называться «header-start», «1» или «`-4`». Вы могли бы также назвать линию после первой

дорожки «header-end», хотя она уже названа «content-start». Чтобы присвоить строке более одного имени, включите в скобки все нужные имена, разделив их пробелами:

```
grid-template-rows: [header-start] 100px [header-end content-start]  
400px [footer-start] 100px;
```

Обычно каждая линия сетки в конце концов несет несколько имен и номеров, и вы всегда можете выбрать, какие из них проще использовать. И мы тоже в подходящий момент воспользуемся этими цифрами и именами для размещения в сетке наших grid-элементов.

FIREFOX CSS GRID INSPECTOR И LAYOUT PANEL

Браузер Firefox 52+ содержит отличный инструмент разработчика под названием CSS Grid Inspector (Просмотр сетки CSS), который выводит представление о структуре элементов сетки, основываясь на значениях `grid` свойства `display`. Как раз такой подход использован для получения правого снимка экрана, показанного на рис. ЦВ-16.32. Для ознакомления с ним откройте Inspector, выполнив команды **Tools | Web Developer | Inspector** (Инструменты | Веб-разработка | Инспектор), найдите нужный grid-элемент и щелкните на ссылке `#icon`. После этого появится наложенная на страницу сетка.

Вы также можете выбрать вкладку **Layout** (Разметка) для получения доступа к панели разметки, где будут указаны все имеющиеся на странице grid-контейнеры и представлены инструменты для анализа линий и областей сетки. Там также имеется компонент, обеспечивающий доступ к свойствам блочной модели, благодаря чему можно просмотреть размеры, отступы, границы и поля для каждого связанного с сеткой элемента, а также многое другое. Подобные визуальные инструменты упрощают настройку проектов.

Совсем скоро подобные инструменты разработки макета сетки появятся в Chrome и Safari. Для дизайнеров сетки будущее выглядит прекрасно!

Указание значений размеров дорожки

Размеры дорожек в нашем примере представлены длиной, заданной в пикселях, что позволяет их легко визуализировать, но фиксированные размеры являются лишь

одним из многих вариантов. Да и уступает такой подход в гибкости, необходимой в нашем мире, переполненном самыми разными устройствами. Модуль Grid Layout предоставляет целый ряд возможностей для указания размеров дорожек, включая устаревшие подходы, связанные с указанием значений длин (например, в пикселях или единицах em) и процентных значений, а также некоторые более новые и специфичные для Grid значения. Далее я приведу краткие описания некоторых полезных значе-

ЗНАЧЕНИЯ РАЗМЕРОВ ДОРОЖКИ

Спецификация Grid поддерживает следующие значения для свойств `grid-template-*`:

- Длины (такие, как px или em)
- Процентные значения (%)
- Дробные единицы (fr)
- auto
- min-content, max-content
- minmax()
- fit-content()

ний, специфичных для Grid: это единица `fr`, функция `minmax()`, ключевое слово `auto` и значения, сформированные на основе контента: `min-content/max-content`. Мы также рассмотрим методы, позволяющие настраивать повторяющийся шаблон для ширины дорожек: функцию `repeat()` с опциональными значениями: `auto-fill` и `auto-fit`.

Дробные единицы (flex factor)

Специфичная для Grid дробная единица `fr` позволяет разработчикам так указать ширину дорожек, что они смогут расширяться и сжиматься в зависимости от доступного пространства. Возвратимся к нашему примеру и изменим средний столбец с `500px` на `1fr` — тогда браузер отдаст дорожкам этого столбца все оставшееся пространство (после размещения дорожек двух других столбцов размером по `200px`), как показано на рис. ЦВ-16.34:

```
#layout {  
    display: grid;  
    grid-template-rows: 100px 400px 100px;  
    grid-template-columns: 200px 1fr 200px;  
}
```

Единица `fr` отлично подходит для объединения фиксированной и гибкой ширин дорожек, однако вы можете также задать как `fr` все единицы, предоставляемые пропорциональную ширину всем столбцам.

В следующем примере все значения для ширины столбцов гибко реагируют на наличие доступной ширины браузера, но ширина среднего столбца всегда будет вдвое больше ширины боковых столбцов:

```
grid-template-columns: 1fr 2fr 1fr;
```

Минимальный и максимальный диапазон размеров

Диапазон размеров дорожки можно ограничить, устанавливая с помощью функции `minmax()` значения для их минимальной и максимальной ширины — вместо задания для дорожки определенного размера:

```
grid-template-columns: 200px minmax(15em, 45em) 200px;
```

Это правило устанавливает ограничения для среднего столбца, который может быть шириной, как минимум, `15em`, но никогда не станет шире, чем `45em`. Этот метод позволяет реализовать гибкость, но разрешает автору устанавливать границы.

Установка размеров на основе контента

Значения `min-content`, `max-content` и `auto` устанавливают размеры для дорожки, основываясь на ее контенте (рис. 16.35).

КАК БРАУЗЕР ЭТО ДЕЛАЕТ?

Технически браузер складывает единицы `fr` (в нашем примере: 4), делит оставшееся пространство на это количество частей, а затем назначает столбцам размеры, исходя из количества указанных для них единиц `fr`.

ОГРАНИЧЕНИЕ ЕДИНИЦЫ `fr`

Единицы `fr` не могут применяться в качестве минимального значения в инструкции `minmax()`.

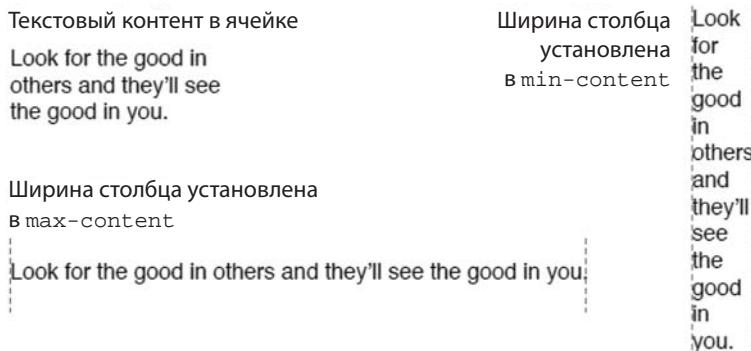


Рис. 16.35. Значения для размеров дорожки `min-content` и `max-content`

Значение `min-content` является *наименьшим*, которое может получить дорожка без переполнения (когда ее контент не может в нее поместиться), — это поведение по умолчанию, если не оно переопределено с помощью явного задания `min-width`. Такой подход эквивалентен заданию ширины «самого большого неразрывного контента» — другими словами, ширине самого длинного слова или наиболее широкого изображения, и вряд ли пригодится в том случае, если элементы содержат нормальные абзацы, но весьма полезен, когда нежелательно, чтобы дорожка увеличивалась по ширине. В следующем примере показаны три столбца, причем размер правого столбца достаточно широк, чтобы вместить самое длинное слово либо изображение:

```
grid-template-columns: 50px 1fr min-content;
```

Свойство `max-content` выделяет контексту максимальный необходимый ему объем пространства, даже если это приводит к расширению дорожки за пределы `grid`-контейнера. При задании этого свойства в качестве ширины столбца дорожка столбца получит такую же ширину, как и самый широкий контент в этой дорожке без переноса строк. Это означает, что если контент представляет собой абзац, дорожка окажется достаточно широка, чтобы вместить весь его текст в одну строку. Поэтому свойство `max-content` более подходит для коротких фраз или элементов навигации, когда нежелателен перенос текста (рассматриваемое далее ключевое слово `auto` может работать успешнее, позволяя при недостатке места выполнять перенос).

Использование ключевого слова `auto` для размера дорожки в основном похоже на передачу в браузер различных ключевых слов. В общем случае размер дорожки становится достаточно большим для размещения контента, принимая во внимание и другие имеющиеся ограничения.

НАЧНИТЕ С `auto`!

Если вы хотите определить размер дорожки в зависимости от ее контента, но не уверены, какое именно ключевое слово применить, начните с `auto`.

Для функции `minmax()` ключевое слово `auto` ведет себя аналогично значению `min-content` или `max-content` — в зависимости от того, помещено ли оно в минимальный или максимальный

слот. Как отдельное ключевое слово, оно функционирует аналогично `minmax` (`min-content`, `max-content`), позволяя сжимать дорожку настолько, насколько это возможно, без каких-либо переполнений, но и увеличивать так, чтобы при наличии достаточного места вмещать соответствующий контент без переноса.

В отличие от `max-content`, использование значения `auto` в качестве максимума позволяет свойствам `align-content` и `justify-content` растянуть дорожку за пределы размера контента. Если же `auto` применяется для задания минимума, оно обеспечивает более разумное поведение, чем при использовании свойства `min-content`, — например, при задании для элемента (или нескольких элементов) вместо их размера `min-content` заданного значения `min-width` или `min-height` и игнорировании содержимого любых `grid`-элементов с полосами прокрутки.

Если вы хотите определить размер дорожки в зависимости от ее контента, но не уверены, какое именно ключевое слово применить, начните с `auto`.

Повторение размеров дорожек

Предположим, у нас имеется сетка с 10 столбцами, ширина которых изменяется, например, так:

```
grid-template-columns: 20px 1fr 20px 1fr 20px 1fr  
20px 1fr 20px 1fr;
```

Зачастую пользователям лень набирать большие объемы текста (я сама нередко выступаю в роли такого пользователя). Для подобных случаев ребята из консорциума W3C предоставили средство ускоренного ввода текста, реализованное в виде функции `repeat()`. В предыдущем примере шаблон `20px 1fr` повторяется пять раз, и это можно записать следующим образом:

```
grid-template-columns: repeat(5, 20px 1fr);
```

Намного лучше, не так ли? Первое число указывает на количество повторений, а после запятой записан шаблон размеров дорожек. Функцию `repeat()` можно использовать и при более длинной последовательности размеров дорожек — например, если 10 наших столбцов расположены между двумя 200-пиксельными столбцами в начале и в конце:

```
grid-template-columns: 200px repeat(5, 20px 1fr) 200px;
```

Также можно указать имена линий сетки до и/или после каждого размера дорожки — тогда эти имена тоже станут повторяться в шаблоне:

```
grid-template-rows: repeat(4, [date] 5em [event] 1fr);
```

Значения `auto-fill` и `auto-fit`

В предыдущих примерах, связанных с функцией `repeat()`, браузеру указывалось, сколько раз надо повторить предоставленный шаблон. Впрочем, можно позволить браузеру самому разобраться в количестве повторов — в зависимости от доступного пространства, используя в функции `repeat()` вместо указания целого числа значений `auto-fill` и `auto-fit`.

Например, если указать:

```
grid-template-rows: repeat(auto-fill, 10em);
```

а grid-контейнер сетки имеет высоту, равную 35 em, то браузер станет формировать строки каждые 10 em, до тех пор пока имеется свободное пространство, и в результате мы получим три строки. При этом, даже если контента хватает только для заполнения первой строки, сформируются все три строки, и пространство сохранится в макете.

РАЗЛИЧНОГО РОДА ОГРАНИЧЕНИЯ...

Для одного объявления можно применять только одно свойство auto-repeat, и его нельзя использовать с единицами fr. Также нельзя указывать «размерные» ключевые слова content-based внутри записей auto-fill или auto-repeat. Обратите внимание, что запись minmax() можно использовать внутри auto-repeat, и ее разрешается применять с единицами fr или ключевыми словами content-based (auto, min-content, max-content), если вы находитесь в позиции, заданной словом «так», когда применяется длина, заданная словом «min».

Значение auto-fit оказывает аналогичное воздействие, за исключением того, что из макета удаляются любые дорожки, для которых отсутствует контент. Если остается место, оно распределяется в соответствии с предоставленными значениями вертикального (align-content) и горизонтального (justify-content) выравнивания (в дальнейшем выравнивание будет рассмотрено подробнее).

Определение областей сетки

До сих пор речь шла о разделении grid-контейнера на дорожки строк и столбцов с помощью свойств grid-template-columns и grid-template-rows, рассмотрены также многие из возможных значений, определяющих размеры дорожек. Кроме того, мы узнали, что можно назначать имена отдельным линиям сетки, чтобы на них было проще ссылаться при размещении в сетке grid-элементов.

Назначать имена можно и областям сетки, что некоторые разработчики считают более интуитивной возможностью, чем вызов конкретных строк. Вспомним, что область сетки состоит из одной или нескольких ячеек, составляющих прямоугольник (не применяются L-формы или другие непрямоугольные блоки ячеек). Обозначение областей сетки реализовать сложнее, но оно обеспечивает удобные сокращения.

Назначение имен областям сетки реализуется с помощью свойства grid-template-area:

grid-template-areas

- **Значения:** none | набор имен областей сетки.
- **По умолчанию** — none.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Значение свойства представляет собой список имен для каждой ячейки в сетке, приведенных строка за строкой, причем каждая строка заключена в кавычки. Если

соседние ячейки имеют общее имя, они образуют область сетки с этим именем (см. врезку «Бонусы присвоения имен»).

В следующем примере я присваиваю названия областям примера сетки, с которым мы уже встречались (рис. 16.36). Обратите внимание, что у каждой из девяти ячеек имеется имя, отображаемое в каждой строке. Списки ячеек строки не обязательно выстраиваются так, как это сделано здесь, но многие разработчики предпочитают называть ячейки по именам, используя символические названия для лучшей визуализации структуры сетки:

```
#layout {
    display: grid;
    grid-template-rows:
    [header-start] 100px
    [content-start] 400px
    [footer-start] 100px;
    grid-template-columns:
    [ads] 200px [main] 1fr
    [links] 200px;
    grid-template-areas:
    "header header header"
    "ads main links"
    "footer footer footer";
}
```



Рис. 16.36. Если соседние ячейки имеют одинаковые имена, они образуют именованную область, на которую можно потом ссылаться

Если в сетке имеются три столбца, для каждой их строк должна быть указано три имени. Если необходимо оставить ячейку без имени, введите на месте, где могло быть записано ее название, одну или несколько точек (.), чтобы не пропустить такую ячейку вовсе. Напомню, что эскиз сетки с идентифицированными областями облегчит планирование значения `grid-template-areas`.

БОНУСЫ ПРИСВОЕНИЯ ИМЕН

Называя область с помощью `grid-template-areas`, вы, в качестве дополнительного бонуса, получаете для работы набор автоматически генерированных имен линий сетки. Например, если вы называете область «`main`», левая и верхняя линии сетки этой области автоматически называются «`main-start`», а правая и нижняя линии сетки — «`main-end`». Полученные таким образом имена линий можно использовать при позиционировании элементов.

Обратное также верно. Если окружающим область линиям явно назначить имена «`portal-start`» и «`portal-end`», то позднее можно будет использовать имя области «`portal`» для назначения контента этой области, даже если оно не определено с помощью свойства `grid-template-areas`. Вы можете запомнить такое сокращение, возникшее при именовании линий сетки, что, впрочем, не обязательно.

Это яркий пример гибкости и сложности модуля Grid Layout.

ОБЛАСТИ ЯЧЕЕК ДОЛЖНЫ ИМЕТЬ ПРЯМОУГОЛЬНУЮ ФОРМУ!

Убедитесь, что вы размещаете имена ячеек так, чтобы они при объединении в область образовывали прямоугольники, — это позволит идентифицировать именованную область. L-формы и прочие фрагменты не применяются.

Имейте также в виду, что размеры дорожек зависят от свойств `grid-template-columns` и `grid-template-rows`. Свойство `grid-template-areas` просто присваивает областям имена, облегчая в дальнейшем добавление к ним элементов.

Сокращенное свойство `grid`

Применяйте сокращенное свойство `grid` для задания значений `grid-template-rows`, `grid-template-columns` и `grid-template-areas` с помощью одного стилевого правила. Помните, что любые не используемые в этом правиле свойства будут сброшены к заданным по умолчанию значениям, как и в случае применения всех сокращений:

`grid`

- **Значения:** `none` | *информация о строке/информация о столбце*.
- **По умолчанию** — `none`.
- **Применение** — к `grid`-контейнерам.
- **Наследование** — нет.

При обращении к свойству `grid` значения строк и столбцов разделяются слэшами (косыми чертами), причем значения для строк отображаются первыми:

`grid: строки/столбцы`

Это легче понять без неразберихи, вносимой названиями линий и областей, поэтому далее приводится краткое объявление примера сетки, содержащее только информацию о дорожках строк и столбцов:

```
#layout {  
    display: grid;  
    grid: 100px 400px 100px / 200px 1fr 200px;  
}
```

Для подключения пользовательских имен строк добавьте эти имена в скобках у соответствующих дорожек, как было показано ранее в примере со строкой имен.

Поначалу включение имен областей воспринимается непросто, но, если помнить, что имена ячеек указываются строка за строкой, имеет смысл отображать их перед слэшем вместе с иной информацией о строке. Полный порядок в этом случае выглядит следующим образом:

[имя начальной линии] «название области» <размер дорожки> [имя конечной линии]

Имена линий и названия областей необязательны, то есть опциональны. Повторяйте эту процедуру для каждой строки в сетке, просто указывая их одну за другой без специальных разделяющих строки символов. Возможно, вы сочтете полезным сложить их так, как я сделала в следующем примере, чтобы показать каждую строку

отдельно. Завершив компоновку строк, добавьте слэш и приведите после него информацию о дорожке столбца. Далее приведен полный пример такой сетки, написанной с помощью сокращенного свойства `grid`:

```
#layout {  
    display: grid;  
    grid:  
        [header-start] "header header header" 100px  
        [content-start] "ads main links" 400px  
        [footer-start] "footer footer footer" 100px  
    / [ads] 200px [main] 1fr [links] 200px; }
```

В несокращенном варианте этот пример выглядит так:

```
#layout {  
    display: grid;  
    grid-template-rows: [header-start] 100px [content-start]  
400px  
    [footer-start] 100px;  
    grid-template-columns: [ads] 200px [main] 1fr [links] 200px;  
    grid-template-areas:  
        "header header header"  
        "ads main links"  
        "footer footer footer" }
```

Имеется также свойство `grid-template`, которое функционирует так же, как `grid`, но может применяться лишь с явно определенными сетками (в отличие от неявных сеток, о которых речь пойдет позже). Спецификация модуля Grid Layout настоятельно рекомендует применять сокращенное свойство `grid`, а не свойство `grid-template`, если вы не заинтересованы в использовании каскадного поведения свойства `grid-template`.

Полагаю, что для вас настало время применить те стили настройки сетки, с которыми вы уже познакомились, выполнив упражнение 16.4.

ПРИДЕРЖИВАЙТЕСЬ ОТДЕЛЬНЫХ СВОЙСТВ ДЛЯ СЛОЖНЫХ СТРУКТУР

Эксперты `Grid` не склонны применять свойства `grid` или `grid-template`, кроме как для самых простых структур сетки. Код с их использованием становится слишком сложным, и одно небольшое изменение может привести к развалу всей сетки. Для сложных структур сетки при определении строк, столбцов и областей придерживайтесь их отдельных свойств.

ЕЩЕ О ПОДДЕРЖКЕ БРАУЗЕРАМИ

Для этого упражнения необходимо использовать браузер, который поддерживает сетки. Я применяю Firefox, дающий возможность задействовать инструмент `Grid Inspector`. Поддерживающие работу с сетками браузеры описаны ранее в разд. «Необходимое отступление по поводу браузерной поддержки». Смотрите также инструкции по открытию инструмента `Grid Inspector` на врезке «`Firefox CSS Grid Inspector u Layout Panel`».

УПРАЖНЕНИЕ 16.4. НАСТРОЙКА СЕТКИ

При выполнении этого упражнения мы настроим шаблон сетки для страницы, показанной на рис. 16.37. Grid-элементы в эту сетку мы поместим при выполнении упражнения 16.5, так что сейчас просто обратите внимание на установку значений для строк и столбцов.

Страница, с которой мы начинаем работу, похожа на страницу пекарни (*bakery*), знакомой нам по предыдущим упражнениям, но содержит еще много элементов и свободного пространства, нуждающихся в упорядочении. Начальный документ — `grid.html` доступен вместе с учебными материалами к этой книге на сайте по адресу: learningwebdesign.com/5e/materials. Откройте его в текстовом редакторе, и вы увидите, что в нем представлены все стили, влияющие на внешний вид элементов.



Рис. 16.37. Страница *Breads of the World*, которую мы отформатируем с помощью модуля Grid Layout

- Начните с превращения содержащего элемента: `#layout div` в grid-контейнер, присвоив режиму `display` значение `grid`:

```
#layout {
  ...
  display: grid;
}
```

- На рис. 16.37 показаны дорожки строк и столбцов, необходимые для размещения контента при нужном макете. Начните с определения указанных в эскизе строк, применяя свойство `grid-template-rows`. У вас должно получиться шесть значений, представляющих каждую из шести строк (настройка этих значений выполняется в следующем упражнении, а здесь это только отправная точка для него):

```
#layout {
  ...
  display: grid;
  grid-template-rows: 3em 20px 150px 300px 5em;
```

- Выполните то же самое для семи столбцов. Поскольку мне желательно, чтобы текстовый столбец увеличивался и уменьшался с учетом доступного пространства, его ширина указана в дробных единицах (`1fr`). Оставшиеся столбцы формируют ячейки шириной `150px` для трех изображений и `20px` пространства перед ними.

Вы можете сделать это следующим образом:

```
grid-template-columns: 1fr 20px 150px 20px 150px 20px 150px;
```

Но поскольку последние шесть столбцов представляют собой повторяющийся шаблон, проще использовать функцию `repeat()` для трехкратного повторения свободного пространства и столбцов:

```
grid-template-columns: 1fr repeat(3, 20px 150px);
```

4. Наконец, присвоим имена линиям, ограничивающим область сетки, где должен отображаться элемент `main` контента. Эти имена предоставят нам некоторые интуитивно понятные варианты размещения указанного элемента позднее. Основная область (область `main`) начинается с дорожки третьей строки, поэтому присвойте имя «`main-start`» линии сетки, находящейся между размерами для второй и третьей строк дорожки:

```
grid-template-rows: 3em 20px [main-start] 150px 300px 5em;
```

Основная область продолжается до последней строки дорожки, поэтому присвойте имя «main-end» последней строке сетки (после дорожки последней строки):

```
grid-template-rows: 3em 20px [main-start] 150px 300px  
5em [main-end];
```

5. Теперь сделайте то же самое для линий сетки, которые отмечают границы дорожки столбцов, куда направляется основной контент:

```
grid-template-columns: [main-start] 1fr [main-end]  
repeat(3, 20px 150px);
```

Завершив работу, я сохранила ее и открыла документ в Firefox с помощью инструмента Grid Inspector (рис. 16.38). Поскольку я не указала, куда должны загружаться grid-элементы, они размещались в ячейках последовательно, создавая беспорядок, который на рис. 16.38 хорошо заметен. Однако, благодаря наложению сетки, видно, что ее структура скомпонована верно. Сохраните этот файл для выполнения следующего упражнения.

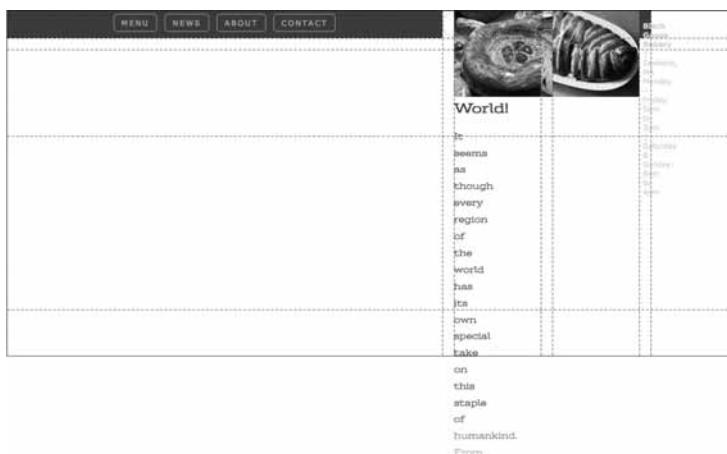


Рис. 16.38. Элементы сетки пока еще размещены некорректно, но Firefox Grid Inspector показывает, что сетка настроена правильно

Размещение grid-элементов

Теперь, когда мы рассмотрели все тонкости настройки сетки, включая присвоение осмысленных названий ее линиям и областям, пора перейти к назначению элементов областям сетки.

Как мы заметили на рис. ЦВ-16.32 и 16.38, при отсутствии каких-либо явных предписаний grid-элементы сетки, тем не менее, последовательно размещаются в доступных ячейках сетки. Это удобно для некоторых случаев, но давайте узнаем, как можно указать grid-элементам, куда им идти!

Позиционирование с использованием линий

Один из способов задания местоположения grid-элемента в сетке состоит в указании четырех линий, граничащих с целевой областью сетки, с помощью четырех свойств, которые определяют начальную и конечную линии строк, а также начальную и конечную линии столбцов. Применяются эти свойства к конкретному grid-элементу, который необходимо позиционировать:

```
grid-row-start  
grid-row-end  
grid-column-start  
grid-column-end
```

- **Значения:** `auto` | линия сетки | `span количество` | `span 'имя линии'` | `число 'имя линии'`
- **По умолчанию** — `auto`.
- **Применение** — к grid-элементам.
- **Наследование** — нет.

Представленный здесь набор свойств поддерживает простой способ описания положения элемента в сетке — путем определения либо имени, либо номера линии сетки на каждой из границ. В качестве альтернативы можно задать только идентификатор строки и указать элементу на необходимость «охватить» определенного количества ячеек. По умолчанию элемент занимает одну ширину дорожки, которая получена с помощью ключевого слова `auto`.

Возвратимся к нашему примеру из пяти элементов — теперь я хочу, чтобы первый элемент находился в верхней строке и распространялся на все три столбца (рис. 16.39).

Один из способов реализации этого варианта — следующим образом применить свойства `start` (начало) и `end` (конец) для четырех линий, идентифицируя линии по их номерам:

```
#one {  
    grid-row-start: 1;  
    grid-row-end: 2;  
    grid-column-start: 1;  
    grid-column-end: 4;  
}
```

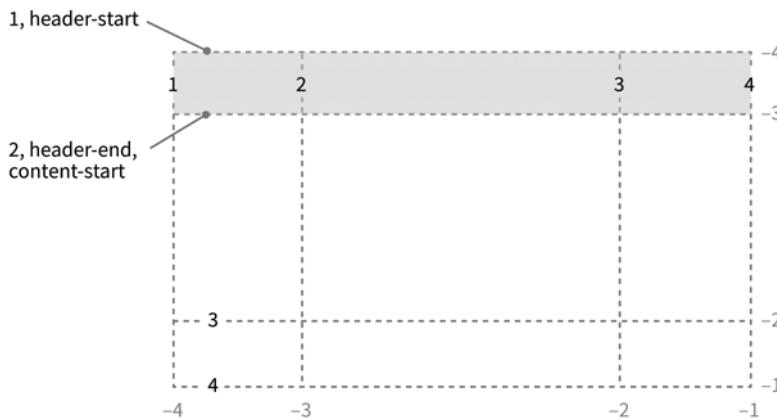


Рис. 16.39. Позиционирование grid-элемента вдоль верхней дорожки строки

Сравните эту ситуацию с позицией `#one div`, показанной ранее на рис. 16.36. Для свойства `grid-row-start` значение 1 ведет к ссылке на первую (верхнюю) линию grid-контейнера. Для свойства `grid-column-start` значение 1 ведет к ссылке на первую линию с левого края контейнера, значение 4 для свойства `grid-column-end` идентифицирует четвертую и последнюю линию с правого края контейнера.

Рассмотрим еще один пример. Следующее объявление стиля размещает элемент `#four` в правом столбце, как показано на рис. 16.36:

```
#four {
    grid-row-start: 2;
    grid-row-end: 3;
    grid-column-start: 3;
    grid-column-end: 4;
}
```

Вы помните, что линии сетки также нумеруются в противоположном направлении, начиная с `-1`? И этим обстоятельством можно воспользоваться, присвоив свойству `grid-column-end` для `#one` значение `-1`, что то же самое, что и `4`. Преимущество этого подхода в том, что при его применении гарантированно охвачен конец дорожки без учета каких бы то ни было ошибок в подсчетах.

ИСПОЛЬЗУЙТЕ ЗНАЧЕНИЕ `-1`

Если нужно охватить последнюю линию сетки в строке или столбце, используйте значение `-1`, избавив себя от каких бы то ни было подсчетов. Кроме того, даже если число строк или столбцов вдоль линии изменяется, при указании значения `-1` всегда выбирается последняя строка, поэтому вам нет необходимости что-либо перенумеровывать.

Можно воспользоваться и заданными нами именованными строками. Следующие значения строк взаимозаменяемы с предыдущим примером:

```
#one {
    grid-row-start: header-start;
    grid-row-end: header-end;
    ...
}
```

Если опустить объявление конца строки, строка будет иметь высоту, равную одной дорожке (по умолчанию). А этого как раз я и хотела добиться, поэтому полное отсутствие объявления `end` (конец) — еще один способ получения желаемого эффекта.

ЕЩЕ ОДНА АЛЬТЕРНАТИВА...

Если опустить начальную или конечную строку, область будет иметь ширину, равную одной дорожке (по умолчанию — `auto`).

сколько дорожек нужно охватить. В следующем примере элемент начинается с левого края дорожки (линия 1) и охватывает три столбца, фактически заканчиваясь на линии 4:

```
#one {
  ...
  grid-column-start: 1;
  grid-column-end: span 3;
}
```

Охват может функционировать и в обратном направлении. Если задать только конечную строку, охват распространится до начальной дорожки. Следующие стили приводят к тому же эффекту, что и предыдущие примеры, поскольку определяют целевую область по ее конечной линии в крайнем правом углу сетки и охватывают три столбца по направлению к началу:

```
#one {
  ...
  grid-column-start: span 3;
  grid-column-end: -1;
}
```

Если четырех объявлений для одного эффекта кажется вам многовато, воспользуйтесь сокращенными свойствами `grid-row` и `grid-column`:

grid-row
grid-column

- **Значения:** начальная линия/конечная линия.
- **По умолчанию** — см. отдельные свойства.
- **Применение** — к grid-элементам.
- **Наследование** — нет.

Эти свойства объединяют свойства `*-start` и `*-end` в одно объявление. Начальная и конечная линии при этом отделяются друг от друга слешем (/). Любой из методов, предназначенных для организации ссылок на строки, работает с этими сокращенными значениями. С помощью сокращенных свойств можно вместить наш пример в следующие два объявления:

```
#one {
  grid-row: 1 / 2;
  grid-column: 1 / span 3;
}
```

Готовы ли к рассмотрению еще одного варианта? Можно указать элементу, с какой линии начинать, но вместо задания конечной линии использовать ключевое слово `span`, что позволит указать,

Позиционирование области

Другой способ позиционирования элемента в сетке — это велеть ему перейти в одну из заданных областей с помощью свойства `grid-area`:

`grid-area`

- **Значения:** название области | от 1 до 4 идентификаторов линий.
- **По умолчанию** — см. отдельные свойства.
- **Применение** — к grid-элементам.
- **Наследование** — нет.

Свойство `grid-area` указывает на одну из областей, именованных с помощью свойства `grid-template-areas`. Оно также может указывать на имя области, которое неявно создается, когда вы именуете строки, ограничивающие область, пользуясь суффиксами «`-start`» и «`-end`». С помощью этого свойства все элементы сетки можно поместить в области, сформированные нами ранее на основе исходного эскиза (рис. 16.40):

```
#one { grid-area: header; }
#two { grid-area: ads; }
#three { grid-area: main; }
#four { grid-area: links; }
#five { grid-area: footer; }
```

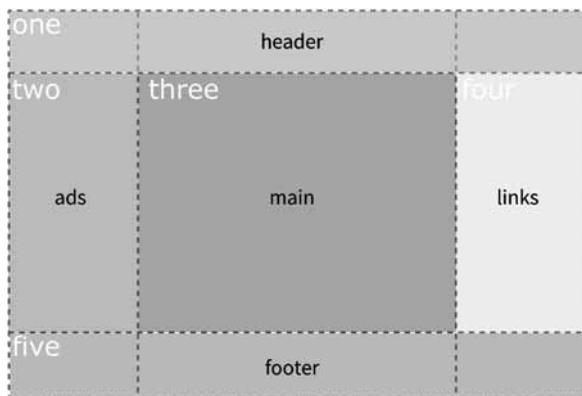


Рис. 16.40. Размещение grid-элементов с помощью имен областей

Не правда ли, это было сделать легко?! Одним из преимуществ использования областей является возможность изменить сетку, но, если нужные области сетки будут иметь те же имена, элементы окажутся в требуемом месте и вам не понадобится перенумеровывать строки в таблице стилей.

Вы также можете применять свойство `grid-area` для предоставления определяющего область из четырех разделенных слешами линий сетки. Порядок их отображения: `<row-start>`, `<column-start>`, `<row-end>`, `<column-end>` (против часовой стрелки, начиная сверху). Существует множество правил того, что произойдет, если опустить какие-либо из этих значений, но сейчас не время вдаваться во все эти

тонкости. Объявление `grid-area` для первого grid-элемента можно записать таким образом, что получится тот же результат, что и в предыдущих примерах:

```
#one {
    grid-area: 1 / 1 / 2 / span 3;
    /* row-start / column-start / row-end / column-end */
}
```

Как вы могли понять, модуль Grid Layout предоставляет большое количество вариантов настройки сетки и различных способов размещения на ней элементов. Кроме всего прочего, спецификация включает еще несколько вариантов использования ключевого слова `span`, которые вы можете изучить самостоятельно. Подберите себе методы, лучше подходящие для создаваемой сетки или более вам близкие.

А сейчас давайте закончим «доводку» сетки, с которой начинали работу ранее, и выполним *упражнение 16.5*.

УПРАЖНЕНИЕ 16.5. РАЗМЕЩЕНИЕ ЭЛЕМЕНТОВ НА СЕТКЕ

У нас уже имеется сетка, настроенная для страницы «Breads of the World», и мы можем начать размещение элементов в нужные области сетки, используя номера линий и их имена.

Я собираюсь быстро к этому перейти, но рекомендую на каждом этапе обработки сохранять файл и просматривать страницу в браузере, поддерживающем сетку. На рис. 16.41 показан вид готового макета, и вы можете уточнить с его помощью информацию о конечных позициях и номерах линий.

1. Откройте файл `grid.html` в текстовом редакторе, если он еще не открыт. Мы начнем с размещения элемента `nav` в первой строке сетки, используя четыре свойства линии сетки:

```
nav {
    grid-row-start: 1;
    grid-row-end: 2;
    grid-column-start: 1;
    grid-column-end: 8; /* вы можете использовать и -1 */
}
```

2. Теперь разместите рисунки в нужных местах сетки. Начните с размещения третьего рисунка (`#figC`) на его месте в дальнем правом столбце, применяя сокращенные свойства `grid-row` и `grid-column`. Он должен занять место между строчными линиями сетки 3-го и 4-го рядов и 7-й и 8-й линиями столбцов. Вместо указания номеров линий 7 и 8 столбцов используйте отрицательное значение последней линии, сместив его (`span`) на один пробел влево, чтобы получить начальную точку:

```
#figC {
    grid-row: 3 / 4;
    grid-column: span 1 / -1;
}
```

Затем позиционируйте элементы `#figA` и `#figB` с помощью свойства `grid-area`

и значений для линий. Напомню, что значения следуют в порядке сверху, слева, вниз и вправо (против часовой стрелки вокруг области):

```
#figA {
    grid-area: 3 / 3 / 4 / 4;
}
#figB {
    grid-area: 3 / 5 / 4 / 6;
}
```

- Итак, мы получили линии сетки вокруг области с именем `main`, и теперь используем их для размещения grid-элемента `main`:

```
main {
    grid-row: main-start / main-end;
    grid-column: main-start / main-end;
}
```

Не забыли ли вы, что при задании линиям вокруг области имен вида `*-start` и `*-end` формируется явно именованная область `*`? Наши линии именованы на основе этого синтаксиса, поэтому мы также можем разместить элемент `main` с помощью свойства `grid-area`:

```
main {
    grid-area: main;
}
```

- Наконец, поставим на место нижний колонтитул. Он начинается с последней строки сетки и охватывает в направлении назад одну дорожку. Для столбцов он начинается с третьей линии и продолжается до последней. Я привожу здесь один из способов написать эти инструкции. Можете ли вы придумать другие, приводящие к такому же результату?

```
footer {
    grid-row: 5 / 6;
    grid-column: 3 / -1;
}
```



Рис. 16.41. Готовый grid-макет веб-страницы «Breads of the World»

Сохраните файл и просмотрите страницу в браузере. У вас может возникнуть проблема, связанная с шириной окна браузера. Если окно браузера широкое, макет функционирует нормально, но, когда окно браузера становится уже, текст в элементе `main` переполняет ячейку. Дело в том, что высоты, равной 300 пикселам, заданной для этой строки, недостаточно для удержания текста, когда он разбивается на дополнительные строки или имеет больший размер.

5. Мы можем исправить это, изменяя размер дорожки пятой строки на `auto`. Тогда высота этой строки всегда будет, по крайней мере, достаточно большой, чтобы вместить контент. Значение `min-content` здесь также можно использовать, но сначала опробуйте значение `auto`:

```
#layout {  
    display: grid;  
    grid-template-rows: 3em 20px  
    [main-start] 150px auto 5em  
    [main-end];  
  
    ...  
}
```

Теперь, когда вы перезагрузите эту страницу в окне браузера, текст всегда будет находиться в области сетки независимо от ширины окна. То есть все должно оказаться на своих местах, как показано на рис. 16.41.

Итак, вы стали автором первой сетки! И, хотя это упражнение лишь дало вам почувствовать вкус возможностей модуля Grid Layout, вы, тем не менее, познакомились с основами настройки сетки и размещения в ней элементов. Отличное начало!

С основами создания явной сетки и размещения на ней элементов вы познакомились. Но есть еще несколько тем, связанных с сеткой, которые важно знать: неявные сетки, пространство между компонентами сетки («желоба») и выравнивание элементов сетки и сетки в целом. К сожалению, у меня здесь есть место только для базового введения в каждую из этих тем, но когда вы начнете создавать макеты на основе сетки самостоятельно, вы сможете совершить в них более «глубокое погружение», необходимое для удовлетворения ваших творческих потребностей.

Неявное поведение сетки

До сих пор у нас шла речь о способах создания явной сетки и осознанного размещения на ней элементов. Но уже на этом пути нам встретились некоторые автоматические, или *неявные*, аспекты поведения системы Grid. Например, без каких-либо явных инструкций grid-элементы размещаются в сетке последовательно, как показано на рис. ЦВ-16.32. Я также обращала ваше внимание на то, что при формировании именованной области неявно создаются линии сетки с суффиксами «`-start`» и «`-end`», и наоборот.

Другое неявное поведение сетки состоит в создании для размещения элементов дорожек в строках и столбцах «на лету», если они не помещаются в заданную сетку. Например, если вы поместите элемент за пределы заданной сетки, браузер автоматически создаст в сетке дорожки для его размещения. Точно так же, если имеется

больше элементов, чем ячеек или областей, браузер генерирует больше дорожек, пока все элементы не поместятся в этой области.

По умолчанию любая строка или столбец, автоматически добавляемые в сетку, имеют размер `auto`, причем размер этот достаточно велик, чтобы соответствовать высоте или ширине контента. Если вы хотите придать неявным строкам и столбцам определенные размеры — например, чтобы они соответствовали масштабу, установленному в другом месте сетки, применяйте свойства `grid-auto-*`:

`grid-auto-rows` `grid-auto-columns`

- **Значения:** список размеров дорожек.
- **По умолчанию** — `auto`.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Свойства `grid-auto-rows` и `grid-auto-columns` предоставляют один или несколько размеров для автоматически сгенерированных дорожек и применяются к контейнеру сетки. Если предоставить более одного значения, оно действует как повторяющийся шаблон. Как уже упоминалось, значением по умолчанию является `auto`, которое определяет размер строки или столбца, пригодный для размещения контента.

В следующем примере я создаю явную сетку в два столбца шириной и в две строки высотой. Один из grid-элементов я помещаю в положение, эквивалентное пятому столбцу и третьей строке. Моя явная сетка недостаточно велика, чтобы разместить его, поэтому к ней добавляются дорожки в соответствии с размерами, которые я указала в свойствах `grid-auto-*` (рис. 16.42):

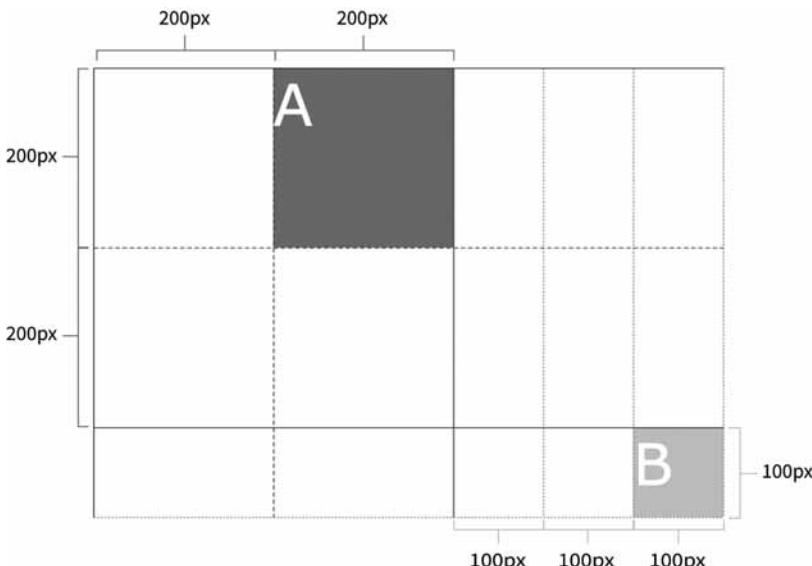
Разметка

```
<div id="littlegrid">
  <div id="A">A</div>
  <div id="B">B</div>
</div>
```

Правила стиля

```
#littlegrid {
  display: grid;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-columns: 100px;
  grid-auto-rows: 100px;
}
#A {
  grid-row: 1 / 2;
  grid-column: 2 / 3;
}
#B {
  grid-row: 3 / 4;
  grid-column: 5 / 6;
}
```

Сетка имеет явно определенные две строки и два столбца размером 200 пикселов каждый.



Дорожки строк и столбцов добавляются автоматически по мере необходимости. Они имеют размеры, указанные в свойствах `grid-auto-rows` и `grid-auto-columns` (100 пикселов).

Рис. 16.42. Браузеры автоматически генерируют строки и столбцы для размещения тех grid-элементов, которые не помещаются в заданную сетку

Надеюсь, что этот пример поможет вам сформировать мысленное представление об автоматически сгенерированных строках и столбцах. Более распространенное использование автоматически генерируемых дорожек — это разбиение изображений, списков товаров и тому подобных объектов на столбцы, что позволяет создавать строки по мере необходимости. Следующие правила стилей устанавливают сетку с явными столбцами (в соответствии с шириной области просмотра, но не уже, чем 200 пикселов) и строками высотой 200 пикселов (в нужном количестве):

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
grid-auto-rows: 200px;
```

Направление потока элементов и плотность их расположения

Вы также можете управлять способом автоматического перетекания элементов в сетку с помощью свойства `grid-auto-flow`:

`grid-auto-flow`

- **Значения:** `row` или `column` | `dense` (опционально).
- **По умолчанию** — `row`.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Используйте свойство `grid-auto-flow` чтобы указать, будут элементы заполнять сетку по строкам или по столбцам. Поток, заданный по умолчанию, следует направлению письма документа (слева направо и сверху вниз для английского и других языков, поддерживающих чтение слева направо).

В следующем примере показано, что `grid`-элементы заполняют сетку по столбцам вместо строк, как предусмотрено по умолчанию:

```
#listings {
    display: grid;
    grid-auto-flow: column;
}
```

По умолчанию элементы размещаются в первой области, в которую они в состоянии поместиться. Ячейки, слишком малые для размещения контента, пропускаются, пока не найдется ячейка, достаточно большая для размещения. Если включить для свойства `grid-auto-flow` необязательное ключевое слово `dense`, оно предпишет браузеру заполнить сетку настолько плотно, насколько это возможно, позволяя элементам размещаться не по порядку, но заполнять доступное пространство:

```
#listings {
    display: grid;
    grid-auto-flow: dense rows;
}
```

Пример, приведенный на рис. ЦВ-16.43, слева, показывает заданный по умолчанию режим потока. Обратите внимание, что элементы сетки упорядочены. Если в сетке недостает места для целого элемента, он перемещается вниз и влево, пока не уместится (аналогично плавающему элементу). Работа в этом режиме, как и показано на иллюстрации, может привести к образованию пустых ячеек.

Для сравнения на рис. ЦВ-16.43, справа показано заполнение сетки элементами в режиме `dense` — сетка полностью заполнена, но если посмотреть на нумерацию элементов, мы заметим, что этот способ размещения элементов нарушает правильный

ПЕРЕСТАНОВКА ЭЛЕМЕНТОВ СЕТКИ

До сих пор `grid`-элементы последовательно перетекали в сетку и явно размещались в небольших ее областях. Однако имеется несколько свойств, которые полезны для настройки положения `grid`-элементов.

Изменение порядка визуализации элементов сетки

Как и при работе с Flexbox, свойство `order` можно применить для изменения порядка, в котором выводятся элементы при визуализации. Учтите при этом, что свойство `order` не изменяет порядок, в котором контент сетки читается вспомогательным средством (см. разд. «Свойство `order` (изменение порядка следования Flex-элементов)» ранее в этой главе для получения дополнительной информации об использовании свойства `order`).

Порядок наложения

Элементы в сетке можно расположить так, чтобы они перекрывали друг друга. Если для области сетки назначено более одного элемента, элементы, которые появляются в источнике позже, отображаются поверх элементов, появившихся в источнике ранее, но можно изменить порядок наложения, применив свойство `z-index`. Присваивание большего значения `z-index` более ранним (по порядку отображения) элементам заставляет их отображаться над элементами, которые появляются позже (см. разд. «Порядок наложения» главы 15 для получения подробной информации об использовании свойства `z-index`).

порядок их следования. Следует также иметь в виду, что плотный поток не всегда полностью заполняет сетку, как здесь показано, но она, скорее всего, будет иметь меньше пустот и более компактный вид, нежели в заданном по умолчанию режиме.

Расширение сокращенного свойства *grid*

Ранее было показано, как сокращенное свойство *grid* применяется для поддержки размеров дорожек и имен областей. Там мы рассматривали явные сетки, но свойство *grid* также можно применять и при работе с неявными сетками.

Добавление ключевого слова *auto-flow* к строке или информации о дорожке указывает на то, что дорожки на этой оси должны автоматически генерироваться в заданном размере.

Скажем, столбцы мы хотим установить явно, но строки могут генерироваться автоматически, по мере необходимости. Использование сокращенного свойства *grid* для такого общего сценария может выглядеть так:

```
grid: auto-flow 12em / repeat(5, 1fr);
```

Не забывайте, что синтаксис сокращенного свойства *grid* предусматривает следующую запись: первой следует информация о строках (*row*), а затем — через слэш — приводится информация о столбцах (*column*). В рассматриваемом случае действует правило указания автоматически создаваемых строк с высотой, равной 12em, и 5 столбцов по 1fr каждый. То есть, если *auto-flow* применяется к строкам, свойство *grid-auto-flow* устанавливается в позицию *row*.

В следующем примере генерируемая сетка имеет две строки по 300px, но столбцы шириной 100px при добавлении элементов сетки генерируются «на лету»:

```
grid: 300px 300px / auto-flow 100px;
```

То есть, если *auto-flow* применяется к столбцам, свойство *grid-auto-flow* устанавливается в позицию *column*.

Важно помнить, что, поскольку свойство *grid* является сокращенным, каждое не указанное в нем значение воспринимается как заданное по умолчанию. Поэтому, если в таблице стилей свойство *grid* где-то применялось и для установки явных строк и столбцов, они будут потеряны, если позже в ней появится сокращенное свойство *grid* с правилами для неявных вариантов.

Управление пространством в сетке и выравнивание элементов в ячейках

Остальные свойства, определенные в модуле Grid Layout, относятся к формированию «желобов» и выравниванию. Вы можете добавлять между дорожками пространство и настроить выравнивание сетки и ее элементов, применяя многие из методов, уже рассмотренных для Flexbox.

Пространство между дорожками («желоба»)

`grid-row-gap`

`grid-column-gap`

- **Значения:** длина (не должна быть отрицательной).
- **По умолчанию** — 0.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Установка значения длины для свойства `grid-row-gap` добавляет пространство («желоба») между дорожками строк сетки, а для свойства `grid-column-gap` — между (уже догадались) дорожками столбцов. Эффект проявляется таким образом, будто бы линии сетки приобретают ширину, однако эта ширина применяется только к линиям между дорожками, а не к первой и последней линиям сетки (расстоянием между внешними краями можно управлять с помощью отступов).

Существует и сокращенное свойство `grid-gap` — для указания ширины пространства для строк и столбцов за один прием (сначала, как обычно, для строк):

`grid-gap`

- **Значения:** `grid-row-gap` `grid-column-gap`
- **По умолчанию** — 0 0.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

В следующем примере добавлено пространство 20px между строками и пространство 50px между столбцами с помощью сокращенного свойства `grid-gap` (рис. ЦВ-16.44):

```
div#container {  
    border: 2px solid gray;  
    display: grid;  
    grid: repeat(4, 150px) /  
    repeat(4, 1fr);  
    grid-gap: 20px 50px;  
}
```

Выравнивание сетки и ее элементов

Можно выравнивать grid-элементы сетки в их ячейках с помощью словаря выравнивания, который применяется для элементов Flexbox (см. врезку «Выравнивание блока»). Мы скоро перейдем к этим вопросам, но можете поэкспериментировать с ними и самостоятельно.

ИЗМЕНЕНИЕ НАЗВАНИЙ СВОЙСТВ

Со временем названия этих свойств будут изменены на `row-gap`, `column-gap` и `gap`. Пока браузеры не станут поддерживать новый синтаксис, можно использовать префиксные версии `grid-*`, которые будут поддерживаться и далее для обратной совместимости.

ВЫРАВНИВАНИЕ БЛОКА

Далеко не случайно Flexbox и Grid имеют общие свойства и значения выравнивания. Они стандартизированы в собственной спецификации, называемой модулем CSS Box Alignment Module, Level 3 (CSS-модуль выравнивания блока, уровень 3), который служит ссылкой на ряд CSS-модулей. Для получения дополнительных сведений по этой теме обратитесь на сайт по следующему адресу: www.w3.org/TR/css-align/.

Выравнивание отдельных элементов

justify-self

- **Значения:** start | end | center | left | right | self-start | self-end | stretch | normal | auto.
- **По умолчанию** — auto (обращается к значению для justify-items, которое по умолчанию установлено равным normal).
- **Применение** — к grid-элементам.
- **Наследование** — нет.

align-self

- **Значения:** start | end | center | left | right | self-start | self-end | stretch | normal | auto.
- **По умолчанию** — auto (обращается к значению для align-items).
- **Применение** — к grid-элементам.
- **Наследование** — нет.

ЕЩЕ О НАПРАВЛЕНИИ ЗАПИСИ КОНТЕНТА

Значения *self-start* и *self-end* учитывают направление записи контента элемента и используются для выравнивания его начального (*start*) или конечного (*end*) края. Например, если контент представлен на арабском языке, его край *self-start* находится справа, и тогда элемент выравнивается вправо. Значения *start* и *end* учитывают направление письма в grid-контейнере. Значения *left* и *right* являются абсолютными и не меняются в зависимости от системы письма, но они соответствуют значениям *start* и *end* для языков, поддерживающих написание слева направо.

ЦЕНТРИРУЕМ GRID-ЭЛЕМЕНТ

Если необходимо, чтобы grid-элемент был выровнен в своей области сетки по центру, установите для двух его свойств: *align-self* и *justify-self* значение *center*.

Когда grid-элемент не заполняет всю свою область сетки, для него можно указать в ней желательное выравнивание. Горизонтальное (по строке) выравнивание задается с помощью свойства *justify-self*. Свойство *align-self* определяет выравнивание по вертикальной (блочной) оси. Оба эти свойства применяются именно к grid-элементу, поскольку обеспечивают ему выравнивание «себя» (от англ. *self*).

На рис. ЦВ-16.45 показано влияние каждого значения ключевых слов выравнивания. Для элементов с размером, установленным в *auto* (или, другими словами, с явно незаданными значениями свойств *width* и *height*), по умолчанию применяется значение *stretch*. Это и показано во всех предыдущих примерах сетки. Если для grid-элемента сетки заданы ширина и высота, эти размеры сохраняются, и по умолчанию используется значение *start*.

Прочитав раздел про Flexbox, вы теперь должны понимать, о чем здесь идет речь, — например, учитывать, что значения «*start*» и «*end*» применяются для того, чтобы сохранить независимость от направления в языковой системе.

Выравнивание всех элементов сетки

justify-items

- **Значения:** start | end | center | left | right | self-start | self-end | stretch | normal.
- **По умолчанию** — normal (stretch — для незаменяемых элементов, start — для заменяемых).
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

align-items

Значения: start | end | center | left | right | self-start | self-end | stretch | normal.

- **По умолчанию** — normal (stretch — для незаменяемых элементов, start — для заменяемых).
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

Для выравнивания всех элементов сетки за один прием примените свойство `justify-items` — для выравнивания по горизонтальной/линейной оси и свойство `align-items` — для выравнивания по вертикальной/блочной оси. Задайте эти свойства для элемента grid-контейнера, и оно окажет влияние на все элементы сетки. Приведенные здесь ключевые слова (значения) обеспечивают для элемента grid-контейнера выполнение тех же действий, что и приведенные ранее значения для свойств grid-элементов. То есть эти ключевые слова делают то же самое, что показано на рис. ЦВ-16.45, — просто представьте, что это происходит последовательно по всей сетке. Имейте только в виду, что эти настройки будут переопределены свойствами `*-self`.

Выравнивание дорожек в grid-контейнере

Иногда может случиться так, что дорожки сетки не заполнят всю область grid-контейнера — например, если ширина и высота дорожек задана в каких-либо пиксельных размерах. Вы можете определить, каким образом браузер должен обработать оставшееся свободным пространство внутри контейнера, используя свойства `justify-content` (горизонтальная/линейная ось) и `align-content` (вертикальная/блочная ось):

ПОГОВОРИЛИ О ПРОСТРАНСТВЕ, А КАК НАСЧЕТ ПОЛЕЙ?

Для grid-элемента, как и для любого другого элемента, вы можете добавлять поля. При этом полезно знать, что блок поля элемента привязан к ячейке или области сетки, причем пространство поля сохраняется.

Поле можно применять для перемещения элемента в области сетки. Например, настройка `auto` для левого поля сдвигает элемент вправо, как показано ранее в примерах для Flexbox. Настройка `auto`, применяемая в качестве значения для левого и правого полей (если элемент имеет заданную ширину), центрирует его по горизонтали. В Grid также можно присвоить значение `auto` для верхнего и нижнего полей, и, если задана высота, центрирование выполняется в вертикальном направлении. Впрочем, разумеется, имеются и собственные свойства Grid для выравнивания grid-элементов сетки, позволяющие достичь тех же эффектов.

ВНИМАНИЕ!

Описанные в этом разделе свойства выравнивания применяются именно к grid-контейнеру.

justify-content

- **Значения:** start | end | left | right | center | stretch | space-around | space-between | space-evenly.
- **По умолчанию** — start.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

align-content

- **Значения:** start | end | left | right | center | stretch | space-around | space-between | space-evenly.
- **По умолчанию** — start.
- **Применение** — к grid-контейнерам.
- **Наследование** — нет.

На рис. ЦВ-16.46 показан grid-контейнер — он обозначен там серым контуром. Строки и столбцы созданной в нем сетки не заполняют весь контейнер, поэтому образовавшимся свободным пространством надо как-то распорядиться. Ключевые слова start, end и center перемещают всю сетку внутри контейнера, помещая дополнительное пространство после, до или одинаково с обеих ее сторон, соответственно.

КЛЮЧЕВОЕ СЛОВО *STRETCH*

Ключевое слово stretch активизируется только в том случае, если для ширины и высоты дорожки выбрано значение auto.

РАСПРЕДЕЛЯЕМОЕ СВОБОДНОЕ ПРОСТРАНСТВО ДОБАВЛЯЕТСЯ К ИМЕЮЩЕМУСЯ

Распределляемое между дорожками и вокруг них с помощью свойств justify-content и align-content свободное пространство добавляется к любому заданному вами ранее пространству между ними.

Ключевые слова space-around и space-between распределяют пространство вокруг дорожек, как рассматривалось в разделе, посвященном Flexbox. Ключевое слово space-evenly добавляет равное пространство в начале и в конце каждой дорожки и между элементами.

Прежде чем завершить рассмотрение модуля Grid Layout, создадим для страницы «Black Goose Bakery» красивый макет, состоящий из двух колонок (*упражнение 16.6*).

УПРАЖНЕНИЕ 16.6. МАКЕТ НА ОСНОВЕ СЕТКИ ДЛЯ СТРАНИЦЫ «BLACK GOOSE BAKERY»

Страница «Black Goose Bakery» прошла длительный путь преобразований. Мы добавляли в нее отступы, границы и поля. Размещали плавающие картинки, позиционировали графическое изображение заслуженной награды, создавали панель навигации с помощью Flexbox. А сейчас мы можем применить свои новые навыки работы с сеткой для придания странице вида двухколоночного макета, пригодного для вывода на планшеты и большие экраны (рис. 16.47).

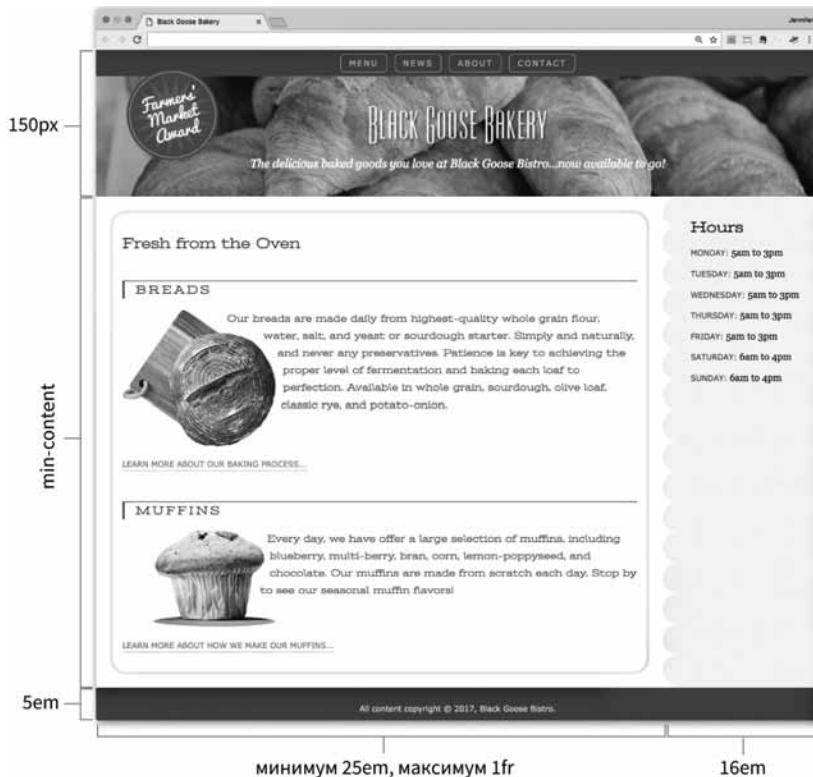


Рис. 16.47. Страница «Black Goose Bakery», оформленная в виде двухколоночного макета на основе сетки

И начнем мы с открытия файла пекарни в том виде, как он был сохранен нами после выполнения [упражнения 16.1](#).

1. Нам нужно добавить в документ немного разметки, которая включит в него элемент, призванный служить нам в качестве контейнера сетки. Откройте HTML-документ `bakery.html`, добавьте элемент `div` вокруг всех элементов контента (от `header` до `footer`) и придайте ему `id` «`container`». Сохраните этот HTML-файл:

```
<body>
  <div id="container">
    <header>...</header>
    <main>...</main>
    <aside>...</aside>
    <footer>...</footer>
  </div>
</body>
```

В таблицу стилей `bakery-styles.css` добавьте следующий стиль, чтобы создать новое отображение элемента `div` в виде сетки:

```
#container {
  display: grid;
}
```

2. Сначала поработаем со строками. На рис. 16.47 показано, что для создания макета нужны три строки. Установите высоту дорожки первой строки в `auto` — это значение будет управлять настройками высоты элементов в ней и автоматически приспосабливать ее для размещения нужного контента. Во второй строке много текста, поэтому снова используйте для ее дорожки значение `auto` — это гарантирует расширение дорожки для размещения всего потребного текста. Для третьей строки высоты, равной `5em`, должно быть достаточно для удобного размещения нескольких строк текста:

```
#container {
    display: grid;
    grid-template-rows: auto auto 5em;
}
```

3. Теперь настроим столбцы дорожек. Похоже, нам понадобятся всего два столбца: один — для основного контента и один для врезки «Hours». Я прибегла к использованию значения `minmax()`, позволяющего гарантировать, что текстовый столбец никогда не сужится меньше, чем до `25em`, но может расширяться для заполнения всего доступного пространства в браузере (`1fr`). Столбец «Hours» отлично выглядит при значении, равном `16em` (впрочем, вы можете поэкспериментировать и с другими значениями):

```
#container {
    display: grid;
    grid-template-rows: auto auto 5em;
    grid-template-columns: minmax(25em, 1fr) 16em;
}
```

4. Затем присвойте областям сетки имена, что позволит легко и эффективно размещать в них элементы, для чего примените свойство `grid-template-areas`:

```
#container {
    display: grid;
    grid-template-rows: auto auto 5em;
    grid-template-columns: minmax(25em, 1fr) 16em;
    grid-template-areas:
        "banner banner"
        "main hours"
        "footer footer";
}
```

5. Теперь, когда все настроено, несложно разместить элементы контента на свои места. Создайте правило стиля для каждого элемента сетки и укажите ему, куда направляться, с помощью свойства `grid-area`:

```
header {
    grid-area: banner;
}
main {
    grid-area: main;
}
aside {
    grid-area: hours;
}
```

```
footer {  
    grid-area: footer;  
}
```

Довольно легко, не правда ли? Сохраните файл и посмотрите страницу в браузере (если еще не сделали этого). Поля в 2,5%, которые ранее были установлены для элемента `main`, отлично размещают его и в изменении не нуждаются. Однако мне кажется уместным удалить правое поле и радиус границы, которые установлены в `aside` для заполнения правого столбца. Я просто закомментировала его, чтобы эта информация сохранилась для дальнейшего применения:

```
aside {  
    ...  
    /* border-top-right-radius: 25px; */  
    /* margin: 1em 2.5% 0 10%; */  
}
```

Вот и все! Откройте страницу `bakery.html` в окне браузера, который поддерживает CSS-сетки, — она должна выглядеть, как на снимке с экрана, представленном на рис. 16.47.

Теперь страница нашей пекарни имеет удобный двухколоночный макет с использованием простой сетки. Тем не менее мы настроили его таким образом, что в реальном мире он сможет соответствовать разным размерам экранов в рамках стратегии адаптивного дизайна (об адаптивном дизайне речь пойдет в следующей главе). Поскольку сетки не поддерживаются браузерами Internet Explorer, Edge и более старыми браузерами, вам, возможно, придется создавать запасные варианты макетов, применяя Flexbox или плавающие элементы, в зависимости от степени универсальности макета. Не хочется вас разочаровывать, но вы должны понять, что хотя это упражнение позволило вам отточить свои навыки, оно является лишь частью гораздо более широкого производственного процесса.

ЕЩЕ О ЗАПАСНЫХ ВАРИАНТАХ

Для уточнения методов разметки, применяющих «плавание» и позиционирование, которые можно использовать в качестве запасных (резервных) вариантов, ознакомьтесь со статьей «*Page Layout with Floats and Positioning*» (PDF), которая доступна на сайте по адресу: learningwebdesign.com/articles/.

Ресурсы Grid в Интернете

Я уверена, что вы, продолжая совершенствоваться в Grid Layout, найдете в Интернете множество отличных ресурсов, поскольку их там все больше и больше постоянно появляется. Тем не менее я хотела бы указать вам на несколько наиболее полных и авторитетных источников, которые я нашла полезными, когда сама разбиралась с сетками.

- Сайт Рэйчел Эндрю «Grid By Example» (gridbyexample.com) — Рейчел Эндрю (Rachel Andrew), пионер в области Grid Layout, располагает невероятной коллекцией статей, бесплатных видеоуроков, информацией о браузерной поддержке и пр. Вы также можете поискать в Интернете ее отличные выступления на конференциях по этой теме.

СВОД СВОЙСТВ GRID

Вот удобный список свойств Grid, упорядоченный по тому, относятся ли они к контейнеру или к отдельным элементам сетки.

Свойства Grid-контейнеров

```
display: grid | inline-grid  
grid  
grid-template  
grid-template-rows  
grid-template-columns  
grid-template-areas  
grid-auto-rows  
grid-auto-columns  
grid-auto-flow  
grid-gap  
grid-row-gap  
grid-column-gap  
justify-items  
align-items  
justify-content  
align-content
```

Свойства Grid-элементов

```
grid-column  
grid-column-start  
grid-column-end  
grid-row  
grid-row-start  
grid-row-end  
grid-area  
justify-self  
align-self  
order (не является частью модуля Grid)  
z-index (не является частью модуля Grid)
```

- Джен Симмонс «Experimental Layout Lab» (labs.jensimmons.com) — Джен Симмонс (Jen Simmons), работающая в Mozilla Foundation, демонстрирует возможности модуля Grid Layout в своей лаборатории Experimental Layout Lab. Вам определенно стоит ознакомиться с ее интересными примерами по Grid и другим новым CSS-технологиям, а также с упражнениями, позволяющими вам совместно с Джен создавать код.

- Большое число статей Джен, посвященных CSS-модулю Grid Layout, можно найти на сайте: jensimmons.com/writing. Также я рекомендую сводку ресурсов для изучения Grid Layout, которая доступна на сайте по адресу: jensimmons.com/post/feb-27-2017/learn-css-grid. Можно также обратиться к серии видеоматериалов Джен на YouTube под названием «Layout Land» ([youtube.com](https://www.youtube.com), введите в строке поиска фразу: Layout Land Jen Simmons).
- Grid Garden от Томаса Парка (cssgrid-garden.com) — Если вы увлеклись игрой Flexbox Froggy, созданной Томасом Парком (Thomas Park), вам будет интересно поиграть в его игру Grid Garden, чтобы познакомиться с CSS Grid Layout.

КОНТРОЛЬНЫЕ ВОПРОСЫ

В главе рассмотрено большое число тем. Посмотрим, как вы справитесь с контрольными вопросами, которые относятся к некоторым основным моментам изложенного материала. Как всегда, в *приложении 1* вы найдете ответы на эти вопросы.

Flexbox

1. Как превратить элемент в гибкий элемент?

2. Сопоставьте свойства с функциями:

- | | |
|--------------------|--|
| a. justify-content | 1. Распределяет пространство вокруг и между flex-линиями поперечной оси. |
| b. align-self | 2. Распределяет пространство вокруг и между элементами главной оси. |
| c. align-content | 3. Размещает элементы на поперечной оси. |
| d. align-items | 4. Размещает определенный элемент на поперечной оси. |

3. Чем отличается align-items от align-content? Что у них общего?

4. Сопоставьте свойства и значения с полученными эффектами:

- | | |
|--------------------|---|
| a. flex: 0 1 auto; | 1. Элементы абсолютно негибкие, не сжимаются и не увеличиваются. |
| b. flex: none; | 2. Элемент при помощи flex: 1 станет в два раза шире по сравнению с другими, а также может уменьшиться. |
| c. flex: 1 1 auto; | 3. Элементы полностью гибкие. |
| d. flex: 2 1 0px; | 4. Элементы могут сжиматься, но не становятся больше. |

5. Сопоставьте следующие объявления flex-flow с результирующими гибкими контейнерами (рис. 16.48):

- a. flex-flow: row wrap;
- b. flex-flow: column nowrap;
- c. flex-flow: row wrap-reverse;
- d. flex-flow: column wrap-reverse;
- e. flex-flow: row nowrap;

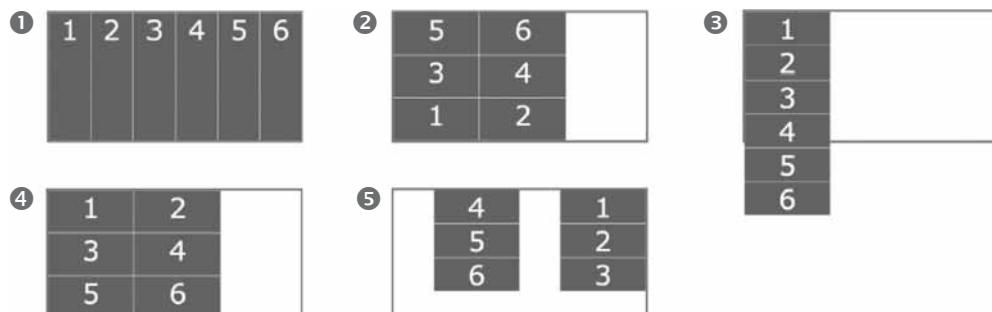


Рис. 16.48. Различные настройки для flex-flow

6. Напишите стили для отображения элементов flexbox в порядке, показанном на рис. 16.49.

Исходный код HTML

```
<div id="container">
<div class="box box1">1</div>
<div class="box box2">2</div>
<div class="box box3">3</div>
<div class="box box4">4</div>
<div class="box box5">5</div>
<div class="box box6">6</div>
<div class="box box7">7</div>
</div>
```

После переупорядочения



Рис. 16.49. Напишите стили для размещения элементов (items) в указанном порядке

Модуль Grid Layout

7. В чем ключевое различие между Grid Layout и Flexbox? Назовите, по крайней мере, одно схожее свойство (может быть несколько ответов).
8. Создайте шаблон сетки для макета, показанного на рис. 16.50, используя свойства `grid-template-rows` и `grid-template-columns`.

Запишите его снова, применяя на этот раз сокращенное свойство `grid`.



Рис. 16.50. Сформируйте макет сетки для этой grid-структурь

9. Сопоставьте следующие объявления стилей с пунктирными элементами сетки, показанными на рис. 16.51. В дополнение к автоматической нумерации некоторые из линий сетки названы так, как помечены.

a. _____

```
grid-row-start: 1;  
grid-row-end: 3;  
grid-column-start: 3;  
grid-column-end: 7;
```

b. _____

```
grid-area: 2 / 2 / span 4 / 3;
```

c. _____

```
grid-area: bowie;
```

d. _____

```
grid-row: -2 / -1;  
grid-column: -2 / -1;
```

e. _____

```
grid-row-start: george;  
grid-row-end: ringo;  
grid-column-start: paul;  
grid-column-end: john;
```

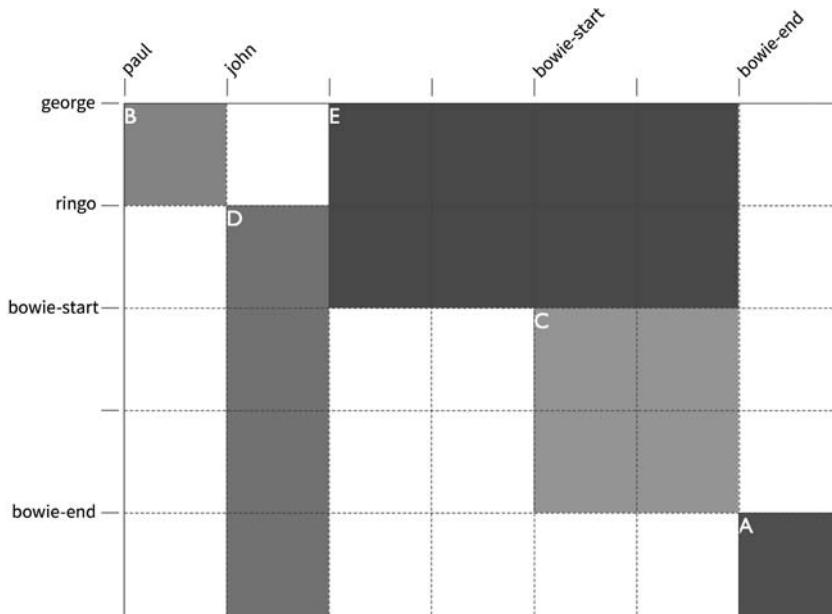


Рис. 16.51. Сопоставьте примеры стиля с элементами этой сетки

10. Запишите правило, которое добавляет между столбцами пространство размером 1em в grid-контейнер под названием #gallery.

11. Сопоставьте задания с объявлениями:

- a. justify-self: end;
- b. align-items: end;
- c. align-content: center;
- d. align-self: stretch;
- e. justify-items: center;

_____ Сформируйте конкретный элемент растяжения для заполнения соответствующего контейнера.

_____ Поместите изображение по правому краю области сетки (если язык чтения имеет направление «слева направо»).

_____ Центрируйте всю сетку вертикально в ее контейнере.

_____ Переместите все изображения сетки в нижнюю часть соответствующих ячеек.

_____ Центрируйте горизонтально все элементы в соответствующих областях

ОБЗОР CSS: СВОЙСТВА МАКЕТА

В табл. 16.1 и 16.2 в алфавитном порядке приводятся краткие описания свойств Flexbox и Grid соответственно с указанием на применимость их по отношению к контейнеру или элементу.

Таблица 16.1. Свойства Flexbox

Свойства flex-контейнера	
display: flex	Включает режим flexbox и превращает элемент в контейнер
flex-direction	Указывает направление, в котором элементы помещаются в flex-контейнер
flex-wrap	Определяет, будут ли flex-элементы смешены на одну строку или разнесены на несколько строк
flex-flow	Сокращенное свойство для flex-direction и flex-wrap
justify-content	Определяет, каким образом пространство распределяется между элементов вдоль главной оси и вокруг них
align-content	Выравнивает гибкие линии внутри flex-контейнера, если на поперечной оси имеется дополнительное пространство
align-items	Определяет, каким образом пространство распределяется вокруг элементов вдоль поперечной оси

Табл. 16.1 (окончание)

Свойства flex-элемента	
<td>Определяет, каким образом один элемент выравнивается вдоль поперечной оси (переопределяет свойство align-items)</td>	Определяет, каким образом один элемент выравнивается вдоль поперечной оси (переопределяет свойство align-items)
flex	Сокращенное свойство для flex-grow, flex-shrink и flex-basis. Определяет, каким образом элементы изменяют свои размеры в соответствии с доступным пространством
flex-basis	Указывает начальный основной размер для flex-элемента
flex-grow	Определяет, насколько flex-элемент может увеличиться, если в контейнере имеется свободное пространство
flex-shrink	Определяет, насколько flex-элемент может сжаться, если в контейнере недостаточно места
order	Указывает порядок размещения элементов в контейнере

Таблица 16.2. Свойства Grid

Свойства grid-контейнера	
display: grid inline-grid	Устанавливает режим перевода элемента в контекст сетки
grid-template	Сокращенное свойство для указания grid-template-areas, grid-template-rows и grid-template-columns
grid-template-areas	Назначает имена областям в сетке
grid-template-columns	Определяет размеры дорожек для столбцов в явных сетках
grid-template-rows	Определяет размеры дорожек для строк в явных сетках
grid-auto-columns	Определяет размеры дорожек для автоматически генерируемых столбцов
grid-auto-flow	Указывает направление и плотность, при значениях которых элементы автоматически попадают в сетку
grid-auto-rows	Определяет размеры дорожек для автоматически генерированных строк
grid	Сокращенное свойство для указания grid-template-rows, grid-template-columns и grid-template-areas, или grid-auto-flow, grid-auto-rows и grid-auto-columns
grid-gap	Сокращенное свойство для grid-row-gap и grid-column-gap
grid-column-gap	Определяет ширину пространства между столбцами
grid-row-gap	Определяет ширину пространства между строками
justify-items	Указывает на выравнивание всех элементов сетки вдоль встроенной оси в соответствующих областях

Табл. 16.2 (окончание)

Свойства grid-контейнера (окончание)	
justify-content	Указывает на выравнивание дорожек сетки вдоль встроенной оси в контейнере
align-items	Указывает на выравнивание всех элементов сетки вдоль оси блока в соответствующих областях сетки
align-content	Указывает на выравнивание дорожек сетки вдоль оси блока в контейнере
Свойства grid-элемента	
grid-column	Сокращенное свойство для указания <code>grid-column-start</code> и <code>grid-column-end</code>
grid-column-end	Обозначает конечную строку столбца, в который должен быть помещен элемент
grid-column-start	Обозначает начальную строку столбца, в который должен быть помещен элемент
grid-row	Сокращенное свойство для указания <code>grid-row-start</code> и <code>grid-row-end</code>
grid-row-end	Обозначает конечную строку, в которую должен быть помещен элемент
grid-row-start	Обозначает начальную строку, в которую должен быть помещен элемент
grid-area	Присваивает элемент сетки именованной области или области, описываемой четырьмя граничными линиями сетки
align-self	Указывает выравнивание для одного элемента по оси блока в пределах области сетки
justify-self	Указывает выравнивание для одного элемента вдоль встроенной оси в пределах области сетки
order	Определяет порядок для отображения элемента относительно других элементов источника
z-index	Определяет порядок наложения элемента относительно других элементов при наличии наложения

ГЛАВА 17

АДАПТИВНЫЙ ВЕБ-ДИЗАЙН

В этой главе...

- ▶ Суть адаптивного веб-дизайна и его значимость
- ▶ Текущие макеты
- ▶ Медиазапросы
- ▶ Стратегии дизайна и шаблоны
- ▶ Способы тестирования

Впервые концепцию адаптивного (отзывчивого) веб-дизайна (Responsive Web Design) я упомянула в главе 3, когда мы искали способы учесть при создании веб-страниц размеры разных экранов. Здесь же мы углубимся в рассмотрение соответствующих стратегий и методов.

Стоит напомнить, что адаптивный (отзывчивый) веб-дизайн (АВД) — это дизайнерский и производственный подход, позволяющий создавать веб-сайты, которые удобно просматривать и с которыми удобно работать на любых устройствах. Основной его принцип состоит в том, что все устройства получают один и тот же HTML-код, расположенный по одному и тому же интернет-адресу (URL), но, в зависимости от размера области просмотра, используются разные стили, помогающие изменить места расположения компонентов веб-страницы и оптимизировать удобство ее просмотра. На рис. 17.1 показаны примеры «отзывчивых» сайтов, которые могут отображаться как на смартфоне, так и на планшете и настольном компьютере, но важно помнить, что эти сайты хорошо работают с любыми значениями ширины экрана.

ОТЗЫВЧИВЫЙ ДИЗАЙН

Каждый сайт, показанный на рис. 17.1, имеет один изменяющийся дизайн, а не три разных макета. Однако некоторые сайты создаются так, чтобы иметь ряд макетов, ориентированных на конкретные устройства, и этот подход — в отличие от «отзывчивого дизайна» — тоже иногда называется адаптивным дизайном (см. в главе 3 примечание «Адаптивный веб-дизайн или Отзывчивый веб-дизайн, в чем разница?»).

ЗАЧЕМ НУЖЕН АВД?

С той поры, когда в 2007-м году появились iPhone, пользователи обращаются в Интернет с помощью телефонов (смартфонов) любых размеров, планшетов, «фаблетов», ноутбуков с сенсорным экраном, носимых устройств, телевизоров, игровых приставок, холодильников, и кто знает, какие устройства еще могут появиться.



Рис. 17.1. Примеры «отзывчивых» сайтов, которые подстраиваются под маленькие, средние и большие экраны, включая все промежуточные размеры

В 2016-м году использование мобильного Интернета превзошло использование настольных компьютеров, что стало важной вехой в его развитии. Доля веб-трафика, поступающего с устройств, отличных от браузеров настольных компьютеров, постоянно растет. Примерно для 10% американцев смартфон или планшет реализуют единственный вариант зайти в Интернет из-за отсутствия доступа к компьютеру или высокоскоростному Wi-Fi на работе или дома (по данным Pew Research Center: «Smartphone Use in 2015», www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/). Молодые пользователи обычно также выбирают для доступа к Интернету мобильные устройства. И, вообще, подавляющее большинство нас в течение дня получает доступ к Интернету исключительно с нескольких платформ (телефон, планшет, компьютер). Вполне естественно и нетрудно догадаться, что все заинтересованы использовать ваш контент или сервис, независимо от варианта доступа к разработанному вам сайту.

В таком случае на выручку вам придет АВД. Создав адаптивный сайт, вы сможете гарантировать, что его мобильные посетители получат такой же контент, что и все остальные (хотя он может быть иначе организован), поскольку они не станут довольствоваться сокращенным вариантом

контента или урезанными функциями только потому, что применяют смартфон. Кроме того, посетителям, которые начинают просматривать ваш сайт на одном устройстве, а завершать — на другом, вы сможете обеспечить непрерывную работу.

АДАПТИВНЫЙ ДИЗАЙН — СТАНДАРТНЫЙ СПОСОБ СОЗДАНИЯ ВЕБ-САЙТОВ

Адаптивный дизайн становится стандартным способом создать веб-сайт, который отвечает требованиям нашей современной мультимедийной среды.

Фактически для многих веб-дизайнеров «адаптивный дизайн» теперь стал просто «веб-дизайном». Вместо нишевого подхода он становится стандартным способом создания веб-сайта, отвечающего требованиям нашей современной мультимедийной среды.

РЕЦЕПТ АДАПТИВНОСТИ

Лавина мобильных устройств с поддержкой доступа в Интернет изначально вызвала шок в сообществе веб-дизайнеров. Привыкнув к разработке сайтов исключительно для больших настольных экранов, они не понимали, как размещать контент на экранах, которые умещаются на ладони.

Одним из решений было использование встроенных в телефон функций веб-дисплея. По умолчанию на мобильных устройствах вся веб-страница отображается в сжатом виде, чтобы поместиться на любом доступном экране. Пользователи могут пальцами увеличивать детализацию и прокручивать различные части страницы. Хотя технически это работает, но такое решение далеко от оптимального.

Другой подход состоит в создании отдельного мобильного сайта только для небольших экранов и находящихся в дороге людей. Имеется большое число компаний и служб, использующих выделенные мобильные («м-дот») сайты, — можно упомянуть Твиттер и Фейсбук, но сайты «м-дот» уже отходят, уступая место АВД. Google поддерживает этот процесс, отдавая предпочтение адаптивным сайтам с единственными URL-адресами перед мобильными версиями сайтов.

В 2010-м году Итан Маркотт (Ethan Marcotte) в своей статье «Адаптивный веб-дизайн» (alistapart.com/article/responsive-webdesign) привел другое, более гибкое решение, которое стало краеугольным камнем современного веб-дизайна. В этой главе я следую «ингредиентам» для АВД, которые Итан описывает в своей книге «Responsive Web Design».

МОБИЛЬНЫЕ САЙТЫ

Мобильные сайты обсуждались на врезке «Сайты М-дот» в главе 3.

Методика адаптивного веб-дизайна включает три основных компонента:

- **гибкую сетку** — вместо сохранения постоянной ширины адаптивные сайты применяют методы, позволяющие им сжиматься и перетекать в доступное пространство браузера;
- **гибкие изображения** — изображения и другие встроенные медиаобъекты должны иметь возможность масштабироваться, чтобы обеспечить соответствие с входящими в них элементами;
- **медиазапросы CSS** — медиазапросы дают возможность доставлять наборы правил только на те устройства, которые соответствуют определенным критериям, таким, как ширина и ориентация.

К этому списку ингредиентов я могу добавить элемент области просмотра `meta`, который позволяет ширине веб-страницы соответствовать ширине экрана. А теперь начнем наше погружение в механику АВД.

НАСТРОЙКА ОБЛАСТИ ПРОСМОТРА

Чтобы вместить стандартные веб-сайты на небольшие экраны, мобильные браузеры отображают их страницы на «холсте», называемом областью просмотра, а затем сужают эту область просмотра в соответствии с шириной экрана (ширины устройства). Например, на iPhone мобильный браузер Safari устанавливает ширину области просмотра в 980 пунктов, поэтому веб-страница отображается в ней так, как в окне браузера настольного компьютера с шириной 980 пикселов. Затем это отображение уменьшается до ширины экрана (от 320 до 414 точек, в зависимости от модели iPhone), в результате большая часть информации размещается в крошечном пространстве.

ЕДИНИЦА ИЗМЕРЕНИЯ «ПУНКТ»

Макеты iOS измеряются в пунктах (*point*) — единицах измерения, не зависящих от количества составляющих физический экран пикселов. Пункты и пиксели устройства будут подробно рассмотрены в главе 23.

документа, указывает браузеру установить ширину области просмотра, равную ширине экрана устройства (`width=device-width`), какой бы он ни была (рис. 17.2). Значение `initial-scale` устанавливает уровень масштабирования в значение 1 (100%):

```
<meta name="viewport" content="width=device-width,  
initial-scale=1">
```

МЕДИАЗАПРОСЫ

Медиазапросы выполняются все время, даже когда страница уже загружена. И если окно просмотра изменяется — например, пользователь переводит смартфон из книжной ориентации в альбомную или изменяет размер окна браузера настольного компьютера, запрос выполняется снова и применяет к контенту стили, соответствующие новой ширине.

Мобильный браузер Safari ввел элемент области просмотра `meta`, который позволяет ему задавать размер начального окна просмотра. Другие мобильные браузеры последовали его примеру.

ИЗБЕГАЙТЕ АТРИБУТА `MAXIMUM-SCALE=1`

Элемент `meta` области просмотра также имеет атрибут `maximum-scale`, и присваивание ему значения 1 (`maximum-scale=1`) запрещает пользователям изменять масштаб страницы. Я настоятельно рекомендую избегать значения, поскольку изменение размера важно для доступности и удобства использования сайта.

Благодаря элементу области просмотра `meta` создается область просмотра того же размера, что и экран.

По умолчанию область просмотра уменьшается до размера экрана.

область просмотра = 980 пунктов



ширина экрана = 320 пунктов

```
<meta name="viewport"  
      content="width=device-width,  
      initial-scale=1">
```

область просмотра = 320 пунктов



ширина экрана = 320 пунктов

Рис. 17.2. Элемент `meta` области просмотра устанавливает соответствие разрешения окна просмотра браузера устройства с разрешением его экрана

ГИБКИЕ СЕТКИ (ТЕКУЧИЕ МАКЕТЫ)

Когда мы в предыдущей главе рассматривали Flexbox и Grid, я приводила вам примеры расширения и сжатия элементов для заполнения доступного пространства их контейнеров. Подобная текучесть — это именно то поведение, которое необходимо, чтобы контент точно подходил к широкому диапазону размеров области просмотра. Текущие макеты (или «гибкие сетки»), как называет их Итан Маркотт в своих работах, являются основой адаптивного (отзывчивого) дизайна.

В *текущем макете* область страницы и ее сетка изменяются пропорционально для заполнения доступной ширины экрана или окна (рис. 17.3, *вверху*). Этого легко добиться за счет применения единиц `fr` и `minmax()` в CSS Grid layout и установок свойства `flex` в Flexbox. Если приходится ориентироваться на устаревшие браузеры, которые не поддерживают стандарты макета CSS, можно применять для горизонтальных размеров процентные значения — тогда элементы останутся пропорциональными при различных размерах экранов (см. врезку «Преобразование пикселов в проценты»).

Текущие макеты пропорционально заполняют область просмотра.



w3c.org

Фиксированные макеты остаются одинаковыми по размеру и могут попасть под обрезку или получить излишнее дополнительное пространство.



kexp.org

Рис. 17.3. Примеры текущего и фиксированного макетов

ПРЕОБРАЗОВАНИЕ ПИКСЕЛОВ В ПРОЦЕНТЫ

Для преобразования единиц измерения в макете из пикселов в проценты используйте следующую формулу:

цель : контекст = результат

где:

- цель — размер элемента, который подвергнут изменению;
- контекст — размер элемента, включенного в контекст;
- результат — процентное соотношение, применяемое в правилах стиля.

Не беспокойтесь об округлении длинных десятичных значений. Браузеры знают, что с ними делать, а дополнительная точность не помешает.

В прошлом, когда было известно, что все рассматривают сайты на настольных мониторах, макеты *фиксированной ширины* были нормой. (О, как прекрасные далекие времена, когда можно было иметь дело только с радикальной несовместимостью поддержки браузеров!) Как следует из названия, макеты с фиксированной шириной создаются при определенной пиксельной ширине области просмотра (рис. 17.3, *внизу*), так что при ширине 960 пикселов мы получим вполне современный вид.

Указание всех измерений в пиксельных значениях позволяло дизайнерам так управлять макетом, как это можно было бы реализовать в случае печати, и гарантировало, что пользователи на всех платформах и в браузерах увидят практически одну и ту же страницу, а, возможно, и точно такую же.

СТАНДАРТНАЯ ШИРИНА СТРАНИЦЫ

Дизайнеры использовали ширину, равную 960 пикселам, в качестве стандартной ширины страницы, потому что она в то время занимала стандартную ширину рабочего стола (1028 пикселов) и легко разделялась на равные столбцы. Были также популярны и системы верстки, основанные на сетках из 12 столбцов на 960-пиксельной странице.

Однако не потребовалось много времени, чтобы понять, что невозможно создать отдельные конструкции фиксированной ширины, адаптированные к разным размерам устройств. И всем стало ясно, что текучесть обеспечивает реальное преимущество. Она основана на внутренней природе нормального потока, поэтому и работает с учетом среды, а не вопреки ее возможностям. Поэтому, пока макет перестраивается для заполнения контентом доступной ширины, вам не надо беспокоиться о горизонтальных полосах прокрутки или неудобном пустом месте в браузере.

С другой стороны, применение текучих макетов может привести к тому, что строки текста станут слишком длинными, поэтому на это стоит обратить внимание. Более подробно о макетах мы поговорим позже в этой главе.

Превращение изображений в гибкие

Превратить обычное изображение в гибкое не столь уж и сложно. Обратимся, например, к следующему правилу стиля, необходимому для уменьшения масштаба изображения до размера его контейнера:

```
img {  
    max-width: 100%;  
}
```

Вот так! Если макет уменьшается, изображения в нем также уменьшаются для соответствия по ширине своим контейнерам. Если контейнер становится больше изображения — например, в случае макетов для планшета или настольного компьютера, то изображение вместе с ним не увеличивается, а останавливается на 100% первоначального размера (рис. 17.4). Когда вы в элементах `img` HTML-документа

```
img { max-width: 100%; }
```



If pure raw cream is stirred rapidly, it swirls and becomes frothy, like the beaten whites of eggs, or "whipped cream." To prevent this in making Philadelphia Ice Cream, un-whipped cream is needed. It is necessary to stir the cream slowly, gently, and when it is very cold, the remaining half of raw cream is added. This gives the smooth, light and rich texture which makes these creams so different from others.

The time for heating varies according to the quality of cream or milk or water; water has requires a longer time than milk. If the cream is heated too long, it will be scalded, not smooth, and if they are churned before the mixture will be cold they will be watery.



If pure raw cream is stirred rapidly, it swirls and becomes frothy, like the beaten whites of eggs, or "whipped cream." To prevent this in making Philadelphia Ice Cream, un-whipped cream is needed, and when it is very cold, the remaining half of raw cream is added. This gives the smooth, light and rich texture which makes these creams so different from others.



In this book, Philadelphia Ice Creams, containing the first group, are very palatable, but expensive. In many parts of the country, however, Philadelphia Ice Cream is very popular. For that reason, I have given a group of recipes for making Philadelphia Ice Cream. It should be remembered that it takes almost "double" to make ice cream from condensed milk and cream, using plain milk, and twice as much to make ice cream from ordinary milk and cream.

Ordinary fruit cream may be made with condensed milk at a cost of about fifteen cents a quart, which, of course, is cheaper than ordinary milk and cream.

In this book, Philadelphia Ice Creams, containing the first group, are very palatable, but expensive. In many parts of the country, however, Philadelphia Ice Cream is very popular. For that reason, I have given a group of recipes for making Philadelphia Ice Cream. It should be remembered that it takes almost "double" to make ice cream from condensed milk and cream, using plain milk, and twice as much to make ice cream from ordinary milk and cream.

Ordinary fruit cream may be made with condensed milk at a cost of about fifteen cents a quart, which, of course, is cheaper than ordinary milk and cream.

СЕЙЧАС УЖЕ СМЕШНО...

Когда я начала заниматься веб-дизайном в далеком 1993-м году, самый распространенный размер монитора ПК составлял 640×480 пикселов, если только вы не занимались наимоднейшим дизайном с экраном 800×600. Первые мои проекты были рассчитаны на фиксированную ширину в 515 пикселов.

Рис. 17.4. Настройка `max-width` для встроенных изображений позволяет им сжиматься (справа) или увеличиваться (в центре) для соответствия доступному пространству, но увеличиваться не больше, чем их фактический размер (слева)

применяете свойство `max-width`, можно опустить для них атрибуты `width` и `height`. Если вы все же устанавливаете атрибут `width`, убедитесь, что для атрибута `height` установлено значение `auto` — в противном случае изображение не будет масштабироваться пропорционально.

Адаптивные изображения

Но подождите — не так все просто, правда? Если вернуться к обсуждению адаптивных изображений в главе 7, вы вспомните, что требуется кое-что предпринять, чтобы избежать ненужной загрузки больших изображений на небольшие устройства и одновременно обеспечить большие мониторы высокой четкости изображениями с максимальным разрешением. Выбор оптимального размера изображения для обеспечения высокой производительности является частью процесса адаптивного дизайна, но в этой главе не станем концентрироваться на этом вопросе. У нас есть вещи посередине!

Другие внедренные медиаобъекты

Видео и другие внедренные (с помощью элементов `object` или `embed`) медиаобъекты также нуждаются в масштабировании в адаптирующемся окружении. К сожалению, видеоролик при уменьшении его ширины не сохраняет свои пропорции, поэтому для получения удовлетворительных результатов нужно еще очень потрудиться. Тьерри Кобленц (Thierry Koblentz) хорошо описывает одну применяемую при этом стратегию в статье «*Creating Intrinsic Ratios for Video*», которая доступна на сайте по адресу: www.alistapart.com/articles/creating-intrinsic-ratios-for-video. Существует также плагин JavaScript под названием `FitVids.js`, созданный Крисом Коулером (Chris Coyier) и сотрудниками фирмы Paravel, который автоматизирует методику Кобленца для видео с большой шириной. Он доступен на сайте по адресу: fitvidsjs.com.

Магия медиазапросов

Теперь рассмотрим настоящий элемент адаптивного дизайна — медиазапросы! Медиазапросы применяют различные стили в зависимости от характеристик браузера: ширины его окна, вертикальной или горизонтальной ориентации, разрешения и т. п. Именно они позволяют «на лету» направлять одноколоночный макет на маленькие экраны, а многоколоночный макет — на большие.

Такой запрос включает мультимедийный тип, за которым следует определенная функция и значение для проверки. Критерии сопровождаются взятым в фигурные скобки набором стилей, применяемых по результатам запроса. Структура медиазапроса, применяемая в таблице стилей, имеет следующий вид:

```
@media type and (feature: value) {  
    /* стили для браузеров, которые соответствуют этому критерию */  
}
```

Давайте разберем это на примере. Следующие медиазапросы проверяют, находится ли область просмотра на экране в горизонтальной (`landscape`) или вертикальной (`portrait`) ориентации. Если запрос обнаружит, что область просмотра находится в горизонтальной ориентации — фоновый цвет страницы станет голубым, если же в вертикальной — коралловым (рис. ЦВ-17.5). При отображении на экране смартфона, который мы будем наклонять от вертикального положения к горизонтальному и обратно, цвета станут соответственно меняться. Это не очень практичный вариант дизайна, но он хорошо иллюстрирует функционирование медиазапросов:

```
@media screen and (orientation: landscape) {  
    body {  
        background: skyblue;  
    }  
}  
@media screen and (orientation: portrait) {  
    body {  
        background: coral;  
    }  
}
```

Мультимедийные типы

Мультимедийные типы, включаемые в первую часть запроса, появились в CSS2 как способ привязки стилей к конкретным мультимедиа. Например, тип `@media` rule предоставляет набор стилей только при печати документа (и не проверяет какие-либо иные функции или значения):

```
@media print {  
    /* здесь указываются специфичные для печати стили */  
}
```

Наиболее часто используемыми мультимедийными типами являются `all` (все), `print` (печать), `screen` (экран) и `speech` (речь). Если вы делаете разработку для экрана, тип носителя указывать не обязательно и его можно опустить, как показано в приведенном далее примере синтаксиса, но его присутствие в любом случае не повредит, и я для большей ясности продолжу включать медиатип экрана в свои примеры.

```
@media (feature: value) {  
}
```

О ВЛОЖЕННЫХ ФИГУРНЫХ СКОБКАХ

Вас может немного запутать наличие фигурных скобок объявления стилей, вложенных в фигурные скобки медиазапроса. Убедитесь, что вы располагаете корректным количеством фигурных скобок и правильно их вкладываете. Полезно и наличие отступа. Многие программы редактирования кода применяют еще и цветовое кодирование, помогающее правильно записать запрос.

УСТАРЕВШИЕ МЕДИАТИПЫ

В CSS2 также определены типы: aural (слуховой), handheld (ручной), braille (брайлевский), embossed (рельефный), projection (проекционный), tty (телетайп) и tv (телевидение), но, согласно последней спецификации Media Queries Level 4 (в настоящее время эта спецификация находится на стадии рабочего проекта), они устарели и не рекомендуются к использованию.

Свойства медиазапросов

Медиазапросы CSS3 поднимают уровень запросов еще на одну позицию вверх, позволяя нам проверять определенное свойство области просмотра или устройства в целом (пример проверки ориентации устройства был приведен на рис. ЦВ-17.5). Чаще всего проверке подвергается ширина области просмотра. Вы также можете проверить ее минимальную (`min-width`) и максимальную ширину (`max-width`).

ЗАПРОСЫ МИНИМАЛЬНОЙ ШИРИНЫ

Запросы минимальной ширины (`min-width`) — это один из способов создания адаптивного дизайна для мобильных устройств.

В следующем простом примере демонстрируется отображение заголовков причудливым курсивным шрифтом, если ширина области просмотра составляет `40em` или больше, — то есть имеется достаточно места, для того чтобы шрифт был разборчивым. Если же область просмотра не соответствует запросу (поскольку уже `40em`), используется простой шрифт с засечками:

```
h1 {
    font-family: Georgia, serif;
}
@media screen and (min-width: 40em) {
    h1 {
        font-family: 'Lobster', cursive;
    }
}
```

Полный список свойств устройств, которые можно обнаружить с помощью медиазапросов, представлен в табл. 17.1.

Таблица 17.1. Медиасвойства, которые можно оценить с помощью медиазапросов

Свойство	Описание
<code>width</code>	Ширина области просмотра. Также <code>min-width</code> и <code>max-width</code>
<code>height</code>	Высота области просмотра. Также <code>min-height</code> и <code>max-height</code>
<code>orientation</code>	Является ли ориентация устройства <code>portrait</code> или <code>landscape</code>
<code>aspect-ratio</code>	Отношение ширины области просмотра к высоте (ширина/высота). Например: <code>aspect-ratio: 16/9</code>
<code>color</code>	Битовая глубина экрана. Например, <code>color: 8</code> указывает, что устройство имеет, по крайней мере, 8-битовый цвет
<code>color-index</code>	Количество цветов в таблице соответствия цветов
<code>monochrome</code>	Количество битов на пиксель в монохромном устройстве
<code>resolution</code>	Плотность пикселей в устройстве. Это актуально для дисплеев с высоким разрешением
<code>scan</code>	Применяет ли тип медиа <code>tv</code> прогрессивную (построчную) или чересстрочную развертку. (Не применяются префиксы <code>min-</code> / <code>max-</code> .)
<code>grid</code>	Использует ли устройство строчный дисплей — например, окно терминала. (Не применяются префиксы <code>min-</code> / <code>max-</code> .)

Устаревшие свойства

В табл. 17.2 приведен список свойств, устаревших согласно рабочему проекту Media Queries Level 4 Working Draft и не рекомендуемых к использованию.

Таблица 17.2. Устаревшие медиасвойства

Свойство	Описание
device-width	Ширина поверхности отображения для устройства (весь экран). Не рекомендуется, поскольку имеется свойство width
device-height	Высота поверхности отображения для устройства (весь экран). Не рекомендуется, поскольку имеется свойство height
device-aspect-ratio	Отношение ширины всего экрана (поверхности отображения) к высоте. Не рекомендуется, поскольку имеется свойство aspect-ratio

Новые свойства, включенные в Media Queries Level 4

В табл. 17.3 приведены свойства, добавленные в рабочий проект Working Draft of MQ4. Некоторые из них могут получить браузерную поддержку, а некоторые, вероятно, будут исключены из окончательной версии проекта. Я включила их в книгу, чтобы показать, как W3C представляет себе развитие системы медийных запросов. Дополнительные сведения по этой теме можно найти на сайте по адресу: drafts.csswg.org/mediaqueries-4.

Таблица 17.3. Новые медиасвойства

Свойство	Описание
update-frequency	Как быстро (если это вообще имеет место) устройство вывода изменяет внешний вид контента
overflow-block	Каким образом устройство обрабатывает контент, который переполняет область просмотра вдоль оси блока
overflow-inline	Можно ли прокрутить контент, который выходит за область просмотра вдоль горизонтальной оси
color-gamut	Примерный диапазон цветов, которые поддерживаются пользовательским агентом и устройством вывода
pointer	Снабжен ли основной механизм ввода указывающим устройством и насколько оно точное
hover	Позволяет ли механизм ввода пользователю наводить курсор на элементы
any-pointer	Снабжен ли еще какой-либо доступный механизм ввода указательным устройством и насколько оно точное
any-hover	Позволяет ли еще какой-либо доступный механизм ввода выполнить наведение

Использование медиазапросов

Вы можете использовать медиазапросы в таблице стилей или для загрузки внешних таблиц стилей при выполнении каких-либо условий. Медиазапросы нельзя использовать со встроенными стилями.

В таблице стилей

Наиболее распространенным способом организации медиазапросов является использование правила `@media` («`at-media`») прямо в таблице стилей. Все примеры в этой главе — это правила `@media`.

При использовании медиазапросов в таблице стилей важен порядок следования правил. Поскольку правила, приведенные в таблице стилей, затем переопределяют предшествующие им правила, медиазапрос должен следовать *после любых* правил с теми же объявлениями.

Стратегия использования медиазапросов заключается в задании базовых стилей, которые применяются по умолчанию, а затем в переопределении конкретных правил, необходимом для оптимизации отображения при обращении к альтернативным средам просмотра. В АВД рекомендуется сначала устанавливать стили для небольших экранов и браузеров, которые не поддерживают медиазапросы, а затем с помощью таблицы стилей добавлять стили для все более крупных экранов.

Именно это и сделано в примере с переключением шрифтов в заголовке (см. ранее разд. «Свойства медиазапросов»): для заголовка `h1` устанавливается базовый уровень работы с локальным шрифтом с засечками, а затем с помощью медиазапроса добавляется усовершенствованный вариант для больших экранов.

Во внешней таблице стилей

Для больших или сложных сайтов разработчики могут в отдельные таблицы стилей поместить стили для разных устройств и при соблюдении определенных условий вызывать соответствующий CSS-файл. Одним из методов для условной загрузки отдельных CSS-файлов служит применение атрибута `media` в элементе `link`. В следующем примере для сайта сначала задаются основные стили, а затем указывается таблица стилей, которая применяется, только если ширина устройства превышает 1024 пикселя (и если браузер поддерживает медиазапросы):

```
<head>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="2column-styles.css"
media="screen and
(min-width:1024px) >
</head>
```

Некоторые разработчики находят этот метод полезным для управления модульными таблицами стилей, но его недостаток состоит в необходимости отдельных HTTP-запросов для каждого дополнительного CSS-файла. Применяя этот метод, обязательно предоставьте столько ссылок, сколько необходимо (возможно, по одной на

каждую основную точку прерывания), и полагайтесь на правила @media в таблицах стилей, чтобы учесть определяемые ими незначительные изменения размеров.

Выполнять медиазапросы можно и с помощью правил @import, которые извлекают внешние таблицы стилей из таблицы стилей. Обратите внимание, что слово «медиа» не появляется в синтаксисе такого запроса — только медиатип и запрос:

```
<style>
  @import url("/default-styles.css");
  @import url("/wide-styles.css") screen and (min-width: 1024px);
  /* other styles */
</style>
```

Поддержка браузерами

Завершить обсуждение медиазапросов нельзя без рассмотрения вопроса о поддержке их браузерами. Приятно сообщить, что медиазапросы поддерживаются практически всеми современными десктопными и мобильными браузерами. Обидное исключение составляет браузер Internet Explorer версии 8 и более ранних, который вовсе не поддерживает медиазапросы. Вследствие все еще широкого использования операционной системы Windows XP IE8 продолжает отображаться в статистике применения браузеров (на 1–2%, по моему мнению, опережая IE 9 и 10). Если сайт имеет сотни тысяч посетителей, то этот 1%, в конечном итоге, соответствует значительному числу тех, кто ушел неудовлетворенным.

Если вы считаете, что ваши посетители применяют старые версии IE, у вас есть несколько путей обхода этой проблемы. Во-первых, можно применить полифил **Respond.js**, который добавляет поддержку свойств min-width и max-width в не поддерживающие их браузеры. Этот полифил создан Скоттом Джелем (Scott Jehl) и доступен по адресу: github.com/scottjehl/Respond. Второй вариант состоит в создании отдельной таблицы стилей с макетом (без излишеств) для настольных компьютеров и ее доставки с помощью соответствующего условного комментария именно пользователям браузера IE8 либо более ранней его версии. Другие браузеры проигнорируют содержание этого специфичного для IE комментария:

```
<!-- [if lte IE 8]>
  <link rel="stylesheet" href="/path/IE_fallback.css">
<! [endif]-->
```

Впрочем, в зависимости от статистики посещений именно вашего сайта, вам может не понадобится проявлять беспокойство по поводу поддержки медиазапросов. Удачи!

ВЫБОР ТОЧЕК ПРЕРЫВАНИЯ

Точка прерывания — это точка, в которой для изменения стиля используется медиазапрос. Если задать в медиазапросе min-width: 800px, то можно ска-

ТОЧКА ПРЕРЫВАНИЯ

Точка прерывания — это ширина, в соответствии с которой вы вводите изменение стиля.

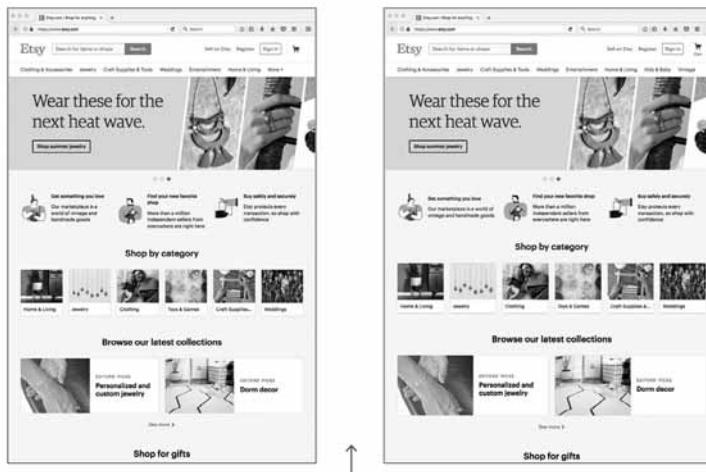
зать, что 800 пикселов — это точка прерывания, в которой следует применять те или иные конкретные стили. На рис. 17.6 показаны точки прерывания, в которых сайт **Etsy.com** на своей домашней странице вносит в макет как значительные изменения дизайна, так и небольшие добавки.



В этой точке прерывания при 480 пикселях навигация по категориям изменяется со списка на фотографии. На верхнюю панель навигации добавляется кнопка «Register».

При 501 пикселе надпись «Sell» изменяется на «Sell on Etsy» (очень тонкая настройка). Кроме того, на панели навигации под полем поиска можно видеть больше ссылок.

При 640 пикселях изображения и сообщения «How Etsy Works» перемещаются в позицию над категориями. При меньших окнах просмотра они были доступны по ссылке «Learn how Etsy works» в желтой полосе.



При 901 пикселе поле ввода для поиска перемещается в верхний заголовок.

При 981 пикселе под пиктограммой корзины покупок появляется слово «Cart», а в заголовке отображается полный список параметров навигации (без ссылки «More»). В этой точке макет расширяется, заполняя большие окна просмотра, пока не достигнет максимальной ширины 1400 пикселов. Затем поля добавляют одинаковое пространство по левому и правому краям для сохранения центрирования макета.

Рис. 17.6. Серия точек прерывания, примененная на адаптивном сайте **Etsy.com** (2017)

Выбор точек прерывания — достаточно сложный процесс, но имеется несколько отличных практических подходов, о которых следует помнить.

Когда АВД был представлен впервые, существовало лишь несколько устройств, о которых следовало позаботиться, поэтому точки прерывания привязывались к общим размерам устройств (320 пикселов — для смартфонов, 768 пикселов — для iPad, и т. п.), и мы создавали отдельный дизайн для каждой точки прерывания. Это не занимало много времени, пока нам не пришлось иметь дело с шириной устройств почти в каждой точке от 240 до 3000+ пикселов. И стало ясно, что подход на основе устройств на эти условия определенно не масштабируется.

Точки прерывания, основанные на модуле

Более подходящим вариантом является создание точек прерывания для отдельных частей страницы (модулей), а не для единовременного переключения всей страницы сразу (хотя для некоторых страниц и такая практика работает просто отлично). Стандартный подход при этом — сначала разработать дизайн для узких экранов, а затем расширять окно браузера и подмечать те моменты, когда каждая из частей страницы становится неприемлемой. Причинить некоторое неудобство при этом могут элементы навигации, способные потребовать организации точки прерывания при ширине, равной 400 пикселам, однако одноколоночный макет может отлично отображаться и до тех пор, пока ширина не достигнет 800 пикселов, после чего можно уже переходить к двухколоночному дизайну.

В книге «Responsive Design: Patterns & Principles» Итан Маркотт (Ethan Marcotte) излагает свой подход к процессу «выхода из контента» следующим образом:

Для меня этот процесс «выхода из контента» начинается с просмотра наименьшей версии фрагмента контента, а затем расширения этого элемента до тех пор, пока не начнут отображаться его ивы и он не станет терять свою форму. Как только это произойдет, можно вносить исправление — ввести точку останова, которая изменяет элемент и сохраняет его целостность.

Если вы обнаружите, что некоторое число возможных точек прерывания отстоят друг от друга всего лишь на несколько пикселов или единиц em, объедините их — это упростит таблицу стилей и весь процесс адаптации в целом. Кроме того, не мешает учитывать размеры экранов наиболее популярных устройств на тот случай, если небольшое снижение точки прерывания поможет улучшить работу целого класса пользователей. Размеры наиболее популярных устройств приведены на сайте Screen Sizes (screenizes.es). Вы также можете с помощью веб-поиска найти и другие аналогичные ресурсы.

Точки прерывания, основанные на использовании единиц em

Приведенные в этом разделе примеры до сих пор были основаны на точках прерывания с размерами, заданными в пикселях. Альтернативный метод, а многие считают

НЕ ИЗМЕНЯЙТЕ ВСЮ СТРАНИЦУ СРАЗУ

Принято создавать точки прерывания для каждого компонента страницы, а не изменять всю страницу сразу.

его лучшим, состоит в использовании в медиазапросе вместо пикселов единиц измерения *em*. Напомню, что единица измерения *em* равна текущему размеру шрифта элемента. При использовании в медиазапросе единица измерения *em* основывается на базовом размере шрифта для документа (по умолчанию 16 пикселов, хотя он может быть изменен пользователем или автором страницы).

Следует отметить, что пиксельные мультимедийные запросы не адаптируются, если пользователь изменяет настройки размера шрифта, как это часто делается для

МЕДИАЗАПРОСЫ НА ОСНОВЕ *em* БОЛЕЕ АДАПТИВНЫ

*Медиазапросы на основе *em* поддерживают макет, пропорциональный размеру шрифта.*

удобства при чтении страницы. А вот основанные на использовании единиц измерения *em* медиазапросы реагируют на размер текста, сохраняя пропорции макета страницы.

Рассмотрим, например, макет, который переключается на две колонки, когда ширина страницы достигает 800 пикселов. При этом вы разработали его с тем, чтобы основной столбец имел оптимальную длину строки текста, когда базовый размер шрифта по умолчанию составляет 16 пикселов. Если пользователь увеличит размер своего шрифта до 32 пикселов, вдвое увеличенный текст должен будет уместиться в пространстве, предназначенном для текста вдвое меньшего размера. Можете себе представить, как это отразится на длине строк.

При использовании запросов на основе *em*, нацеленных на браузеры шире, чем 50*em*, и базовом размере шрифта, равном 16 пикселам, переключение на двухколоночный макет — как и было задумано — происходит при 800 пикселях. Однако когда базовый размер шрифта увеличится до 32 пикселов, двухколоночный макет вырастет до 1600 пикселов ($50\text{em} \times 32\text{px} = 1600\text{px}$), и у текста окажется достаточно места для заполнения основного столбца той же длиной строки, что и при исходном дизайне.

В этом примере имелось в виду, что переключению подвергается макет всей страницы, но рассмотренные ранее медиазапросы к отдельным компонентам также могут применять единицы измерения *em*. В следующем разделе я расскажу вам о некоторых аспектах веб-страниц, которые при выборе точек прерывания требуют дополнительного внимания.

СТРАТЕГИИ АДАПТИВНОЙ РАЗРАБОТКИ

К этому моменту мы рассмотрели основные моменты АВД — теперь давайте поболтаем о некоторых решениях, принимаемых дизайнёрами при создании адаптивных сайтов. Здесь я смогу только «поцарапать поверхность», но вы найдёте гораздо более глубокие исследования по этим темам в книгах и статьях, упоминаемых мной по мере изложения материала. Сейчас же я просто хочу немного поднять вашу осведомленность об адаптивных стратегиях. В конце этого раздела вы сможете использовать некоторые из них, чтобы сделать страницу «Black Goose Bakery» более отзывчивой.

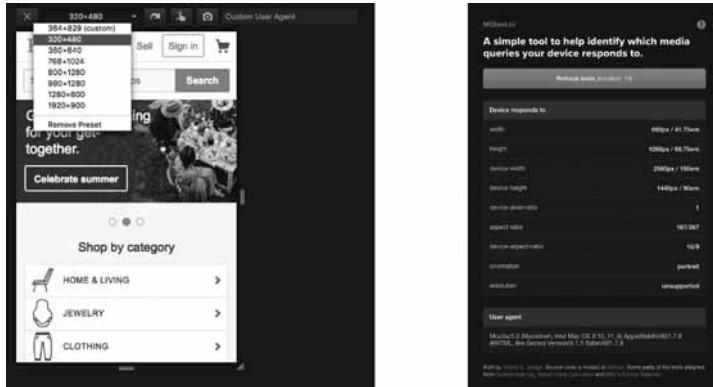
НАСКОЛЬКО ШИРОКА ОБЛАСТЬ ПРОСМОТРА?

Я предлагаю вам развернуть окно своего браузера настолько широко, пока вы не почувствуете, что необходима точка прерывания, но как узнать, сколько это конкретно? Для этого существует ряд инструментов, предоставляющих данные о размере области просмотра.

Браузеры Firefox, Chrome и Safari содержат инструменты, которые показывают, какой вид имеет страница при определенных размерах области (окна) просмотра. В режиме адаптивного дизайна в области просмотра откроется панель, из которой вы сможете задать размер этой области с учетом стандартных размеров устройства или же вручную. При изменении размеров области просмотра в этой панели будут отображаться такие размеры в пикселях.

В Firefox в режим адаптивного дизайна (рис. 17.7, слева) можно перейти, выбрав команды Tools | Web Developer | Responsive Design Mode (Инструменты | Веб-разработка | Адаптивный дизайн). Режим адаптивного дизайна Safari доступен по команде Develop | Enter Responsive Design Mode (Рабочий стол | Войти в режим адаптивного дизайна). Браузер Chrome предлагает панель инструментов, для открытия которой надо выбрать команды View | Developer | Developer Tools (Вид | Разработчик | Инструменты разработчика), а затем найти значок Toggle Device Toolbar (Переключить панель инструментов устройства), который находится в строке меню слева. Все эти инструменты работают примерно одинаково, но вам может показаться, что пользовательский интерфейс одного браузера предпочтительнее другого.

Чтобы узнать, каким образом ваше окно браузера или устройство реагирует на медиазапросы, зайдите на сайт MQTest.io (mqtest.io) от Вильями Салминена (Viljami Salminen). В дополнение к ширине и высоте области просмотра здесь сообщается о других функциях устройства, таких, как соотношение устройств и пикселей, соотношение сторон и многое другое (рис. 17.7, справа).



Режим адаптивного (отзывчивого) дизайна в Firefox показывает точные размеры в пикселях для области просмотра. Он включает сокращенные свойства, предназначенные для изменения его размера с учетом общих размеров устройства. Браузеры Chrome и Safari имеют схожие адаптивные возможности.

MQTest.io — это веб-страница, которая сообщает, каким образом ваш браузер реагирует на медиазапросы, и показывает ширину области просмотра.

Рис. 17.7. Проверка размеров окна просмотра в режиме адаптивного дизайна Firefox (слева) и на сайте MQTest.io (справа)

В некоторых предыдущих примерах мы рассматривали ситуации, когда окно браузера становится слишком узким или очень широким. Макет из трех столбцов просто не поместится, да и текст в изображении может оказаться нечитаемым, если его уменьшить до размера экрана, равного 320 пикселям. С другой стороны, строка в макетах с одним столбцом может стать настолько длинной, что это воспрепятствует удобству чтения, особенно когда область просмотра распространяется на всю ширину настольного монитора с высоким разрешением. То есть один размер веб-страницы для многих ситуаций подходит не всегда. И нам, дизайнерам, следует обращать внимание на подобные особенности и устанавливать точки прерывания для «сохранения целостности» элементов (как об этом хорошо пишет Итан Маркотт (Ethan Marcotte)).

В самых общих подходах оптимизация для широкого диапазона размеров области просмотра включает в себя следующие понятия:

- иерархия контента;
- макет;
- типографика;
- навигация;
- изображения;
- специальный контент — такой, как таблицы, формы и интерактивные функции.

Иерархия контента

Контент имеет первостепенное значение в Интернете, поэтому важно тщательно продумывать и организовывать его, прежде чем писать код. Это задачи для создателей информации и контент-стратегов, которые должны решать проблемы организации, маркировки, планирования веб-контента и управления им.

Организация и построение иерархии для различных видов сайта являются первостепенными задачами с особым акцентом на работу с маленькими экранами. Лучше всего начать с анализа потенциального контента и вычленить из него то, что может стать наиболее полезным и важным для просмотра всего сайта. Как только вы разберетесь, как ваш контент можно разделить на части (модули), соответствующие возможным точкам прерывания, вы сможете решить, в каком порядке они будут отображаться на экранах разных размеров.

Помните, что следует стремиться к *паритету контента* (*content parity*), то есть к пониманию, что один и тот же контент должен быть доступен всем посетителям, независимо от того, с какого устройства они заходят на сайт. Может случиться так, что некоторым вашим посетителям придется следовать к нужной им информации по несколько иному навигационному пути, но не показывать какие-либо части вашего сайта посетителям с маленькими экранами, только потому что вы думаете, будто пользователям мобильных устройств они не нужны, в корне неверно. Кстати, пользователи в процессе решения своих задач могут менять устройства, и им следует дать уверенность, что у них это получится.

Я привела здесь весьма краткое введение в тему иерархии контента, хотя, возможно, это наиболее важный первый шаг при создании сайта, но подробное его рассмотрение выходит за рамки этой книги. Для правильного освоения контент-стратегии, особенно в той его части, что касается АВД, я рекомендую следующие книги:

- «Content Strategy for the Web», 2-е издание, (издательство New Riders), авторы Кристина Халвортон (Kristina Halvorson) и Мелисса Рич (Melissa Rich);
- «Content Strategy for Mobile» от Карен Макгрейн (Karen McGrane).

Макет

Перекомпоновка контента в разные макеты может быть первым шагом, который приходит вам на ум, когда представляете себе, как может выглядеть адаптивный дизайн, и на то есть веская причина. Макет помогает сформировать наше первое впечатление о содержании и удобстве использования сайта.

Как упоминалось ранее, адаптивный (отзывчивый) дизайн основывается на текущих макетах, которые растягиваются и сжимаются для заполнения доступного пространства в областях просмотра. Однако одного текущего макета недостаточно для обслуживания экранов всех размеров. Более часто, чтобы удовлетворить требования различных устройств, разрабатываются два или три макета с небольшой подгонкой при смене макетов.

Обычно дизайнеры начинают с одноколоночного макета, который хорошо подходит для небольших портативных устройств, и перекомпоновывают элементы в большее число колонок, когда становится доступно больше места. У них также могут иметься макеты для самых широких экранов, разработанные на ранних стадиях дизайна, так что им остается лишь проработать промежуточные варианты. В процессе проектирования они периодически переключаются между представлениями макетов и принимают соответствующие решения.

УСЛОВНАЯ ЗАГРУЗКА

Паритет контента не означает, что весь размещаемый на большом экране контент должен быть вставлен и в макет маленького экрана. Сопутствующие этому варианты прокрутки и докачки дополнительных данных мобильным пользователям только усложняют работу.

Лучшим подходом является использование *условной загрузки*, при которой пользователи устройств с небольшими экранами получают наиболее важный контент со ссылками для доступа к дополнительному контенту (комментариям, сведениям о продукте, объявлениям, спискам ссылок и т. д.), если они этого пожелают. Таким образом, им оказывается доступна вся информация, только не сразу. Между тем на больших экранах подобные дополнительные части контента автоматически отображаются на боковых панелях (врезках).

Для реализации условной загрузки необходим JavaScript, поэтому я не стану приводить здесь конкретные инструкции, но вам надо четко усвоить, что для встраивания каждой мелочи в каждое из устройств существуют альтернативные варианты.

Макет и длина строки

Хорошим поводом для принятия решения о пригодности макета является длина текстовых строк. Слишком короткие или слишком длинные текстовые строки трудно читать, поэтому вам следует стремиться к оптимальной длине строки от 45 до 75 символов, включая пробелы. Если текстовые строки получаются значительно длиннее, необходимо внести в макет изменения — например, увеличить поля или ввести дополнительную колонку. Можно также увеличить размер шрифта текста, чтобы количество символов оставалось в желаемом диапазоне.

Кларисса Петерсон (Clarissa Peterson) книге «Learning Responsive Web Design» (O'Reilly) представляет изящный прием для тестирования длин строк. Выделите в тексте диапазон от 45 до 75 символов и задайте ему фоновый цвет (рис. 17.8). Это даст вам возможность проверить, происходят ли разрывы строк в безопасной зоне. Конечно, эту подсказку длины строки надо не забыть удалить до того, как сайт станет общедоступным.

In this book, Philadelphia Ice Creams, comprising the first group, are very palatable, but expensive. In many parts of the country it is quite difficult to get good cream. For that reason, I have given a group of creams, using part milk and part cream, but it must be remembered that it takes smart "juggling" to make ice cream from milk. By far better use condensed milk, with enough water or milk to rinse out the cans.

Рис. 17.8. Выделите диапазон от 45 до 75 символов, чтобы сразу определить оптимальную длину строки

Шаблоны адаптивных макетов

Способ, которым сайт переходит от макета с экраном небольшим к макету с экраном широким, относится, разумеется, только к конкретному сайту, но есть несколько шаблонов (общий и повторные попытки), которые появились в последние годы. Мы можем поблагодарить Люка Вроблевски (Luke Wroblewski) (известен как автор подхода к веб-дизайну «Mobile First», который стал стандартом) за его обзор способов обработки макетов адаптивными сайтами. Статья «Multi-Device Layout Patterns» (www.lukew.com/ff/entry.asp?1514), где подробно изложены его выводы, появилась давно, но шаблоны сохраняются. Далее представлены основные шаблоны, описанные в статье Люка (рис. ЦВ-17.9):

- **сильно текущий** — шаблон использует одноколоночный макет для небольших экранов и текучие макеты, охватывающие средние и большие экраны, с ограничением максимальной ширины, установленным для предотвращения их слишком большого растяжения. В целом это не столь затратно по усилиям, чем другие решения;
- **выпадение столбца** — решение заключается в переключении между макетами с одним, двумя и тремя столбцами в зависимости от доступного пространства. Когда нет места для дополнительных столбцов, столбцы боковой панели опускаются ниже других столбцов, до тех пор пока все не будет правильно расположено в виде одного столбца;

- **сдвиг макета** — если вы хотите прослыть реально модным, то можете изобрести собственный макет для экранов различных размеров. Хотя это выразительно и потенциально круто, но, тем не менее, не обязательно. В общем и в целом, вы можете решить проблему подгонки вашего контента к нескольким средам, не выходя за рамки разумного;
- **небольшая настройка** — некоторые сайты используют одноколоночный макет и настраивают тип, интервалы и изображения, принуждая его работать с устройствами различных размеров;
- **вне холста** — в качестве альтернативы вертикальному размещению контента на небольших экранах вы можете использовать решение «вне холста». В этом шаблоне компонент страницы расположен слева или справа от экрана и отображается по запросу. То есть часть экрана основного контента остается видимой на краю, чтобы ориентировать пользователей относительно взаимосвязи движущихся частей страницы. Это стало популярным в Facebook, где «Избранное» и «Настройки» размещались на панели, которая выезжала слева, когда пользователи нажимали соответствующий значок меню.

Рабочие примеры этих и других шаблонов макетов можно увидеть на странице «Responsive Patterns», представленной Брэдом Фростом (Brad Frost) на сайте по адресу: bradfrost.github.io/this-is-responsive/patterns.html.

Типографика

Типографика, чтобы сделать контент хорошо читаемым и приятно выглядящим, требует тонкой настройки параметров контента как для небольших экранов, так и для широкоэкраных представлений. Вот несколько примеров параметров контента для типографики (рис. 17.10):

- **гарнитура шрифта** — будьте осторожны с использованием на маленьких экранах причудливых шрифтов и обязательно проверяйте их разборчивость. При уменьшении размеров некоторые шрифты оказываются трудными для чтения, поскольку штрихи их символов становятся слишком тонкими, а украшательские завитки превращаются в некое подобие простых точек. Учтите также, что устройства с небольшими экранами могут подключаться к сети по мобильной связи, поэтому использование локально доступных шрифтов может быть в плане производительности лучше, чем загрузка веб-шрифтов. Если строгие требования идентификации бренда требуют согласованности шрифтов на всех устройствах, обязательно выберите шрифт, который подходит для всех размеров. Если это возможно, используйте веб-шрифты только на больших экранах. Мы, конечно, стремимся обеспечить одинаковый дизайн для всех устройств, но, как и для всего остального в веб-дизайне, гибкость важна не меньше;
- **размер шрифта** — изменение ширины области просмотра может повлиять на длину строк. Поэтому в ряде случаев вам придется увеличивать размер шрифта текстовых элементов для более широких областей просмотра, чтобы поддержать длину строки от 45 до 75 символов. Это также облегчает чтение с расстояния, на котором пользователи обычно сидят перед своими большими мониторами. И

наоборот, вы можете применять медиазапросы на основе `em`, чтобы макет оставался пропорциональным размеру шрифта. Напомню, что при запросах на основе `em` длина строк остается неизменной;

- **высота строк** — это еще один размер, который вы можете подстраивать по мере увеличения размеров экрана. В среднем, высота строки должна быть около 1,5 (имеется в виду числовое значение для свойства `line-height`), однако немного меньший межстрочный интервал (от 1,2 до 1,5) легче читать при наличии более коротких строк на маленьких экранах. Для широких экранов, где шрифт может быть и крупнее, можно устанавливать и большую высоту строк (от 1,4 до 1,6);
- **поля** — на маленьких экранах максимально используйте доступное пространство, сводя к минимуму левые и правые поля в главном столбце (2–4%). По мере увеличения размеров экранов вам, вероятно, придется увеличивать боковые поля, чтобы управлять длиной строк или просто добавлять в макет желаемое пространство. Не забудьте также указать поля над и под текстовыми элементами в единицах `em`, чтобы они оставались пропорциональными шрифту.

Philadelphia Ice Creams

Comprising the first group, are very palatable, but expensive. In many parts of the country it is quite difficult to get good cream. For that reason, I have given a group of creams, using part milk and part cream, but it must be remembered that it takes smart "juggling" to make ice cream from milk. By far better use condensed milk, with enough water or milk to rinse out the cans.

Ordinary fruit creams may be made with condensed milk at a cost of about fifteen cents a quart, which, of course, is cheaper than ordinary milk and cream. The cream for Philadelphia Ice Cream should be rather rich, but not double cream.

If pure raw cream is stirred rapidly, it swells and becomes frothy, like the beaten whites of eggs, and is "whipped cream." To prevent this in making Philadelphia Ice Cream, see half

Philadelphia Ice Creams

Comprising the first group, are very palatable, but expensive. In many parts of the country it is quite difficult to get good cream. For that reason, I have given a group of creams, using part milk and part cream, but it must be remembered that it takes smart "juggling" to make ice cream from milk. By far better use condensed milk, with enough water or milk to rinse out the cans.

Ordinary fruit creams may be made with condensed milk at a cost of about fifteen cents a quart, which, of course, is cheaper than ordinary milk and cream. The cream for Philadelphia Ice Cream should be rather rich, but not double cream.

Узкие экраны:

- Разборчивые шрифты
- Меньший размер шрифта
- Меньшая высота строки
- Узкие поля

Широкие экраны:

- Стильные шрифты
- Большой размер шрифта
- Большая высота строки
- Более широкие поля

Рис. 17.10. Общие советы по типографике для маленьких и больших экранов

ПЕРЕМЕННЫЕ ШРИФТЫ

В конце 2016-го года OpenType разработала новую технологию шрифтов под названием OpenType Font Variations, широко известную как «переменные шрифты». Используя свойства стиля `font-*`, вы можете изменять жирность, ширину, стиль (курсив), наклон и оптический размер переменного шрифта. Отличительный момент этой технологии в том, что можно доставить один файл шрифта (это всего лишь один вызов серверу), а затем манипулировать им в своих целях — например, сделать его уже, чтобы поддержать высоту и длину строк на маленьких экранах. Поддержка браузерами переменных шрифтов должна была появиться уже в 2018-м году. Для получения дополнительной информации обратитесь к статье «Get Started with Variable Fonts» Ричарда Раттера (Richard Rutter), которая доступна по адресу: medium.com/@clagnut/get-started-withvariable-fonts-c055fd73ecd7.

Сайт axis-praxis.org позволяет поэкспериментировать с переменными шрифтами, используя движки для регулировки его жирности и других свойств. Обратите внимание, что для этого необходим браузер, поддерживающий обработку переменных шрифтов.

Навигация

Навигация немного напоминает Святой Грааль адаптивного веб-дизайна. И очень важно организовать ее правильно. Поскольку проблема навигации на широких экранах в значительной степени решена, реальные проблемы возникают лишь при реорганизации параметров навигации для небольших экранов. И как раз для них появился ряд удачных шаблонов (рис. ЦВ-17.11), которые я кратко представлю здесь:

- *панель навигации в верхней части экрана* — если на вашем сайте всего несколько навигационных ссылок, они вполне могут поместиться в одну или две строки в верхней части экрана;
- *приоритет +* — в этом шаблоне наиболее важные навигационные ссылки отображаются в строке, расположенной в верхней части экрана рядом со ссылкой **More** (Еще), которая открывает дополнительные, скрытые пока ссылки. Преимущество здесь в том, что основные ссылки отображаются в строке меню явно и общее количество отображаемых в ней ссылок может расти с увеличением ширины экрана. Отрицательным моментом при этом является сложность выбора, какие именно ссылки для небольшого экрана наиболее важны;
- *форма ввода select* — для среднего размера списка ссылок некоторые сайты используют элемент формы ввода *select*. При выборе такого меню список его пунктов открывается с помощью пользовательского интерфейса выбора операционной системы в виде прокручиваемого списка ссылок внизу экрана или наложенного на экран. Преимущество его в том, что он компактен, но, с другой стороны, формы обычно не используются для навигации, и меню может быть пропущено;
- *связь с меню в колонтитуле* — в этом несложном варианте ссылка **Menu** (Меню) помещается в верхней части страницы и связывается с расположенной в нижней части страницы полной системой навигации. Риск работы с этим шаблоном заключается в том, что он может дезориентировать пользователей, которые внезапно оказываются в нижней части области прокрутки;

ТЕКУЧАЯ ТИПОГРАФИКА НА ОСНОВЕ ЕДИНИЦ ИЗМЕРЕНИЯ ОБЛАСТИ ПРОСМОТРА

Для поддержки размера текста пропорционально размеру области просмотра применяйте значения длины в процентах области просмотра: *vw* и *vh* для *font-size*. Одна единица *vw* (ширина области просмотра) равна 1% от ширины области просмотра (или «начального содержащего блока», как его называют в спецификации). Одна единица *vh* составляет 1% от высоты области просмотра. Спецификация также определяет единицы *vmin* (меньшее от *vw* или *vh*) и *vmax* (большее от *vw* или *vh*), но эти единицы измерения недостаточно хорошо поддерживаются.

Поддержка браузерами единиц *vw* и *vh* весьма уверенная, за исключением браузера IE9 и более ранних его версий и поддержки *vmin* и *vmax*. Впрочем, при использовании этих единиц наблюдается достаточно много распространенных ошибок, поэтому обязательно проверьте для этих значений вкладку **Known Issues** (Известные проблемы) на странице CanIUse.com. И ознакомьтесь с применением единиц просмотра в тексте, обратившись к статье Майкла Ритмюллера (Michael Riethmüller) «*Responsive Font Size and Fluid Typography with vh and vw*» в журнале Smashing Magazine (www.smashingmagazine.com/2016/05/fluid-typography).

- **аккордеонная субнавигация** — при наличии большого числа навигационных ссылок, да еще с вложенными подменю субнавигации решение для небольшого экрана становится весьма сложным, особенно когда у вас нет возможности точного наведения на ссылку, которую давала бы мышь. Для раскрытия и скрытия такой субнавигации иногда прибегают к организации системы вложенных подменю, распахивающихся как меха аккордеона по нажатию на маленькую стрелку в пункте основного навигационного меню. При этом вложенность подменю может достигать нескольких уровней. Чтобы избежать вложенной навигации в аккордеонных подменю, некоторые сайты просто ссылаются на отдельные целевые страницы, которые содержат список субнавигации каждого раздела;
- **переключатели сдвигающие и наложения** — обычно содержимое навигационного меню скрыто, но раскрывается вниз при нажатии на значок меню. При этом оно может сдвинуть основной контент страницы вниз (push-переключатель) или наложитьсь на него (overlay-переключатель);
- **вне холста/вылетающее** — этот популярный шаблон размещает навигационные ссылки на панели, находящейся за пределами экрана слева или справа от основного содержимого страницы и отображающейся (как бы вылетающей) при нажатии на значок меню.

Работающие примеры этих шаблонов с кодом, использованным для их создания, приведены на странице «Adventures in Responsive Navigation», представленной Эриком Арбе (Eric Arbé) по адресу: responsivenavigation.net.

Для более основательного ознакомления с плюсами и минусами навигационных шаблонов прочтайте статью Брэда Фроста (Brad Frost) «Responsive Navigation Patterns» (bradfrost.com/blog/web/responsive-nav-patterns). Брэд также приводит примеры шаблонов и многое другое на своей странице «Responsive Patterns» (bradfrost.github.io/this-is-responsive/patterns.html).

Изображения

Изображения требуют особого внимания в адаптивном дизайне. Далее кратко излагаются некоторые ключевые моменты, большинство из которых звучит уже знакомо:

- используйте адаптивные методы разметки изображений (описанные в главе 7), чтобы предоставить несколько версий ключевых изображений для различных размеров и разрешений экранов;
- используйте наименьшую версию из заданных по умолчанию, чтобы избежать ненужной загрузки данных;
- убедитесь, что важные детали изображения не теряются при его уменьшенном размере. Попробуйте представить для маленьких экранов уменьшенную, но правильную, версию изображения взамен версии, утрачивающей важные детали;
- старайтесь не помещать текст в графику, но, если это необходимо, предоставьте для небольших экранов альтернативные версии изображений с увеличенным текстом.

Специальный контент

Нормально выглядящие на широких экранах настольных компьютеров, некоторые из общих элементов страницы становятся проблемными при размещении их на экранах небольшого размера:

- **формы** — пытаясь заполнить все доступное пространство, формы нередко создают дизайнерам проблемы. Отличным инструментом для добавления формам гибкости является Flexbox. Поискав в Сети, вы найдете много полезных советов на этот счет. Также убедитесь, что ваша форма максимально удобна и не имеет лишних полей, что является хорошим подходом для любого размера экрана. Наконец, учтите, что для ввода данных в форму могут использоваться кончики пальцев, а не указатели мыши, поэтому увеличьте ее размер, добавив достаточные отступы или поля и сделав метки, которых можно коснуться, чтобы начать ввод;
- **таблицы** — обработка больших таблиц с данными представляет собой одну из самых больших проблем в дизайне для малых экранов. Поэтому неудивительно, что при большом числе типов таблиц также имеется множество решений по их обработке и представлению (см. врезку «Проблема с таблицами» для получения вспомогательной информации и сведений о дополнительных ресурсах);
- **интерактивные элементы** — большая встроенная карта может быть хороша при просмотре сайта на широком экране настольного компьютера, но она не столь полезна, если имеет размер почтовой марки. Подумайте, не следует ли заменить некоторые интерактивные функции другими методами для выполнения той же задачи. В случае с картой добавление ссылки на карту может привести к открытию на устройстве собственного приложения для работы с картами, которое обеспечит лучшее восприятие карты на маленьком экране. Впрочем, другие интерактивные компоненты — такие, как, например, карусели, вполне могут быть адаптированы для небольших областей просмотра.

ДИЗАЙН ДЛЯ ПАЛЬЦЕВ

Имейте в виду, что люди в наши дни используют пальцы для управления сенсорными устройствами: смартфонами, планшетами и даже планшетами-трансформерами с экранами больших размеров, такими, как Microsoft Surface и iPad Pro.

Навигационные ссылки на сенсорных экранах должны быть достаточно крупными, чтобы было легко попадать в них кончиками пальцев. Apple требует для своих приложений 44 пикселя, и это хороший пример для ссылок на веб-страницах.

Еще одной особенностью навигации на сенсорных устройствах является отсутствие у них привычного состояния наведения, которое мы получаем на обычных компьютерах, когда подводим мышь к ссылке. Состояние наведения определено специальным соглашением, оговаривающим порядок субнавигации на веб-страницах, открытых на настольных компьютерах, но без мыши этот опыт сильно отличается от сенсорного, — ведь большинство компьютеров настроены так, что открывают подменю по двойному щелчку. Если для навигации, а также в других местах на сайте используется наведение, необходимо провести тщательное тестирование применяемых устройств. Когда-нибудь можно будет написать медиазапрос для проверки наведения, но в настоящее время следует либо избегать его, либо искать альтернативы.

Есть замечательная книга, посвященная этим вопросам, — «Designing for Touch» Джоша Кларка (Josh Clark).

ПРОБЛЕМА С ТАБЛИЦАМИ

Большие таблицы, наподобие показанных на рис. 8.1, могут быть трудны для показа на устройствах с маленькими экранами. По умолчанию они сжимаются до ширины экрана, что делает текст в ячейках слишком мелким для чтения. Пользователи могут увеличивать масштаб отображения, чтобы прочитать этот текст, но тогда одновременно им станут видны только несколько ближайших ячеек, и им сложно будет разобраться в организации заголовков и столбцов.

Дизайнеры и разработчики предложили ряд подходов для адаптации таблиц. Честно говоря, использование таблиц на небольших устройствах все еще является относительно новым явлением, поэтому сейчас осуществляется большое число экспериментов, чтобы понять, что получится. Большинство решений включают столь продвинутые модели веб-разработки, что для их описания потребовалось бы значительно больше места, чем можно здесь предоставить, однако мне все же хочется, хотя бы конспективно, ознакомить вас с адаптивными таблицами. Итак, существуют три основных подхода: прокрутка, укладка и скрытие.

При выборе решений с прокруткой таблица остается настолько широкой, насколько это необходимо, и пользователи могут прокручивать ее вправо, чтобы увидеть дополнительные столбцы. Это может быть достигнуто только с помощью JavaScript или CSS. Вы можете даже привязать левый столбец к окну, чтобы он оставался на месте, когда прокручивается остальная часть таблицы.

Другой подход состоит в том, чтобы сложить записи в длинный узкий свиток. В каждой записи повторяются заголовки, поэтому данные всегда представляются в соответствующем контексте. Опять же, это можно сделать только с помощью JavaScript или CSS. Недостаток этого подхода заключается в том, что свиток может оказаться очень длинным, что затрудняет сравнение записей, но, по крайней мере, вся информация отображается без горизонтальной прокрутки.

Можно также скрыть определенные столбцы информации при первоначальной загрузке таблицы на небольшие устройства и дать пользователю возможность касанием экрана просматривать всю таблицу или включать или отключать определенные столбцы. Это немного более рискованно с точки зрения организации взаимодействия, поскольку такие столбцы просто могут быть не видны вообще.

Использование таблиц CSS и Flexbox — это другие варианты того, чтобы сделать табличный материал адаптивным. Лучший подход полностью зависит от типа публикуемых данных и ожидаемого способа использования таблицы. Если вы заинтересованы в получении дополнительной информации, рекомендую вам следующие ресурсы:

- Дэвиде Риццо (Davide Rizzo) о CSS-трюках: «Accessible, Simple, Responsive Tables» (css-tricks.com/accessible-simple-responsivatables/) — сводка решений с использованием CSS-таблиц;
- Дэвид Бушелл (David Bushell): «CSS-only Responsive Tables» (dbushell.com/2016/03/04/css-only-responsive-tables/) — подход к прокрутке только для CSS с использованием теней CSS для повышения удобства при использовании;
- Джейсон Григсби (Jason Grigsby) из Cloud Four: «Picking a Responsive Tables Solution» (cloudfour.com/thinks/picking-responsivatables-solution/);
- Адаптивные таблицы от ZURB Studios (zurb.com/playground/responsive-tables) — решение с прокруткой и фиксации левого столбца с использованием JavaScript и CSS;
- Tablesaw от Filament Group (github.com/filamentgroup/tablesw) — группа плагинов JQuery (JavaScript), предназначенная для создания различных адаптивных табличных эффектов.

Представленный в этом разделе материал должен дать вам представление о некоторых аспектах сайта, которые требуют особого внимания в плане адаптивного дизайна. Мы рассмотрели иерархию контента, различные шаблоны макетов, настройки типографики, адаптивные шаблоны навигации и стратегии изображений, а также таблицы, формы и интерактивные элементы. Полагаю, что лекций достаточно. Пришло время перейти к практическому применению полученных знаний, выполнив упражнение 17.1.

УПРАЖНЕНИЕ 17.1. ДЕЛАЕМ ДОМАШНЮЮ СТРАНИЦУ «BLACK GOOSE BAKERY» АДАПТИВНОЙ

В упражнениях последних нескольких глав мы уделили сайту «Black Goose Bakery» значительное внимание, но получившийся сайт лучше всего работает на больших экранах. В этом упражнении мы собираемся сделать несколько шагов назад и сформировать его заново на основе стратегии «сначала на маленьком экране», внося изменения в макет, навигацию, типографику и многое другое в стратегических точках прерывания.

Я проделала большую работу по созданию необходимых стилей для каждой точки прерывания и расскажу вам о каждом своем шаге, а также поделюсь обоснованием проводимых изменений. Исходную таблицу стилей (`bakery-rwd.css`), а также финальную таблицу стилей (`bakery-rwd-finished.css`) и другие необходимые для работы файлы вы найдете в материалах для этой главы. Файл HTML-файл `bakery.html` не изменился, поскольку элемент контейнера мы добавили в него в главе 16, и необходимости что-либо в нем редактировать нет.

Начинаем

Откройте HTML-файл (`bakery.html`) в браузере, поддерживающем режим адаптивного дизайна (см. ранее приведенную врезку «*Насколько широка область просмотра?*»), чтобы можно было развернуть окно области просмотра и наблюдать за изменением размеров, заданных в пикселях. На рис. 17.12 показана страница шириной 320 пикселов со стандартными для узких экранов стилями, которые станут отправной точкой для нашей работы.

Контент страницы аналогичен тому, который присутствовал в ней в предыдущих главах, но если вы работали над упражнениями главы 16, то должны заметить, что я изменила несколько стилей, чтобы сделать этот исходный макет подходящим для небольших экранов.

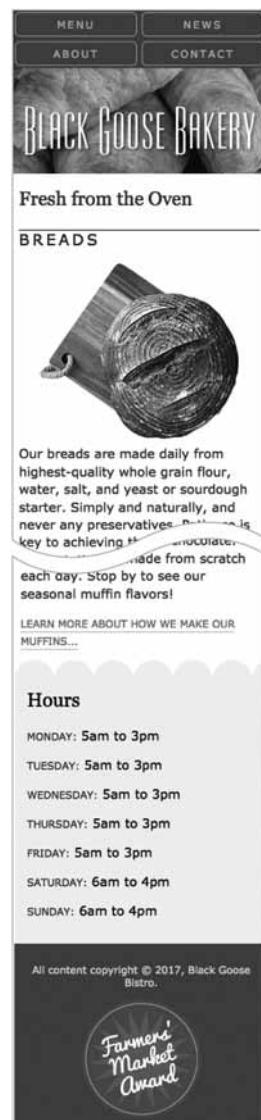


Рис. 17.12. Дизайн для маленьких экранов — наша отправная точка

Позвольте мне пояснить новые характеристики этого базового дизайна:

- **макет** — страница организована в одну колонку и предназначена для маленьких экранов. Вокруг основной текстовой области нет границ, а в разделе **Hours** фестончатый край сделан сверху, а не сбоку слева. Это придавало ему некоторый шарм, но более уместно, когда разделы накладываются друг на друга;
- **навигация** — меню навигации, созданное с помощью Flexbox, не было достаточно маленьким, чтобы поместиться на узком экране. Чтобы его уместить, включена разбивка на ряды (`flex-wrap: wrap;`), а также установлена ширина для каждого элемента `li`, равная 50%, в результате чего они стали размещаться по два таких элемента в каждом ряду. Я также придала им возможность увеличиваться и уменьшаться по мере необходимости (`flex: 1 1 50%`);
- **подзаголовок** — слоган занимал много пространства по вертикали, и я решила, что без него можно обойтись. Поэтому я пока спрятала этот абзац (`display: none;`), и сделаю его снова видимым, когда появится больше места;
- **тиографика** — для текста на маленьком экране я использовала разборчивый бессерифный шрифт, а не собственный веб-шрифт, поскольку его, вероятно, было бы трудно читать на таком экране;
- **изображения** — элементы `img` для изображений хлеба и маффина я установила равными `display: block`, чтобы они занимали всю ширину области просмотра без какого-либо окружающего их текста;
- **прочее:**
 - изображение награды отображается в нижней части страницы, поскольку в верхней для ее размещения недостаточно места;
 - Я подсветила диапазон (`span`) от 45 до 75 символов, чтобы заметить, когда строка становится слишком длинной.

Доработка навигационной панели

Теперь мы можем начавть адаптацию этого дизайна под другие размеры экрана. Используя инструмент Responsive View, я могу изменять размеры области просмотра и мгновенно получать данные о размерах окна. Попробуйте отобразить это в окне своего браузера. Продолжайте делать окно шире, и вы увидите, что некоторые объекты выглядят хорошо, а некоторые достаточно быстро начинают выглядеть некрасиво.

Один из таких объектов — это сложенная вдвое навигационная панель вверху экрана. Мне бы хотелось, чтобы она переключалась на одну центрированную линию, как только для нее появится достаточно места, что, на мой взгляд, происходит, когда область просмотра получает ширину 400 пикселов (рис. 17.13).



До точки прерывания



После

Рис. 17.13. Навигационная панель выглядит некрасиво (вверху), поэтому в дизайн добавлена точка прерывания, обеспечивающая по достижении областью просмотра ширины 400 пикселов переключение этой панели в одну линию (внизу)

Готовы ли вы создать свой первый медиазапрос? Откройте таблицу стилей (`bakery-rwd.css`) в текстовом редакторе. Помните, что медиазапросы должны размещаться после других правил для тех же объявлений, поэтому, чтобы не усложнять наше упражнение, добавим их в конец таблицы стилей перед тегом `</style>`. Добавьте этот запрос, как показано здесь. Убедитесь, что используется корректное число вложенных фигурных скобок:

```
@media screen and (min-width: 400px) {  
    nav ul li {  
        flex: none;  
    }  
    nav ul {  
        justify-content: center;  
    }  
}
```

Этот код предпишет браузеру следующее: если страница находится на экране и область просмотра составляет 400 пикселов или шире, установить `flex` для элементов списка меню равным `none`. Ключевое слово `none` является эквивалентом записи `flex: 0 0 auto;`, то есть элементам не позволяет расти или сжиматься и приобретать размер на базе их контента. Контейнер `flexbox` я центрировала с помощью настройки `justify-content: center`.

Сохраните таблицу стилей и перезагрузите страницу в окне браузера. Попробуйте изменить размер окна просмотра, чтобы увидеть, как это работает при более широких размерах. Я полагаю, что подобное центрированное расположение подойдет даже для самых широких экранов, поэтому навигацию можно считать настроенной. Если же у вас была навигация с дополнительными элементами, такими, как встроенный логотип и окно поиска, вам может быть лучше создать несколько различных механизмов для нескольких точек прерывания.

Плавающие изображения

По мере постепенного расширения области просмотра я замечую, что основные изображения начинают выглядеть одиноко и что по достижении ширины области просмотра порядка 480 пикселов появляется достаточно места, чтобы снова подключить обтекание их текстом. Позаботимся об этом, смешая изображения влево, когда ширина экрана достигает 480 пикселов (рис. ЦВ-17.14):

```
@media screen and (min-width: 480px) {  
    main img {  
        float: left;  
        margin: 0 1em 1em 0;  
    }  
}
```

Текст и типографика

Как только экран приобретает ширину, равную, приблизительно, 600 пикселам, я чувствую, что появилось достаточно места, чтобы кое-что приукрасить. Так, в заголовке теперь есть место для слогана, поэтому я снова сделаю его видимым.

ОБТЕКАНИЕ МОЖНО ОРГАНИЗОВАТЬ И ИНТЕРЕСНЕЕ

При желании для организации более интересных вариантов обтекания текста можно подключить формы CSS, рассмотренные в главе 15. Я здесь опустила подобное экспериментирование для краткости и из-за ограниченной поддержки браузера.

Теперь уделим немного внимания типографике. Мне нравится веб-шрифт *Stint Ultra Expanded*, хотя он и не является ключевым для нашего бренда, однако я не применяла его в узком макете из-за проблем с длиной строки. Но с этой точки прерывания можно начать его использование, поскольку он более разборчив и обеспечивает удобную длину строки. Я также немного уменьшила высоту строки. Я также воспользуюсь дополнительным пространством, чтобы добавить закругленную границу вокруг основной текстовой области, чтобы приблизить ее к общему виду нашего сайта. Результатом является улучшенная компоновка в одну колонку, которая хорошо подходит для устройств размером с планшет (рис. ЦВ-17.15).

Далее я привожу медиазапрос для точки прерывания на ширине 600 пикселов. Добавьте его в конец таблицы стилей после двух других запросов:

```
@media screen and (min-width: 600px) {  
    header p {  
        display: block;  
        margin-top: -1.5em;  
        font-family: Georgia, serif;  
        font-style: italic;  
        font-size: 1.2em;  
    }  
    main, h2, h3 {  
        font-family: 'Stint Ultra Expanded', Georgia, serif;  
    }  
    h2, h3 {  
        font-weight: bold;  
    }  
    main {  
        line-height: 1.8em;  
        padding: 1em;  
        border: double 4px #EADDCC;  
        border-radius: 25px;  
        margin: 2.5%;  
    }  
}
```

Многоколоночный макет

Продолжая расширять окно просмотра и поглядывая на подсвеченный желтым диапазон символов, я замечаю, что длина строки текста превышает 75 символов. Можно было бы увеличить размер шрифта или поля, но я полагаю, что это хороший момент для добавления в макет второй колонки. Впрочем, назначение именно здесь точки прерывания — мой сугубо субъективный подход, и он пригоден, только если вы не разрабатываете макет под конкретное устройство. Но, как бы там ни было, в качестве точки, на которой страница получает многоколоночный макет, я выбрала ширину 940 пикселов.

Чтобы создать эту точку прерывания, я просто взяла стили макета сетки из предыдущей главы и применила их здесь. Для элемента *aside* я переместила волнистый фоновый рисунок к левому краю. Кроме того, я задала максимальную ширину контейнера 1200 пикселов и установила боковые поля в *auto*, так что если окно браузера станет шире, чем 1200 пикселов, макет останется при фиксированной

ширине и будет центрирован в области просмотра. Наконец, я разместила изображение награды вверху страницы, поскольку места там достаточно (рис. ЦВ-17.16).

Добавьте этот последний медиазапрос в конец таблицы стилей. Можно скопировать их из заключительного упражнения главы 16 и вставить сюда (именно это я и сделала) и внести некоторые изменения в правила `#container` и `#aside`, как показано здесь:

```
@media screen and (min-width: 940px) {  
    #container {  
        display: grid;  
        grid-template-rows: auto min-height 5em;  
        grid-template-columns: minmax(25em, 1fr) 16em;  
        grid-template-areas:  
            "banner banner"  
            "main hours"  
            "footer footer";  
        max-width: 1200px;  
        margin: 0 auto;  
        position: relative;  
    }  
    header {  
        grid-area: banner;  
    }  
    main {  
        grid-area: main;  
    }  
    aside {  
        grid-area: hours;  
        background: url(images/scallop.png) repeat-y left top;  
        background-color: #F6F3ED;  
        padding: 1em;  
        padding-left: 45px;  
    }  
    footer {  
        grid-area: footer;  
    }  
    #award {  
        position: absolute;  
        top: 30px;  
        left: 50px;  
    }  
}
```

ОТКЛЮЧИТЕ ПОДСВЕЧЕННЫЙ ФОН

Подсвеченный желтым фон, демонстрирующий длину строки, перед публикацией сайта должен быть отключен, но я оставила его видимым на приведенных здесь иллюстрациях, чтобы вам было виднее, как меняется длина строки в зависимости от настройки макета.

И вот мы сделали это! Это самый сложный адаптивный сайт? Нет. Можно ли сделать что-либо еще, чтобы улучшить дизайн для экранов разных размеров? Конечно! Но сейчас вы должны почувствовать, что это такое: начать с дизайна для маленьких экранов и постепенно вносить изменения, которые все больше и больше оптимизируют макет под экраны больших размеров. Считайте это скромным первым шагом к будущим приключениям в АВД.

НЕСКОЛЬКО СЛОВ О ТЕСТИРОВАНИИ

В только что выполненном упражнении для принятия решений об изменениях стиля под экраны различных размеров мы воспользовались инструментом Responsive View современного браузера, однако, хотя это и удобный инструмент для создания первоначального дизайна, требуется гораздо более основательное тестирование сайта, прежде чем его дизайн можно будет считать готовым для финального запуска. Это особенно важно для сайтов, которые включают функции, основанные на JavaScript, или функциональность на стороне сервера.

Существуют три основных варианта для тестирования сайтов: реальные устройства, эмуляторы и сторонние сервисы. И в этом разделе мы рассмотрим каждый из этих вариантов.

Реальные устройства

Как бы там ни было, но ничто не заменит тестирования сайта на множестве реальных устройств и операционных систем. Помимо возможности просмотра внешнего вида сайта, тестирование на реальных устройствах показывает, как *на самом деле работает* ваш сайт. Как быстро загружается? Легко ли переходить по ссылкам? Хорошо ли работают все интерактивные функции? А, вообще, они работают?

Занимающиеся веб-разработкой компании могут иметь *лаборатории устройств*, оснащенные iPhone и iPad с экранами различных размеров, смартфонами и планшетами на Android с экранами различных размеров, а также компьютерами Mac и PC с новейшими операционными системами (Windows и Linux), которые используются дизайнерами и разработчиками при тестировании сайтов (рис. 17.17). Разумеется, размер лаборатории устройств зависит от величины бюджета (электронные устройства недешевы!).

СОЗДАНИЕ ЛАБОРАТОРИИ УСТРОЙСТВ

Если вы захотите создать собственную лабораторию устройств, я рекомендую прочитать учебник по теме «Создание лаборатории устройств» Дестини Монтегю и Лары Хоган («Пять простых шагов к публикации»). Книга представляет собой краткое изложение всего, что авторы узнали при создании лаборатории «убийц устройств» для Etsy. Книга доступна на бесплатной основе по адресу: buildingadevicelab.com.



Рис. 17.17. Лаборатория устройств Филамент Групп в Бостоне, штат Массачусетс

Если вам не посчастливилось работать в крупной компании, имеющей большую лабораторию устройств, у вас есть альтернативы:

- если вы живете в большом городе, недалеко от вас может располагаться лаборатория устройств, открытая для публичного использования. Посетите сайт: opendevicelab.com, чтобы узнать, есть ли у вас такой шанс;
- вы можете создать собственную лабораторию с коллекцией использованных устройств. Как минимум, у вас должен быть доступ к iPhone, смартфону Android, iPad, 7-дюймовому планшету (например, iPad Mini) и компьютерам под управлением macOS и Microsoft Windows. При этом вам не потребуется приобретать тарифный план на каждое устройство, поскольку вы сможете проводить тестирование через Wi-Fi;
- если покупка устройств вам не по карману, вы можете попросить друзей и коллег ненадолго одолжить вам свои телефоны и планшеты. Обращение в розничный магазин мобильных устройств для тестовой загрузки веб-страниц на их устройства также не исключается и довольно-таки распространено.

Если у вас есть несколько реальных устройств для тестирования своих веб-страниц, использование инструментов синхронизации делает этот процесс намного более приятным. Такие приложения, как BrowserSync (browsersync.io) и Ghostlab (www.vanamco.com/ghostlab/), запущенные на вашем компьютере, передают все, что находится на его экране, на все ваши устройства одновременно, поэтому вам не требуется загружать страницу по отдельности на каждом из них. Это подобно волшебству!

Эмуляторы

Если какое-либо конкретное устройство вам недоступно, вы можете использовать эмулятор — компьютерное приложение, эмулирующее оборудование мобильных устройств и их операционные системы. Эмулятор предоставляет окно, которое показывает, как именно сайт будет вести себя на этом конкретном устройстве (рис. 17.18).



Android Emulator (эмulateur Android) позволяет настроить для тестирования множество телефонов, телевизоров, носимых устройств и планшетов. Здесь я выбрали Nexus 5X.

Эмулятор Nexus 5X предоставляет изображение устройства в реальном размере. Все кнопки работают как на физическом телефоне.

Страница нашей пекарни просматривается на эмуляторе Nexus 5X.

Рис. 17.18. Примеры применения эмулятора Android Emulator (загружено с сайта по адресу developer.android.com/studio/index.html)

Эмуляторы сильно загружают память на вашем компьютере, и они могут содержать ошибки, но это, безусловно, лучше, чем вообще не тестировать устройство.

В качестве начального пособия по освоению эмуляторов могу порекомендовать книгу «Mobile Emulators & Simulators: The Ultimate Guide» Максимилиано Фиртмана (Maximiliano Firtman) (www.mobilexweb.com/emulators).

Услуги сторонних сервисов

Еще один вариант для тестирования сайта на более чем 1000 устройств — подписаться на сервис, такой, как BrowserStack (browserstack.com) или CrossBrowserTesting (crossbrowsertesting.com). За ежемесячную плату вы сможете получить доступ к огромному количеству симуляторов устройств (рис. 17.19). Существует множество подобных сервисов, и некоторые из них либо бесплатны, либо имеют бесплатные пробные версии. Они не дадут вам того же понимания ситуации, что и тестирование на реальных устройствах, но это все же еще одна альтернатива не проводить тестирование вовсе.



BrowserStack.com



CrossBrowserTesting.com

Рис. 17.19. Снимки экрана, созданные BrowserStack и CrossBrowserTesting (я использовала бесплатные пробные версии этих инструментов). Обратите внимание на варианты отображения страницы нашей пекарни. Вот поэтому необходимо тестирование!

ДОПОЛНИТЕЛЬНЫЕ СПРАВОЧНЫЕ РЕСУРСЫ ПО АДАПТИВНОМУ ВЕБ-ДИЗАЙНУ

Мы рассмотрели механизмы использования гибких макетов, гибких изображений и медиазапросов для создания страниц, пригодных для использования в широком диапазоне размеров экрана. А также познакомились со стратегиями адаптивной разработки и некоторыми общими адаптивными шаблонами для макета, навигации, типографики и изображений. На этом основании у вас может возникнуть уверенность в возможности самостоятельно создать адаптивную страницу. Однако на самом деле полученные вами знания — это только лишь верхушка айсберга, и я призываю вас продолжить изучение АВД, особенно если рассматривать веб-дизайн как ступень карьеры. Далее я привожу список ресурсов АВД, которые нахожу полезными и способными указать вам правильное направление.

Книги

- «*Responsive Web Design*, 2-е издание», Итан Маркотт (Ethan Marcotte) — эта книга обязательна к прочтению. Итан предоставляет более подробные сведения о том, как рассчитать гибкие сетки и как использовать медиазапросы, чем эта книга. Кроме того, ее просто весело читать;
- «*Learning Responsive Web Design: A Beginner's Guide*» от Клариссы Петерсон (Clarissa Peterson) (издательство O'Reilly) — Кларисса предоставляет исчерпывающий обзор всех аспектов адаптивного дизайна: от подробных примеров кода до широких стратегий рабочего процесса и дизайна, ориентированного на мобильные устройства;
- «*Smashing Book #5: Real-Life Responsive Web Design*», разные авторы (Smashing Magazine) — коллекция практических приемов и стратегий от известных веб-дизайнеров;
- «*Atomic Design*», Брэд Фрост (Brad Frost) — Брэд описывает свой модульный подход к АВД, ставший весьма популярным для разработки крупных сайтов;
- «*Responsive Design Workflow*», Стивен Хей (Stephen Hay) (издательство New Riders) — Стивен Хей представляет для создания адаптивных сайтов свой метод «дизайн в браузере». Его книга наполнена предложениями о том, как подходить к веб-дизайну и разработке;
- «*Implementing Responsive Design*», Тим Кадлек (Tim Kadlec) (издательство New Riders) — Тим Кадлек является лидером сообщества мобильных веб-дизайнеров, а его книга представляет собой обширное руководство по проектированию и созданию адаптивных сайтов.

НЕ ЗАБЫВАЙТЕ, ЧТО CSS GRID LAYOUT УЖЕ СУЩЕСТВУЕТ

Большинство упомянутых здесь изданий вышли еще до того, как CSS Grid Layout вошел в свои права. Так что имейте в виду, что у вас уже есть расширенные инструменты для гибких макетов, не упомянутые в этих книгах.

Ресурсы в Интернете

- «*A Responsive Web Design Is...*» (responsivedesign.is) — сборник статей и подкастов о веб-дизайне. Вы также можете подписаться на новостную рассылку «RWD Weekly» и держать руку на пульсе АВД. Сайт является проектом Джастина Эйвери и Simple Things (Justin Avery and Simple Things);
- «*Responsive Resources*» (bradfrost.github.io/this-is-responsive/resources.html) — если у вас есть вопросы об АВД, вы не найдете ответов лучше, чем у Брэда Фроста (Brad Frost's Responsive Resources). Он собрал сотни ссылок на ресурсы, связанные со стратегией, инструментами дизайна, макетом, медиазапросами, типографикой, изображениями, компонентами, разработкой, тестированием, системами управления контентом, электронной почтой, учебными пособиями и многим другим. Серьезно, здесь достаточно материала, чтобы занять вас на месяцы;
- «*Media Queries*» (mediaqueri.es) — созданная Эйвингнд Угедал (Eivind Uggdal) галерея исключительных примеров адаптивных сайтов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Поскольку наступил конец очередной главы, вам уже понятно, что это значит... пришло время викторины! В случае появления затруднений обратитесь к *приложению 1*, где приведены ответы на эти вопросы:

1. В чем отличие адаптивного сайта от мобильного (м-дот) сайта?

2. Что выполняет следующий код?

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

3. Как убедиться, что изображение уменьшается, если его контейнер уменьшается в макете?

4. Что выполняет следующий код?

```
@media screen and (min-width: 60em) {  
    body {  
        margin: 0 10%;  
    }  
}
```

5. Каковы стратегии для создания макета, который приспосабливается к доступной ширине области просмотра?

6. В чем преимущество использования em в качестве единиц измерения в медиазапросах?

7. Приведите три способа использования медиазапроса.

8. Назовите три настройки, которые можно внести в типографику для ее успешной работы на маленьких экранах.

9. Каким образом можно управлять навигацией с большим количеством подменю на маленьком экране?

10. Приведите три варианта тестирования сайтов на нескольких устройствах.

ГЛАВА 18

ПЕРЕХОДЫ, ТРАНСФОРМАЦИИ И АНИМАЦИЯ

В этой главе...

- ▶ *Создание плавных переходов*
- ▶ *Перемещение, вращение и изменение размеров элементов*
- ▶ *Комбинирование переходов и трансформаций*
- ▶ *Немного о трехмерных трансформациях*
- ▶ *Анимация по ключевым кадрам*

К этому моменту мы уже рассмотрели возможности CSS, используемые для формирования визуальных эффектов, — таких, как закругленные углы, цветовые градиенты и тени, которые ранее приходилось создавать с помощью графики. В этой главе мы познакомимся с некоторыми свойствами CSS3, применяемыми для создания анимированных интерактивных эффектов, реализация которых ранее была возможна только с помощью JavaScript или Flash.

И начнем мы с модуля CSS Transitions (CSS-переходы), предоставляющего изящный способ плавного перехода от одного стиля к другому. Затем мы рассмотрим возможности модуля CSS Transforms (CSS-трансформации), обеспечивающего перемещение, масштабирование, поворот и наклон элементов, и разберемся, как можно анимировать их с помощью переходов. А завершить главу я собираюсь кратким введением в 3D-преобразования и CSS-анимацию, знать о которых важно, но поскольку эта тема слишком обширна, чтобы охватить ее здесь целиком, мы ее только коснемся.

Проблема этой главы в том, что анимация и изменяющиеся во времени визуальные эффекты не работают на бумаге, поэтому в книге продемонстрировать их невозможно. И максимум того, что я могу сделать, чтобы вам все это стало понятнее, — это выложить исходный код для рисунков в папку **figures** материалов для этой главы, доступных по ссылке: learningwebdesign.com/5e/materials. Следует лишь открыть соответствующий файл в браузере.

CSS TRANSITIONS

Представьте себе, например, ссылку в навигационном меню, цвет фона которой меняется с синего на красный, когда на нее наводится указатель мыши. Изначально фон окрашен в синий цвет... мышь проходит над ним... и раз! — фон становится красным. Он мгновенно переходит из одного состояния в другое без каких-либо промежуточных состояний. А теперь представьте, что когда указатель мыши наводится на ссылку, ее фон постепенно меняет цвет с синего на красный, проходя через несколько оттенков фиолетового. Это очень круто! А когда вы отводите мышь, ссылка снова постепенно становится синей.

Как раз *вот это* и делает модуль CSS Transitions. Он на некотором временнóм отрезке сглаживает резкие изменения значений свойств между двумя состояниями, создавая кадры промежуточных состояний. Этот процесс аниматоры называют твинингом. Используя модуль CSS Transitions, можно добавить утонченность и изысканность вашим интерфейсам и даже сделать их более удобными в эксплуатации.

Модуль CSS Transitions изначально был разработан командой WebKit для браузера Safari и является рабочим проектом Консорциума W3C (Working Draft at the W3C).

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О МОДУЛЕ CSS TRANSITIONS

Дополнительные сведения о модуле CSS Transitions можно найти на сайте по адресу: www.w3.org/TR/css-transitions-1/.

браузерах, не поддерживающих CSS Transitions (привет тебе, IE), то как выглядит в них переход от синего к красному цвету, не имеет особого значения.

Браузеры прекрасно поддерживают переходы (см. врезку «Поддержка CSS Transitions»), поэтому нет причин исключать этот эффект из проектов, особенно если рассматривать его как расширение. Ну а если говорить о некоторых устаревших

Основы организации переходов

Переходы — это очень интересно, так что давайте приступим к их рассмотрению. В процессе CSS-верстки по части переходов доступны несколько решений, каждое из которых настраивается с помощью CSS-свойств:

- какое CSS-свойство перехода изменяется (`transition-property`) — требуется;
 - как долго длится эффект перехода (`transition-duration`) — требуется;
 - в каком направлении ускоряется переход (`transition-timing-function`);
 - будет ли пауза перед началом перехода (`transition-delay`).

ИСПОЛЬЗУЙТЕ СВОЙСТВА ПЕРЕХОДА

Добавьте свойства перехода к объекту, который будет подвергаться переходу.

Переходы требуют указания начального и конечного состояний. Элемент, который появляется при первой загрузке, является начальным состояни-

ем. Конечное состояние вызывается изменением состояния — таким, как `:hover`, `:focus` или `:active`, которое мы будем использовать для примеров в этой главе. Вы также можете использовать JavaScript для изменения элемента (например, добавления атрибута `класс`) и использовать его как триггер перехода.

ПОДДЕРЖКА CSS TRANSITIONS

Мне приятно сообщить вам, что все современные браузеры, выпущенные после 2013 года, поддерживают свойства CSS Transitions без использования префиксов. Однако реализация этой поддержки связана с рядом проблем, о которых вам следует знать:

- в частности, браузер Internet Explorer версии 9 и более ранних не поддерживает переходы, полностью игнорируя их свойства;
- версии браузеров Chrome и Safari, выпущенные в период с 2010-го по 2013-й год, поддерживают переходы с помощью префикса `-webkit-`. Более современные версии этих браузеров не нуждаются в префиксах;
- для мобильных устройств версий iOS 3.1–6.0 (2010–2013) и Android-версий 2.1–4.3 (2009–2013) требуется префикс `-webkit-`. Более современные версии этих браузеров не нуждаются в префиксах;
- версии браузера Firefox, выпущенные между 2011-м и 2012-м годами, нуждаются в префиксе `-moz`, но эти версии сейчас уже почти не используются.

Как всегда в таких случаях, проверьте статистику собственного сервера (не забудьте обратить внимание на поддержку мобильных устройств), чтобы увидеть, какие браузеры нуждаются в поддержке, и проверьте сайт CanIUse.com на предмет поддержки браузеров и подробных описаний ошибок.

В примерах этой главы задействованы только стандартные свойства CSS Transitions (без префикса). Если необходимо организовать поддержку браузеров, требующих префиксы, я рекомендую воспользоваться утилитой Autoprefixer, которая будет рассмотрена в главе 19. И помните: при задействовании префиксных свойств всегда выбирайте последнюю версию браузера без префикса для прямой совместимости с поддерживающими браузерами.

Проиллюстрируем сказанное на простом примере, представляющем собой разметку для представленного ранее сине-красного перехода (рис. ЦВ-18.1). В этой разметке нет ничего особенного — был просто добавлен класс `class`, что позволяет уточнить, какие ссылки получают переходы.

Свойства перехода применяются здесь к объекту, который будет участвовать в переходе, — в нашем случае это элемент `a`, который находится в своем нормальном состоянии. Эти свойства можно увидеть во множестве других объявлений для `.smooth` — таких, как `padding` и `background-color`. Цвет фона для ссылки изменяется на красный с помощью объявления `background-color` для состояния `:hover` (а также для `:focus` в случае, если кто-либо наводит курсор на ссылку с помощью клавиатуры):

Разметка

```
<a href="..." class="smooth">awesomesauce</a>
```

Правило стиля

```
.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;
}
.smooth:hover, .smooth:focus {
background-color: red;
}

```

Спецификация свойства

`transition-property`

- **Значения:** имя свойства | all | none.
- **По умолчанию** — all.
- **Применение** — ко всем элементам, :before и :after псевдоэлементам.
- **Наследование** — нет.

Свойство `transition-property` идентифицирует CSS-свойство, которое изменяется и к которому нужно плавно перейти. В нашем примере это `background-color`. Также можно изменить цвет переднего плана, границы, размеры, шрифты, текстовые атрибуты и многое другое. В табл. 18.1 приведены анимируемые CSS-свойства, известные на момент подготовки этой книги. Общее правило заключается в том, что если значение является цветом, длиной или числом, то такое свойство может быть свойством перехода.

Таблица 18.1. Анимационные CSS-свойства

Фоны (Backgrounds)

`background-color`
`background-position`

Границы и контуры (Borders and outlines)

`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-width`

Цвет и непрозрачность (Color and opacity)

`color`
`opacity`
`visibility`

Шрифт и текст (Font and text)

`font-size`
`font-weight`
`letter-spacing`
`line-height`
`text-indent`
`text-shadow`
`word-spacing`
`vertical-align`

Табл. 18.1. (окончание)

Размеры блока элемента (Element box measurements)	Позиция (Position)
height	top
width	bottom
max-height	left
max-width	z-index
min-height	clip-path
min-width	
margin-bottom	
margin-left	
margin-top	
padding-bottom	
padding-left	
padding-right	
padding-top	

Преобразования (Transforms)
отсутствуют в спецификации на момент подготовки книги, но поддерживаются

transform
transform-origin

Сколько времени это занимает?

`transition-duration`

- **Значения:** время.
- **По умолчанию** — 0 сек.
- **Применение** — ко всем элементам, `:before` и `:after` псевдоэлементам.
- **Наследование** — нет.

Свойство `transition-duration` устанавливает интервал времени, необходимый для завершения анимации в секундах (s) или миллисекундах (ms). Я выбрала значение, равное 0,3 секунды, чего достаточно, чтобы заметить происходящее, но при этом не столь много, чтобы переход воспринимался медленным или замедлял действия пользователя. Конечно, точной статистики, отражающей корректное время перехода, нет, но я нашла, что значение порядка 0,2 секунды является популярным интервалом перехода, применяемым для элементов пользовательского интерфейса. Попробуйте, чтобы уточнить интервал времени, имеющий смысл для вашего приложения.

Функции работы со временем

`transition-timing-function`

- **Значения:** `ease` | `linear` | `ease-in` | `ease-out` | `ease-in-out` | `step-start` | `step-end` | `steps` | `cubic-bezier(#,#,#,#)`.
- **По умолчанию** — `ease`.
- **Применение** — ко всем элементам, `:before` и `:after` псевдоэлементам.
- **Наследование** — нет.

Свойство продолжительности необходимо и составляет основу перехода, но вы можете его дополнительно уточнить. Есть несколько способов, которыми переход

может разворачиваться во времени. Например, он может начинаться быстро, а затем замедляться, начинаться медленно и ускоряться или оставаться на одной и той же скорости на всем его протяжении. Я представляю себе эти способы как «стили» перехода, но в спецификации они известны как *функция времени* (timing function) или *функция замедления* (easing function).

Выбранная функция времени может оказывать большое влияние на ощущение от анимации и ее правдоподобность, поэтому, если вы планируете применять переходы и CSS-анимацию, рекомендуется ознакомиться с ее параметрами.

Если я присвою свойству `transition-timing-function` значение `ease-in-out`, переход начнется медленно, затем ускорится, затем, на пути к конечному состоянию, снова замедлится:

```
.smooth {  
    ...  
    transition-property: background-color;  
    transition-duration: 0.3s;  
    transition-timing-function: ease-in-out;  
}
```

Свойство `transition-timing-function` принимает одно из следующих значений (ключевых слов):

- `ease` — переход начинается медленно, быстро ускоряется, затем в конце замедляется. Это значение задано по умолчанию и отлично работает для большинства коротких переходов;
- `linear` — скорость перехода остается неизменной от начала и до конца. У некоторых пользователей при этом создается ощущение излишней механистичности;
- `ease-in` — переход начинается медленно, затем ускоряется;
- `ease-out` — переход начинается быстро, затем замедляется;
- `ease-in-out` — переход начинается медленно, затем постепенно ускоряется и снова замедляется в самом конце. Это похоже на эффект использования `ease`, но с менее выраженным ускорением в середине;
- `cubic-bezier(x1, y1, x2, y2)` — ускорение перехода можно сформировать с помощью кривой, называемой кривой Безье. Крутые части кривой указывают на высокую скорость изменения, а ее плоские части — на медленную. На рис. ЦВ-18.2 показаны кривые Безье, которые представляют ключевые слова функций, а также созданную мною пользовательскую кривую. Можно заметить, что кривая `ease` вначале имеет плоский вид, затем становится крутой (изменяется быстро), затем заканчивается плоским участком (изменяется медленно). С другой стороны, ключевое слово `linear` задает перемещение с постоянной скоростью в течение всего перехода.

Вы можете получить представление о своей анимации, *просто* формируя необходимую кривую. Сайт **Cubic-Bezier.com** служит отличным инструментом для изучения времени перехода и генерирования результирующего кода. Четыре числа в значении представляют положения `x` и `y` начальных и конечных маркеров кривой Безье (розовые и синие точки, показанные на рис. ЦВ-18.2);

- `steps(#, start | end)` — разбивает переходы на несколько шагов, как определено этой пошаговой функцией. Первое значение задает количество шагов, ключевые слова `start` и `end` определяют, изменяется ли состояние в начале (`start`) или в конце каждого шага. Пошаговая анимация полезна для анимации ключевых кадров с изображениями движущихся фигур. За подробными пояснениями и примерами я рекомендую обратиться к статье «Using Multi-Step Animations and Transitions» Джекфа Грэма (Geoff Graham), посвященной описанию приемов, связанных с CSS (css-tricks.com/using-multistep-animations-transitions/);
- `step-start` — изменяет состояние за один шаг в начале интервала длительности (аналогично результату применения значения: `steps(1, start)`). Результатом является внезапное изменение состояния, как если бы переход не применялся вовсе;
- `step-end` — изменяет состояние за один шаг в конце интервала длительности (аналогично результату применения значения: `steps(1, end)`).

Нелегко отобразить различные варианты анимации на неподвижной странице книги, но я скомпоновала для вас небольшую демонстрацию, которая показана на рис. 18.3 и доступна в папке **figures** материалов для этой главы. Ширина каждого помеченного элемента (прямоугольник белого цвета с синей границей) изменяется

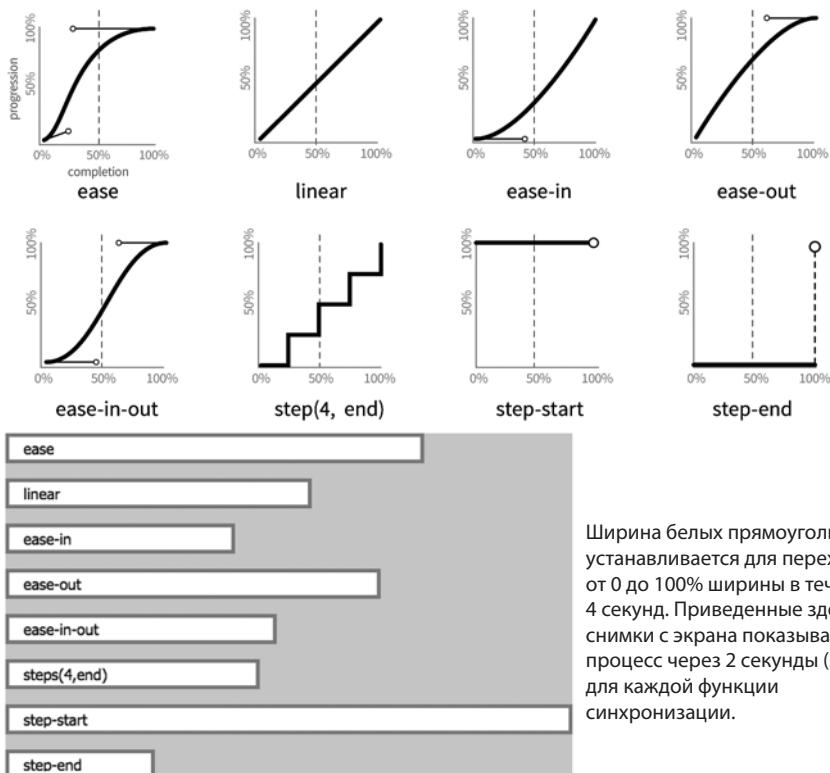


Рис. 18.3. В этой демонстрации свойства `transition-timing-function` элементы достигают полной ширины в одно и то же время, но различаются в зависимости от того, как они это делают. Если вы хотите увидеть реализацию, обратитесь к файлу `ch18_figures.html`, доступному в материалах этой главы

РАЗДЕЛЕНИЕ ФУНКЦИЙ ВРЕМЕНИ

W3C разнес функции времени на отдельные спецификации, чтобы их было легче распределить между модулями. Информация об этом доступна по ссылке: www.w3.org/TR/css-timing-1/.

в течение 4 секунд при наведении указателя мыши на зеленое поле. Они все достигают полной ширины в одно и то же время, но достигают по-разному. Показанные на рис. 18.3 изображения получены на 2-секундной отметке времени перехода, то есть на полпути.

Установка задержки

`transition-delay`

- **Значения:** *время*.
- **По умолчанию** — 0 сек.
- **Применение** — ко всем элементам, `:before` и `:after` псевдоэлементам.
- **Наследование** — нет.

Свойство `transition-delay`, как можно догадаться, задерживает запуск анимации на указанное количество времени. В следующем примере переход цвета фона начинается через 2 секунды после перемещения указателя на ссылку:

```
.smooth {
...
    transition-property: background-color;
    transition-duration: 0.3s;
    transition-timing-function: ease-in-out;
    transition-delay: 0.2s;
}
```

Сокращенное свойство `transition`

К счастью, авторы CSS3-спецификации предусмотрели сокращенное свойство `transition`, позволяющее скомбинировать все свойства перехода в одно объявление. Мы уже выполняли подобные преобразования с помощью сокращенного свойства `border`. Далее приводится синтаксис сокращенного свойства `transition`:

```
transition: property (свойство) duration (продолжительность) timing-function (функция времени) delay (задержка);
```

Значения для каждого из свойств `transition-*` задаются в списке через пробелы. Порядок их следования неважен, если значение `duration` (требуемое) отображается перед значением `delay` (необязательным). Если указать только одно значение времени, оно и будет считаться значением продолжительности.

Возвращаясь к примеру сине-красной ссылки, объединим в одной строке четыре свойства перехода:

```
.smooth {
...
    transition: background-color 0.3s ease-in-out 0.2s;
}
```

Не стоит и говорить, что так гораздо удобнее.

Применение нескольких переходов

До сих пор за один прием изменялось только одно свойство, но возможно выполнить переход, состоящий сразу из нескольких свойств. Вернемся к примеру со ссылкой «awesomesauce» (см. рис. ЦВ-18.1). Но на этот раз, в дополнение к изменению цвета фона ссылки с синего на красный, я немного увеличу расстояние между буквами, добавив в код значение `letter-spacing`. Я также хочу, чтобы цвет текста ссылки изменился с белого на черный, но медленнее, чем это реализовано для других анимаций (рис. ЦВ-18.4).

Один из способов реализовать это — указать через запятую все значения для каждого свойства, как показано в следующем примере:

```
.smooth {  
  ...  
  transition-property: background-color, color, letter-spacing;  
  transition-duration: 0.3s, 2s, 0.3s;  
  transition-timing-function: ease-out, ease-in, ease-out;  
}  
.smooth:hover, .smooth:focus {  
  background-color: red;  
  letter-spacing: 3px;  
  color: black;  
}
```

Значения сопоставляются соответственно их позициям в списке. Например, переход к свойству `color` (второму в списке) имеет длительность `2s` и использует функцию времени `ease-in`. Если один список имеет меньше значений, чем другие, браузер повторяет значения этого списка, начав с первого. В приведенном примере, если для `transition-duration` пропустить третье значение (`0.3s`), браузер вернется к началу списка и задействует первое значение (`0.3s`), относящееся к `letter-spacing`. В нашем случае эффект будет тот же.

Вы также можете выстроить значения в ряд, воспользовавшись сокращенным свойством `transition`. Тот же набор стилей, который мы только что привели, можно записать и так:

```
.smooth {  
  ...  
  transition: background-color 0.3s ease-out,  
             color 2s ease-in,  
             letter-spacing 0.3s ease-out;  
}
```

Переход для всех случаев

Но что если вы просто хотите добавить немного плавности ко всем изменениям состояния, независимо от того, какое свойство должно измениться? В случаях, когда надо, чтобы одинаковые длительность (`duration`), функция времени

(`timing-function`) и задержка (`delay`) применялись ко всем переходам, которые могут происходить в элементе, используйте значение `all` для `transition-property`.

В следующем примере я указала, что любое свойство, изменяющееся для элемента `.smooth`, должно длиться 0,2 секунды, а его анимация реализуется с помощью функции `ease-inout`:

```
.smooth {
...
  transition: all 0.2s ease-in-out;
}
```

Завершим урок по CSS3-переходам, выполнив *упражнение 18.1*.

ЗНАЧЕНИЕ `ALL`

Для организации изменений в пользовательском интерфейсе часто необходим небольшой краткий переход, объединяющий все ваши переходы, и для этого вам всегда пригодится значение `all`.

УПРАЖНЕНИЕ 18.1. ПРИМЕНЕНИЕ ПЕРЕХОДОВ

При выполнении этого упражнения мы создадим следующие анимированные переходы (рис. ЦВ-18.5):

- при наведении указателя мыши на ссылку (кнопку меню) изменяется цвет ее фона и границ;
- при щелчке мышью на кнопке меню изменяется вид ее границ (она становится как бы «утопленной» в панель).

Сначала рассмотрим уже примененные к документу стили. Список преобразован в горизонтальное меню с помощью Flexbox. Элемент `a` настроен на отображение в виде элемента блока, подчеркивания отключены, заданы размеры и отступы, установлены цвет, цвет фона и границы. Я применила свойство `box-shadow`, в результате чего ссылки выглядят так, как будто они всплывают над страницей.

1. Определим стили для состояний наведения и фокуса. Когда пользователь наводит указатель мыши на ссылку (или вкладку), выбирается зеленый цвет фона (`#c6de89`), а цвет границы изменяется на более темный оттенок зеленого (`#a3c058`):

```
a:hover, a:focus {
  background-color: #c6de89;
  border-color: #a3c058;
}
```

ВНИМАНИЕ!

Если упражнение выполняется для сенсорного устройства, вы упустите демонстрируемые в нем эффекты, поскольку на сенсорных экранах отсутствует отдельное от щелчка состояние наведения. Нечто похожее на эффект наведения вы увидите, однократно нажав на кнопку экранного меню. Переходы, инициируемые щелчком/нажатием, или тот случай, когда загрузка страницы выполняется на всех устройствах, здесь не рассматриваются.

Я выложила исходный документ (файл `exercise_18-1.html`) в папку `materials` для этой главы, доступную по адресу: learningwebdesign.com/5e/materials. Удостоверьтесь, что на вашем настольном компьютере установлен браузер, способный отобразить выполнение этого упражнения.

- Когда пользователь щелкнет на ссылке (:active), это приведет к ее смещению на 3 пикселя, как будто выполняется нажатие. Выполните это, установив для элемента `a` значение `position` в `relative`, а его позицию `top` — в значение `0px`, после чего измените значение свойства `top` на состояние `active`. В результате ссылка сместится на 3 пикселя от верхнего края (другими словами, «заглубится»).

```
a {  
    ...  
    position: relative;  
    top: 0px;  
}  
a:active {  
    top: 3px;  
}
```

ОБХОДИМ ОШИБКУ!

Первоначальное присваивание значения `0px` свойству `top` предназначено для обхода ошибки, возникающей при переходе свойств `top`, `bottom`, `left` и `right`.

- Логично предположить, что если кнопку нажать, то тень от нее будет меньше, поэтому мы также уменьшим расстояние `box-shadow`:

```
a:active {  
    top: 3px;  
    box-shadow: 0 1px 2px rgba(0,0,0,.5);  
}
```

- Сохраните файл и просмотрите документ в браузере. Ссылки должны при наведении на них курсора становиться зелеными и «заглубляться», если на них выполняется щелчок или касание. Что ж, все это очень хорошо. Но мы можем улучшить производимое впечатление, добавив несколько плавных переходов.
- Введите переходы цвета фона и границы, выполняемые за более чем 0,2 секунды, и посмотрите, как это отразится на нашем меню. Я использую сокращенное свойство `transition`, чтобы упростить код. Здесь также задействуется функция времени `ease`, но поскольку она задается во умолчанию, ее значение мы можем опустить.

Я не использую здесь префиксы поставщиков, поскольку современные браузеры в них не нуждаются. Если же вы собираетесь поддерживать мобильные браузеры, выпущенные в 2013-м году и ранее, можно также включить версию кода с префиксом `-webkit-`, но, поскольку в нашем случае мы все-таки создаем не рабочий код, а учебный, пока обойдемся без этого:

```
a {  
    transition: background-color 0.2s,  
    border-color 0.2s;  
}
```

- Сохраните документ, откройте его в браузере и попробуйте навести курсор мыши на ссылки. Вы заметили, насколько удобнее стало работать? Теперь попробуйте другие значения для длительности (`duration`) перехода. Посмотрите, заметна ли разница при длительности, равной 0,1 с. Теперь введите значение длительности в одну секунду (`1s`). Полагаю, вы увидите, что 1 секунда — это слишком медленный переход. Попробуйте выбрать длительность перехода в несколько секунд и опробовать различные значения `timing-function` (просто добавьте эти значения после настройки длительности). Заметна ли разница? Что бы вы предпочли? Закончив экспериментировать, снова установите значение длительности, равной 0,2 секунды.

7. Посмотрим, что произойдет, если добавить переход к «заглублению» ссылки, когда на нее нажимают или касаются. Введите переход для свойств `top` и `box-shadow`, поскольку они должны изменяться в тандеме. Начнем со значения длительности, равного `0.2s`, а затем попробуем другие значения:

```
a {
    transition:
        background-color 0.2s,
        border-color 0.2s,
        top 0.2s,
        box-shadow 0.2s;
}
```

Сохраните файл, откройте его в браузере и попробуйте щелкать на ссылках. Созданный нами переход действительно изменяет представление об использовании меню, не так ли? Теперь кнопки «труднее» нажимать. Попробуйте еще увеличить длительность. Становится еще «труднее»? Весьма любопытно увидеть влияние времени задержки на пользовательский интерфейс. Важно понять это правильно и не придавать элементам интерфейса лишней «вязости». На мой взгляд, очень короткий переход — такой, как `0.1` секунды или вообще без перехода — сохранит ощущение четкого срабатывания кнопок.

8. Если вы решили, что увеличение длительности сделало меню менее удобным для работы, добавьте небольшую задержку, равную `0.5` секунды, к свойствам `top` и `box-shadow`:

```
a {
    transition:
        background-color 0.2s,
        border-color 0.2s,
        top 0.1s 0.5s,
        box-shadow 0.1s 0.5s;
}
```

И в самом деле, наличие лишней задержки при нажатии кнопок меню только мешает работать. Ведь соблюдение сроков — это так важно!

МОДУЛЬ CSS TRANSFORMS

`transform`

- **Значения:** `rotate()` | `rotateX()` | `rotateY()` | `rotateZ()` | `rotate3d()` | `translate()` | `translateX()` | `translateY()` | `scale()` | `scaleX()` | `scaleY()` | `skew()` | `skewX()` | `skewY()` | `none`.
- **По умолчанию** — `none`.
- **Применение** — к преобразуемым (трансформируемым) элементам (см. соответствующую врезку).
- **Наследование** — нет.

Модуль CSS3 Transforms (www.w3.org/TR/css-transforms-1) предоставляет разработчикам возможность вращать, перемещать, изменять размер и наклонять HTML-элементы как в двух-, так и в трехмерном пространстве. Стоит отметить, что

трансформация изменяет то, как отображается элемент, но не зависит от его движения или значения времени трансформации. Тем не менее вы можете анимировать переход из одного состояния трансформации в другое, используя анимации переходов или ключевых кадров, поэтому трансформацию полезно изучать в контексте анимации.

Эта глава посвящена более простым — двумерным — преобразованиям, поскольку они имеют широкое практическое применение. Трансформации поддерживаются практически всеми текущими версиями браузеров без префиксов поставщиков (соответствующие исключения приведены во врезке «Поддержка CSS-трансформаций»).

Трансформацию можно применить к обычному состоянию элемента — тогда элемент в преобразованном состоянии

ПРЕОБРАЗУЕМЫЕ (ТРАНСФОРМИРУЕМЫЕ) ЭЛЕМЕНТЫ

Свойство `transform` можно применять для большинства типов элементов:

- HTML-элементов с заменяемым контентом, таким, как `img`, `canvas`, поля ввода форм и вложенные медиа;
- элементов с отображаемыми на них элементами `block`, `inline-block`, `inline-table` (или любым из отображаемых типов `table-*`), `grid` и `flex`;

Может быть, легче выделить типы элементов, которые невозможно трансформировать, куда входят:

- незаменяемые встроенные элементы типа `em` или `span`;
- столбцы таблиц и группы столбцов (но кто же будет пытаться это делать?).

ПОДДЕРЖКА CSS-ТРАНСФОРМАЦИЙ

На момент подготовки книги CSS-трансформации поддерживаются всеми основными браузерами без применения префиксов поставщиков; тем не менее такая поддержка реализована уже после реализации поддержки переходов, что вызывает ряд потенциальных проблем. Далее приводится несколько относящихся к браузерам замечаний:

- браузер Internet Explorer 8 и его более ранние версии не поддерживают трансформации. Версия 9 этого браузера поддерживает трансформации с помощью суффикса `-ms-`;
- браузеры IE 10 и 11, а также все версии Edge поддерживают трансформации без префиксов, но не поддерживают трансформации для элементов в формате SVG;
- применяйте префикс `-webkit-`, если требуется поддержка следующих браузеров:
 - ◆ Android v2.1 до 4.4.4 (префиксы опущены в 2017-м году);
 - ◆ OS Safari v3.2 до 8.4 (префиксы опущены в 2015-м году);
 - ◆ Safari 8 и более ранние версии (префиксы опущены в 2015-м году);
 - ◆ Opera версии до v.22 (префиксы опущены в 2014-м году).

На момент подготовки книги рекомендовано включать для свойства `transform` префиксы `-ms-` и `-webkit-`, но ситуация может измениться к тому времени, когда вы читаете эту врезку. Просмотрите сайт CanIUse.com для получения обновленной информации о браузерах, а также сайт ShouldIPrefix.com для получения рекомендаций.

отображается уже при загрузке страницы. Просто убедитесь, что страница по-прежнему доступна в браузерах, которые не поддерживают трансформации. Обычно трансформация добавляется, если пользователи взаимодействуют с элементом с помощью состояния `:hover` или события JavaScript. В любом случае, трансформации служат для постепенного улучшения страницы — если пользователь IE8 видит элемент прямо, а не под углом, это не столь уж и важно.

На рис. 18.6 показано представление четырех функций двумерных трансформаций: `rotate()`, `translate()`, `scale()` и `skew()`. Пунктирная линия показывает исходную позицию элемента.

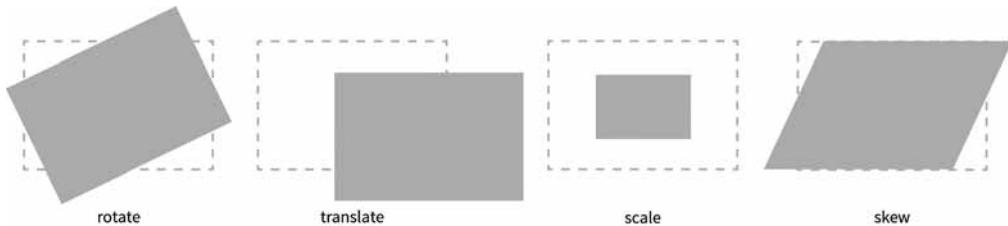


Рис. 18.6. Четыре типа трансформаций: `rotate()`, `translate()`, `scale()` и `skew()`

ВНИМАНИЕ: ПЯТЬ ДВУМЕРНЫХ ФУНКЦИЙ ПРЕОБРАЗОВАНИЯ

На самом деле в спецификации CSS определена также пятая функция: `matrix()`, которая позволяет формировать собственное комбинированное преобразование, используя шесть значений и некоторую особую тригонометрию. Существуют инструменты, которые могут принимать ряд трансформаций и объединять их в матричную функцию, но результат получается не слишком удобным для пользователя. Увлекательно в теории, но вряд ли применимо на практике.

Свойства `transform`, вы перемещаете только его изображение. Это изображение не влияет на окружающий макет. Давайте пройдемся по функциям преобразования по очереди, начиная с `rotate()`.

Функция `rotate()` (трансформация углов)

Если нужно, чтобы элемент изменил угол, под которым он расположен на странице, примените функцию трансформации `rotate()`. Значением этой функции служит угол в градусах, указанный при положительной величине (по часовой стрелке) или отрицательной величине (против часовой стрелки). В следующем примере изображение, показанное на рис. 18.7, повернуто на -10 градусов (350 градусов) с помощью соответствующего правила стиля (тонированное изображение показывает исходное положение элемента для справки):

Когда элемент трансформируется, его блок элементов сохраняет свое исходное положение и влияние на окружающий его макет, так же как и пространство, остающееся позади относительно позиционированного элемента. Это похоже на то, как будто трансформация волшебным образом собирает пиксели визуализируемого элемента, смешивается с ними и укладывает их обратно в верхнюю часть страницы. То есть, если вы перемещаете элемент с помощью

свойства `transform`, вы перемещаете только его изображение. Это изображение не влияет на окружающий макет. Давайте пройдемся по функциям преобразования по

очереди, начиная с `rotate()`.

```
img {
    width: 400px;
    height: 300px;
    transform: rotate(-10deg);
}
```

Обратите внимание, что изображение вращается вокруг центральной точки, которая является заданной по умолчанию точкой, вокруг которой выполняются все трансформации. Для изменения этого поведения следует применить свойство `transform-origin`:

`transform-origin`

- **Значения:** процентное соотношение | длина | left | center | right | top | bottom.
- **По умолчанию** — 50% 50%.
- **Применение** — к трансформируемым элементам.
- **Наследование** — нет.

Значением свойства `transform-origin` служат либо два ключевых слова из 5 возможных, либо единицы длины, либо процентные соотношения. Первое значение — горизонтальное смещение, второе — вертикальное. Если указано только одно значение, оно будет применено для обоих типов смещений. Синтаксис свойства `transform-origin` совпадает с приведенным ранее для свойства `background-position` (см. главу 13). Если нужно повернуть изображение вокруг точки в центре его верхнего края, можно написать код любым из следующих способов:

```
transform-origin: center top;
transform-origin: 50%, 0%;
transform-origin: 200px, 0;
```

Изображения, показанные на рис. 18.8, повернуты на 25 градусов, но из разных исходных точек. Исходную точку легко здесь продемонстрировать с помощью функции `rotate()`, но ее можно установить для любой из функций трансформации.



`transform: rotate(-10deg)`

Рис. 18.7. Вращение элемента `img` с помощью свойства `transform: rotate()`



`transform-origin:`
`center top;`



`transform-origin:`
`100% 100%;`



`transform-origin:`
`400px 0;`

Рис. 18.8. Изменение точки, вокруг которой вращается изображение, с помощью свойства `transform-origin`

Функции *translate()* (трансформация позиции)

Следующей возможностью для свойства `transform` является отображение элемента в новом положении на странице с помощью одной из трех функций `translate()`, как показано в примерах на рис. 18.9. Функция `translateX()` обеспечивает перемещение элемента вдоль горизонтальной оси, `translateY()` — предназначена для перемещения вдоль вертикальной оси; а `translate()` — комбинирует обе величины: X и Y:

```
transform: translateX(50px);
transform: translateY(25px);
transform: translate(50px, 25px); /* (translateX, translateY)
*/
```



`transform: translate(90px, 60px);` `transform: translate(-5%, -25%);`

Рис. 18.9. Перемещение элемента с помощью функции `translate()`

Значения длины можно указать в любых CSS-единицах или в процентах. Проценты рассчитываются по ширине ограничительного блока, то есть от одного края границы до другого (что, кстати, служит способом вычисления процентов в SVG, для которого были адаптированы трансформации). Вы можете также применять как положительные, так и отрицательные значения, что и показано на рис. 18.9.

Если поддерживается только одна величина для сокращенной функции `translate()`, она будет соотнесена со значением `translateX`, а значение `translateY` установится равным нулю. Поэтому функция `translate(20px)` будет эквивалентна совместному назначению `translateX(20px)` и `translateY(0)`.

Как вам понравилась такая ситуация со свойством `transform`? Да, и у нас имеются еще две функции.

Функции *scale()* (трансформация размера)

Увеличить или уменьшить элемент можно с помощью одной из трех функций масштабирования: `scaleX()` — вдоль горизонтальной оси, `scaleY()` — вдоль вертикальной оси и сокращенной функции `scale()`. Значением функций служит число без единиц измерения, которое определяет соотношение размеров. Следующий пример приводит изображение к значению 150% от его первоначальной ширины:

```
a img {
    transform: scaleX(1.5);
}
```

Сокращенная функция `scale()` объединяет значения для `scaleX` и для `scaleY` — в следующем примере элемент в два раза шире и в два раза выше оригинала:

```
a img {
    transform: scale(2, .5);
}
```

Но, в отличие от функции `translate()`, если поддерживается только одно значение для функции `scale()`, оно применяется как коэффициент изменения масштаба в обоих направлениях. Указание `scale(2)` аналогично применению `scaleX(2)` и `scaleY(2)`, что соответствует интуитивному представлению. На рис. 18.10 показаны результаты трансформации размера.



`transform:
scale(1.25);`



`transform:
scale(.75);`



`transform:
scale(105, .5);`

Рис. 18.10. Изменение размера элемента с помощью функции `scale()`

Свойства `skew()` (наклон)

Для добавления наклона служат свойства `skewX()`, `skewY()` и сокращенное свойство `skew()` — они обеспечивают изменение угла либо вдоль горизонтальной, либо вдоль вертикальной оси (или вдоль обоих осей) на указанное число градусов. Как и в случае со свойством `translate()`, если указано одно значение, оно применяется для `skewX()`, а значение `skewY()` устанавливается равным нулю.

Лучший способ понять, как работает наклон — взглянуть на некоторые примеры (рис. 18.11):

```
a img {
    transform: skewX(15deg);
}
a img {
    transform: skewY(30deg);
}
a img {
    transform: skew(15deg, 30deg);
}
```



Рис. 18.11. Добавление наклона с помощью функции `skew()`

Множественное применение трансформаций

К одному элементу можно применить несколько трансформаций, указав через пробелы функции и их значения:

```
transform: функция(значение) функция(значение);
```

В следующем примере (рис. 18.12) — если указатель мыши наведен на изображение леса или оно находится в фокусе — изображение увеличивается, немножко наклоняется и смещается вниз и вправо:

```
img:hover, img:focus {
    transform: scale(1.5) rotate(-5deg) translate(50px,30px);
}
```

Исходное состояние



:hover, :focus
Применены функции `rotate()`, `translate()` и `scale()`



Рис. 18.12. К одному элементу применены функции `scale()`, `rotate()` и `translate()`

Важно отметить, что трансформации применяются в порядке указания. К примеру, если сначала применить функцию `translate()`, а затем `rotate()`, результат будет отличаться от применения сначала функции `rotate()`, а затем функции `translate()`. То есть порядок имеет значение. Есть и еще один момент, на который следует обратить внимание: если вы захотите применить дополнительную трансформацию

к другому состоянию (такому, как `:hover`, `:focus` или `:active`), нужно повторить все трансформации, которые были ранее применены к элементу. Так, в следующем примере элемент повернут на 45 градусов в своем нормальном состоянии. И если применить трансформацию `scale()` при состоянии `hover`, вращение будет утеряно, если только не объявить его снова явным образом:

```
a {  
    transform: rotate(45deg);  
}  
a:hover {  
    transform: scale(1.25); /* вращение элемента будет утеряно */  
}
```

Чтобы применить и вращение, и изменение масштаба, укажите для трансформации оба значения:

```
a:hover {  
    transform: rotate(45deg) scale(1.25);  
/* вращение и изменение масштаба */  
}
```

Пла-а-а-авные трансформации

Множественная трансформация, примененная к изображению леса красных деревьев, выглядит интересно, но, возможно, *будет лучше*, если создать для него плавную анимацию, а не задать один переход скачком! Теперь, когда вам известно о переходах и трансформациях, соединим эти возможности и добавим немного «магии». Под «магией», конечно же, я здесь понимаю некоторые базовые анимационные эффекты между двумя состояниями. Выполним это шаг за шагом в *упражнении 18.2*.

УПРАЖНЕНИЕ 18.2. ПЕРЕХОДНЫЕ ТРАНСФОРМАЦИИ

При выполнении этого упражнения мы — с использованием плавных переходов — создадим галерею фотографий, которые будут плавно увеличиваться и наклоняться на заданный угол при наведении на них указателя мыши (рис. 18.13). Исходный документ находится в файле `exercise_18-2.html`, а все изображения доступны в папке `materials` для этой главы.

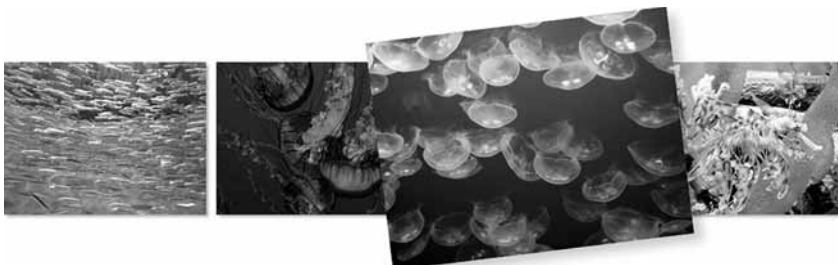


Рис. 18.13. Фотографии увеличиваются и наклоняются — состояния `:hover` и `:focus`. Для сглаживания изменений между состояниями применяется переход. Для окончания работы над этим упражнением (или на странице `ch18_figures.html`) вы можете увидеть, как это функционирует

1. Откройте файл **exercise_18-2.html** в текстовом редакторе, и вы заметите, что в нем уже есть стили, которые располагают элементы списка по горизонтали и добавляют небольшую тень. Первое, что мы сделаем, — добавим для каждого изображения свойство `transform`.
2. Желательно, чтобы трансформации выполнялись только в том случае, когда указатель мыши находится над изображением или же изображение имеет фокус, поэтому свойство `transform` следует применить к состояниям `:hover` и `:focus`. Поскольку каждое изображение наклоняется по-своему, нужно для каждого из них создать собственное правило, применяя в качестве селектора его ID (уникальный идентификатор). По окончании работы можно сохранить ее результаты и проверить их:

```
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);
}
```

ЕЩЕ О ПРИМЕНЕНИИ ПРЕФИКОВ...

На момент подготовки книги для свойства `transform` еще рекомендуются префиксы, поэтому для реально применяемого кода полное правило имеет вид:

```
a:hover #img1, a:focus #img1 {
    -webkit-transform: rotate(-3deg);
    -ms-transform: rotate(-3deg); /* for IE9 */
    transform: rotate(-3deg);
}
```

Впрочем, так как наша работа проверяется в современном браузере, для этого упражнения префиксы мы можем опустить.

3. Теперь немного увеличим изображения, чтобы посетители страницы могли лучше их рассмотреть. Для этого добавим свойство `scale(1.5)` к каждому значению `transform`. Далее показано, как сделать это для первого изображения, а остальное — за вами:

```
a:hover #img1 {
    transform: rotate(-3deg) scale(1.5);
}
```

Обратите внимание, что файлы изображений исходно создаются в большом размере, а затем для просмотра уменьшаются. Если же начать с небольших изображений с последующим их увеличением, они будут выглядеть неаккуратно.

4. При создании видимости отделения фотографий от экрана, увеличивая смещение и размытие, сделаем так, чтобы тень от них падала немного дальше, и так-

же осветлим оттенок серого. Все изображения должны иметь похожий эффект, поэтому добавьте одно правило, используя в качестве селектора `a:hover img`:

```
a:hover img {  
    box-shadow: 6px 6px 6px rgba(0,0,0,.3);  
}
```

Сохраните файл и просмотрите его в браузере — изображения при наведении на них указателя мыши должны наклоняться и укрупняться. Но такое изменение, выполняемое скачком, немножко раздражает. Исправим это с помощью перехода.

- Добавим сокращенное свойство `transition` к нормальному состоянию `img` (то есть не для `:hover` или `:focus`). При этом для свойства `transform` формируется переход. Установим длительность, равную 0,3 секунды, и применим функцию времени `linear`:

```
img {  
    ...  
    transition: transform 0.3s linear;  
}
```

И СНОВА О ПРИМЕНЕНИИ ПРЕФИКОСОВ...

В контексте перехода свойство `transform` также должно включаться с префиксами, как показано в следующем полном префиксном объявлении:

```
-webkit-transition: -webkit-transform .3s linear;
```

А вот префикс `-ms-` не понадобится, поскольку переходы не поддерживаются IE9. Его пользователи увидят скачкообразное изменение преобразованного изображения — без плавного перехода, что, в общем, вполне удовлетворительно.

Это все, что нужно сделать! Вы можете еще поэкспериментировать с разными длительностями и функциями времени или попробовать другие трансформации или их исходные точки, получая любые эффекты, которые вы только сможете придумать.

Трехмерные трансформации

В дополнение к только что рассмотренным двумерным трансформациям спецификация CSS Transforms также описывает систему, применяемую для имитации трехмерного пространства и перспективы. В сочетании с переходами трехмерные трансформации можно применять для создания многофункциональных интерактивных интерфейсов, таких, как карусели изображений, карты с переворотом или вращающиеся кубики! На рис. 18.14 показано несколько примеров интерфейсов, созданных с помощью трехмерных трансформаций.

Заметим, что этот метод не формирует трехмерных объемных изображений — части плоского элемента просто наклоняются относительно трех осей. Эксперт по анимации Вэл Хэд (Val Head) называет их «открытками в космосе» («postcards in space»). Пример изображения вращающегося куба, показанного на рис. 18.14, внизу справа, состоит из комбинации шести элементных блоков, расположенных под разными углами. Тем не менее трехмерные трансформации добавляют плоской веб-странице интересную глубину.



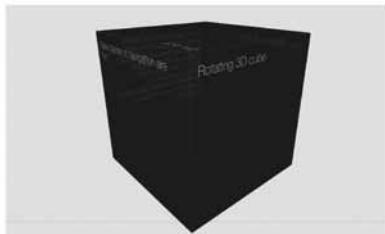
Анимированные обложки книг Марко Баррия (tympanus.net/Development/AnimatedBooks/)



Инструмент трансформации Webflow с примерами (3d-transforms.webflow.com)



Анимация постера фильма Марко Койпера (demo.marcofolio.net/3d_animation_css3/)



Трехмерный вращающийся куб CSS от Пола Хэйса (Paul Hayes) (paulrhayes.com/experiments/cube-3d/)

Рис. 18.14. Примеры трехмерных трансформаций. Обложки книг, постеры фильмов и трехмерный куб представляют интересные анимационные эффекты, поэтому имеет смысл перейти по ссылкам и посмотреть на них. Webflow (справа вверху) — это инструмент визуального веб-дизайна, включающий возможность создания трехмерных трансформаций для элементов

Трехмерные трансформации входят в обязательную программу обучения для пользователей, начинающих заниматься веб-дизайном, поэтому я не буду вдаваться в детали, а рассмотрю лишь основные положения, которые помогут вам почувствовать вкус к добавлению в свой дизайн третьего измерения. Если же вы желаете узнать больше, лучше начать со следующих уроков (хотя информация о поддержке браузеров, которую они содержат, может быть устаревшей):

- Питер Гасстон (Peter Gasston), «Adventures in the Third Dimension: CSS 3D Transforms» (coding.smashingmagazine.com/2012/01/06/adventures-in-thethird-dimension-css-3-d-transforms/);
- Дэвид ДеСандро (David DeSandro), «Intro to CSS 3D Transforms» (desandro.github.com/3dtransforms/).

Чтобы показать вам трехмерную трансформацию на простом примере, воспользуемся изображениями из *упражнения 18.2* и расположим их так, как будто они находятся в трехмерной галерее, выполненной в стиле 3D-карусели (рис. 18.15).

Разметка для этого примера представляет собой тот же маркированный список, который присутствовал в *упражнении 18.2*:

```
<ul>
  <li><a href=""></a></li>
```

```
<li><a href=""></a></li>
<li><a href=""></a>
</li>
<li><a href="">
</a></li></ul>
```

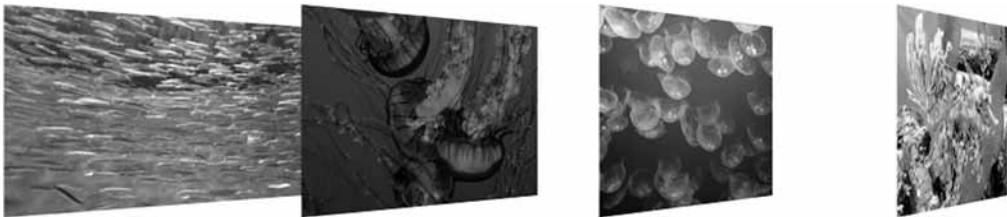


Рис. 18.15. Наши изображения находятся в космосе... космосе... космосе

На первом шаге мы добавим небольшую «перспективу» к содержащему элементу с помощью свойства `perspective`. Браузер при этом получит указание, что дочерние элементы должны вести себя так, как будто они находятся в трехмерном пространстве. Значение свойства `perspective` — это некоторое положительное целое число, определяющее расстояние от исходного элемента на оси `z`. Чем меньше значение, тем круче перспектива. Я полагаю, что значения от 300 до 1500 вполне подойдут, но для получения необходимого эффекта придется поэкспериментировать:

```
ul {
    width: 1000px;
    height: 100px;
    list-style-type: none;
    padding: 0;
    margin: 0;
    perspective: 600;
}
```

ЕЩЕ О ПРИМЕНЕНИИ ПРЕФИКОВ...

При использовании для свойства `transform` префикса `-webkit-` включите версию с префиксом также и для `perspective` в виде `-webkit-perspective`.

Свойство `perspective-origin` (не показано) описывает положение ваших глаз относительно трансформированных предметов. Значения служат горизонтальной позицией (`left`, `center`, `right` или заданная в процентах величина) и вертикальной позицией (`top`, `bottom`, `center` или заданная в процентах величина). По умолчанию трансформация (см. рис. 18.15) центрирована по вертикали и горизонтали (`perspective-origin: 50% 50%`). Третьим свойством, относящимся к трансформациям, является `backface-visibility` — это свойство управляет тем, видна ли обратная сторона элемента при вращении.

Для задания трехмерного пространства примените к каждому дочернему элементу одну из функций трехмерной трансформации — в этом случае элемент `li` находится внутри элемента `ul`. При этом используются следующие трехмерные функции: `translate3d`, `translateZ`, `scale3d`, `scaleZ`, `rotate3d`, `rotateX`, `rotateY`, `rotateZ` и `matrix3d`. Уточним, что функции `*Z` определяют ориентацию объекта относительно оси `z` (представьте, что он убегает от вас по направлению к этой странице, тогда как оси `x` и `y` находятся непосредственно на ней).

В примере, представленном на рис. 18.15, каждый элемент списка `li` с помощью функции `rotateY` поворачивается на 45 градусов вокруг своей оси `y` (вертикальной оси), что можно представить так, как будто блоки элементов вращаются на нити, натянутой между полом и потолком.

Сравните с этим результатом, показанный на рис. 18.16, где элементы списка `li`, используя функцию `rotateX`, поворачиваются вокруг своей оси `x` (горизонтальной оси) так, как будто блоки элементов вращаются на нити, натянутой от стены к стене:

```
li {  
    float: left;  
    margin-right: 10px;  
    transform: rotateX(45deg);  
}
```

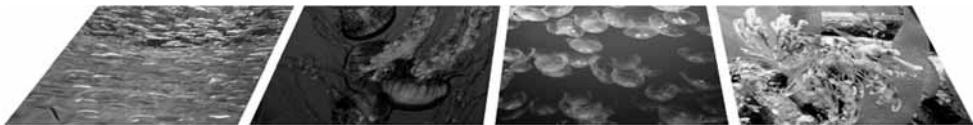


Рис. 18.16. Те же изображения, что и показанные на рис. 18.15, вращаются вокруг горизонтальных осей с помощью функции `rotateX()`

Очевидно, что здесь я лишь слегка «поцарапала поверхность» спектра возможностей, которые доступны при работе с трехмерными трансформациями, но это должно помочь вам понять ее основные посылки. Далее я познакомлю вас с более изощренным способом приведения ваших веб-страниц в движение.

АНИМАЦИЯ ПО КЛЮЧЕВЫМ КАДРАМ

CSS-ПЕРЕХОДЫ: АНИМАЦИЯ ПО ДВУМ КЛЮЧЕВЫМ КАДРАМ

CSS-переходы — это анимация по двум ключевым кадрам: начальное состояние и конечное состояние. Более сложные анимации нуждаются в большем числе ключевых кадров для управления изменениями свойств в последовательности.

АНИМАЦИЯ ПО КЛЮЧЕВЫМ КАДРАМ

Анимация по ключевым кадрам — это явная анимация, поскольку ее поведение программируется. В отличие от нее, переходы служат примером неявной анимации, поскольку они выполняются только при изменении свойства.

Модуль CSS Animations Module позволяет разработчикам создавать настоящую, истинную и качественную анимацию по ключевым кадрам. На рис. 18.17 показано несколько примеров, которые вы можете посмотреть в действии онлайн.

В отличие от переходов, когда формируется переход из начального состояния в конечное, анимация по ключевым кадрам позволяет явно указывать промежуточные состояния в определенных точках пути, обеспечивая пошаговое управление действием. Эти «точки пути» определяют **ключевые кадры**, которые задают начало или конец сегмента анимации.



МАДМАНИМАЦИЯ Энтона Кальзадиллы и Энди Кларка (stuandnonsense.co.uk/content/demo/madmanimation/) — подсказка: щелкните на WATCH



«Как я учился ходить» от Эндрю Ван-Хойера (andrew.wang-hoyer.com/experiments/walking/)



Восхитительная анимированная подводная лодка Альберто Хереса (codepen.io/ajerez/pen/EaEEOW)



Анимированный веб-баннер от Калеба Джейкоба (tympanus.net/codrops/2012/01/10/animated-web-banners-with-css3/)

Рис. 18.17. Примеры анимации, использующей только CSS

Создание анимации по ключевым кадрам — задача непростая, поэтому в этой главе будут рассмотрены лишь ее основы. Если же вы хотите получить дополнительную информацию по этой теме, обратитесь к следующим ресурсам:

- CSS Animations Level 1 (рабочий проект на момент подготовки книги) — доступно по адресу: www.w3.org/TR/css-animations-1/;
- Эстель Вейль (Estelle Weyl), «Transitions and Animations in CSS», издательство О’Рейли;
- «Animation & UX Resources» от Вэла Хеда (Val Head) (valhead.com/ui-animation/). Вэл составил громадный список ресурсов веб-анимации, включая ссылки на учебные пособия, статьи, инструменты, галереи и многое другое. И он не ограничился анимацией по ключевым CSS-кадрам, поэтому здесь вы найдете массу ценной информации;

ИНСТРУМЕНТЫ АНИМАЦИИ

Если к элементу необходимо добавить простой эффект анимации — быстрый поворот или небольшое перемещение, можно подобрать уже готовый эффект и затем применить к своему дизайну. Вот несколько сайтов, которые предоставляют готовую CSS-разметку для обычных анимационных эффектов (некоторые также используют плагины JQuery, объясняя при этом методику их использования):

- Даниэль Иден (Daniel Eden), «Animate.css» — daneden.github.io/animate.css/;
- Джастин Агилар (Justin Aguilar), «CSS Animation Cheat Sheet» — www.justinaguilar.com/animations/index.html;
- «AngryTools CSS Animation Kit» — angrytools.com/css/animation/.

- Курс «CSS: Animation» от Вэла Хеда (Val Head), доступный на сайте Lynda.com (www.lynda.com/CSS-tutorials/CSS-Animation/439683-2.html). Для знакомства с этим курсом вам потребуется подписка на сайт Lynda.com, но если вы обучаетесь веб-дизайну, то «овчинка стоит выделки»;
- Рэйчел Коуп (Rachel Cope), «CSS Animation for Beginner» (robots.thoughtbot.com/css-animation-for-beginners) — толково написанное руководство с большим числом примеров;
- Том Уотерхаус (Tom Waterhouse), «The Guide to CSS Animation: Principles and Examples» (www.smashingmagazine.com/2011/09/the-guide-to-css-animation-principles-and-examples/) — это руководство выходит за рамки программирования на CSS и содержит советы по созданию естественных эффектов анимации.

Настройка ключевых кадров

Процесс создания анимации состоит из двух частей:

1. Установка ключевых кадров с помощью правила @keyframes.
2. Добавление свойств анимации к элементам, которые предполагается анимировать.

В следующем примере приведен простой набор ключевых кадров, который постепенно изменяет цвет фона элемента. Это не очень активная анимация, но она должна дать вам основы понимания принципа функционирования правила @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 правилах для ключевых кадров указывается имя анимации, ее этапы, представленные в процентах (%), и CSS-свойства, влияющие на каждый этап. Далее приводится правило @keyframes, демонстрирующее соответствующий синтаксис:

```
@keyframes animation-name {  
    ключевой кадр { свойство: значение; }  
    /* дополнительные ключевые кадры */  
}
```

Приведенное чуть ранее простое правило @keyframes предлагает создать последовательность анимации под названием `colors`. В начале анимации цвет фона элемента (свойство `background-color`) должен быть красным (`red`), по прошествии 20% времени выполнения анимации цвет фона должен стать оранжевым (`orange`) и так далее, пока не будет достигнуто завершение анимации. При этом браузер дополняет все оттенки цветов между ключевыми кадрами (то есть выполняет плавные переходы цвета в цвет). Это хорошо видно на рис. ЦВ-18.18.

Каждое процентное значение и объявление свойства/значения определяют ключевой кадр в последовательности анимации.

В качестве альтернативы процентным соотношениям можно использовать ключевое слово `from` для обозначения начала последовательности анимации (эквивалентно 0%) и ключевое слово `to` для обозначения конца анимации (эквивалентно 100%). В следующем примере элемент перемещается справа налево, а левое поле уменьшается до 0:

```
@keyframe slide {  
    from { margin-left: 100% }  
    to { margin-left: 0%; }  
}
```

Добавление свойств анимации

Теперь можем применить эту последовательность анимации к элементу или нескольким элементам в документе, используя набор свойств анимации, которые очень похожи на известный вам набор свойств перехода.

В следующем примере я применила к элементу `#magic` `div` анимацию в виде радужного перехода:

```
<div id="magic">Magic!</div>
```

В CSS-правиле для `#magic` определяются применяемые анимационные свойства:

- какая анимация используется (`animation-name`) — *требуется*;
- как долго она длится (`animation-duration`) — *требуется*;
- способ ее ускорения (`animation-timing-function`). Это свойство использует те же ключевые слова функции времени, с которыми мы уже познакомились при рассмотрении CSS Transitions;
- будет ли пауза перед ее началом (`animation-delay`).

ПОДДЕРЖКА БРАУЗЕРАМИ КЛЮЧЕВЫХ КАДРОВ CSS

Все текущие версии основных настольных и мобильных браузеров поддерживают анимацию по ключевым кадрам CSS без префиксов поставщиков. Далее приводится список исключений:

- браузер Internet Explorer 9 и более ранние его версии вообще не поддерживают анимацию по ключевым кадрам. Соответственно, эти браузеры покажут только начальный кадр анимации, поэтому убедитесь, что этот кадр может служить доступным резервным вариантом;
- следует использовать префикс `-webkit-` для поддержки следующих браузеров: Safari и iOS Safari 8, а также их более ранних версий (2014), Chrome 41 и его более ранних версий (2015), Opera 29 и ее более ранней версии (2015), а также Android 4.4.4 и его более ранней версии (2014). На момент подготовки книги эти браузеры поддерживают достаточный объем трафика, однако все равно включать префикс `-webkit-` рекомендуется. Тем не менее он может применяться или нет в зависимости от того, когда вы начали заниматься разработкой и кто входит в вашу целевую аудиторию.

Обратите внимание, что, как правило, вам понадобится префиксный ключевой кадр, а также префиксные свойства `animation-*`. Как всегда, стандартные правила без префиксов идут после версий с префиксами:

```
@-webkit-keyframes  
-webkit-animation-*
```

Знакомо, не правда ли? Существует также несколько других специфических для анимации свойств:

- `animation-iteration-count` — количество повторов анимации. Задается с помощью целого числа или ключевого слова `infinite`;
- `animation-direction` — будет ли анимация воспроизводится в направлении вперед (`normal`), назад (`reverse`) или же чередоваться в направлении вперед/назад, но при этом начинаться в начальной точке (`alternate`) или в конечной (`alternate-reverse`);
- `animation-fill-mode` — режим заполнения анимации определяет, что происходит с анимацией до ее начала и по окончании. По умолчанию (`none`) анимация показывает, какие значения свойств не были указаны с помощью `@keyframes`. Если необходимо, чтобы последний ключевой кадр оставался видимым после воспроизведения анимации, используйте ключевое слово `forwards`. Если для анимации выбрана задержка, но нужно, чтобы первый ключевой кадр показывался во время этой задержки, используйте ключевое слово `backwards`. Для сохранения начального и конечного состояний применяйте ключевое слово `both`;
- `animation-play-state` — определяет, должна ли анимация работать (`running`) или приостанавливаться (`paused`) при загрузке. Состояние воспроизведения может включаться и выключаться на основе пользовательского ввода с помощью JavaScript или при наведении указателя мыши.

Свойство `animation-name` указывает браузеру, какая последовательность ключевых кадров применяется к элементу `#magic` `div`. Я установила длительность (`duration`) и функцию времени (`timing function`), а также применила свойство `animation-iteration-count`, что позволит бесконечно выполнять повтор радуги. Можно, конечно, указать конкретное числовое значение — например, 2, чтобы воспроизвести ее дважды, но будет ли интересен эффект двух переходящих радуг? И, ради интереса, я присвоила свойству `animation-direction` значение `alternate`, что заставляет анимацию воспроизводиться в обратном направлении после воспроизведения вперед. В итоге правило для анимированного элемента `div` теперь выглядит так:

```
#magic {  
  ...  
  animation-name: colors;  
  animation-duration: 5s;  
  animation-timing-function: linear;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}
```

Как можно видеть, объем кода нарастает, особенно если учесть, что каждое свойство может также иметь префиксную версию. Впрочем, вы можете применять сокращенное свойство `animation` для объединения значений, как это сделано для свойства `transition` в следующем примере:

```
#magic {  
    animation: colors 5s linear infinite alternate;  
}
```

Итак, мы рассмотрели основные возможности создания ключевых кадров и применения анимации к элементу на странице. Чтобы заставить элементы перемещаться (что обычно и называют «анимацией»), используйте ключевые кадры для изменения положения элемента на экране с помощью свойства `translate` (лучший вариант с точки зрения производительности) или с помощью свойств `top`, `right`, `bottom` и `left`. Если ключевые кадры анимированы, объект будет плавно перемещаться из одного положения в другое. Вы также можете анимировать и другие функции трансформации — такие, как `scale` и `skew`.

Когда применять анимацию по ключевым кадрам?

Чтобы упростить мой пример, я решила изменять только цвет фона элемента кнопки, но, конечно, анимацию по ключевым кадрам можно применять и для создания реальной анимации, особенно в сочетании с CSS-функциями трансформации для вращения и перемещения элементов по странице. Когда вам нужно только изменить одно состояние элемента на другое, удобнее применять переход. Но, если подразумевается своего рода художественная анимация — например, перемещение персонажа, объекта или окружающих его предметов, анимация по ключевым кадрам окажется наиболее подходящим выбором.

Впрочем, для более сложных анимаций по ключевым кадрам, особенно тех, которые зависят от взаимодействия с пользователем или требуют сложной физики, все же предпочтительнее вместо CSS-анимации применять JavaScript. Анимация JavaScript также лучше поддерживается устаревшими браузерами, что дает ей явное преимущество, если анимация имеет определяющее значение для задач, решаемых на странице. Так что CSS-анимация по ключевым кадрам — это хорошее решение для простых анимаций, используемых для улучшения общего вида страницы.

Следует отметить, что сейчас в веб-сообществе появилось очень много интересных разработок по анимации SVG-графики. Когда вы размещаете исходный код SVG непосредственно в документе HTML, его элементы становятся доступными для анимации. На момент подготовки этой книги ограничения и проблемы с поддержкой браузеров в части использования CSS для анимации SVG все еще существуют, однако по мере роста такой поддержки этот подход выглядит очень многообещающим. В то же время JavaScript располагает лучшим доступом к свойствам SVG, имеет более широкую поддержку со стороны браузеров и является более распространенным решением для SVG-анимации.

НЕСКОЛЬКО ЗАВЕРШАЮЩИХ СЛОВ

Я надеюсь, что помогла вам разобраться в том, каким образом можно применять CSS, чтобы добавить немного движения и плавности на ваши страницы. Для добавления

ИНСПЕКТОРЫ АНИМАЦИИ

Браузеры Chrome и Firefox предлагают инструменты для проверки и внесения изменений в веб-анимацию (рис. 18.19). При просмотре анимированного элемента выберите пункт **Developer Tools** (Инструменты разработчика), а затем перейдите на вкладку **Animations** (Анимации) для просмотра временной шкалы всех примененных к этому объекту анимаций. Вы сможете замедлить анимацию, чтобы показать происходящее более детально. Вы также сможете подправить анимацию, внести изменения в функцию времени, изменить время задержек, длительность и ключевые кадры. Для получения дополнительной информации обратитесь к следующим материалам:

- Firefox Animation Inspector — developer.mozilla.org/en-US/docs/Tools/Page_Inspector/How_to/Work_with_animations;
- Chrome Animation Inspector — developers.google.com/web/tools/chromedevtools/inspect-styles/animations.

Firefox Animation Inspector



Chrome Animation Inspector

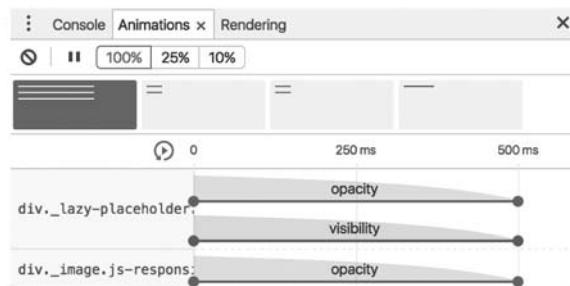


Рис. 18.19. Инспекторы анимации являются частью инструментов разработчика, предлагаемых браузерами Firefox (вверху) и Chrome (внизу)

движения на веб-страницу применяется модуль CSS Transitions, сглаживающий переходы из одного состояния в другое, а также модуль CSS Keyframe Animation — для анимации ряда состояний. Мы также рассмотрели модуль CSS Transforms, с помощью которого осуществляются изменение положения, выполнение вращения, изменение размера и наклон элемента при его отображении на экране.

Продуманная анимация может сделать ваши интерфейсы более привлекательными и повысить индивидуальность вашего бренда. Благодаря ей вы получите большие возможности, но не забывайте, что с ними связана и большая ответственность. Для того чтобы использовать веб-анимацию как средство серьезного улучшения взаимодействия с пользователем, я рекомендую книгу Вэла Хеда (Val Head) «*Designing*

Interface Animation: Meaningful Motion for User Experience» (издательство Rosenfeld Media).

А теперь попробуйте ответить на следующие 12 вопросов!

КОНТРОЛЬНЫЕ ВОПРОСЫ

Думаете, вам известно, как обрабатывать переходы, трансформации и анимацию по ключевым кадрам? Далее приводятся несколько вопросов, которые позволят вам проверить свои знания (ответы приведены в *приложении 1*).

1. Что такое твининг?
2. Если бы у перехода были ключевые кадры, сколько бы он мог их иметь?
3. Запишите объявление перехода (свойство и значение), которое можно применить для выполнения следующих задач:
 - a. Подождите 0,5 секунды до начала перехода.
 - b. Сделайте так, чтобы переход проходил с постоянной скоростью.
 - c. Сделайте переход за последние 0,5 секунды.
 - d. Сделайте так, чтобы строки текста расширялись постепенно.
4. Что из перечисленного нельзя анимировать?
 - a. width
 - b. padding
 - c. text-transform
 - d. word-spacing
5. Какая временная функция применяется, если опускается свойство transition-timing function? Опишите ее действие.

НУЖНО НЕМНОГО ВДОХНОВЕНИЯ?

Блог Codrops (tympanus.net/codrops/), курируемый Мануэлой Илик (Manoela Illic) и Педро Ботело (Pedro Botelho), — сокровищница примеров CSS-переходов, трансформаций и анимации. Посетите игровую площадку (раздел **Playground**) и познакомьтесь с выложенными там интересными экспериментами (например, с коллекцией эффектов наведения, наподобие показанных на рис. 18.20) и раздел **Tutorials** для получения пошаговых инструкций с примерами кода.



Рис. 18.20. Один из многих примеров CSS-переходов, трансформаций и анимаций, представленных в блоге Codrops

6. Что означает в следующем переходе запись `.2s`?

```
transition: color .2s linear;
```

7. Какой переход завершится первым?

- a. `transition: width 300ms ease-in;`
- b. `transition: width 300ms ease-out;`

8. Запишите объявление трансформации для выполнения следующих действий:

- a. Наклоните элемент на 7 градусов по часовой стрелке.
- b. Переместите элемент на 25 пикселов вверх и на 50 пикселов влево.
- c. Поверните элемент относительно его правого нижнего угла.
- d. Увеличьте изображение шириной 400 пикселов до ширины 500 пикселов.

9. В следующем объявлении преобразования что именно описывает значение `3`?

```
transform: scale(2, 3)
```

10. Какое трехмерное преобразование выглядит более угловатым и драматичным?

- a. `perspective: 250;`
- b. `perspective: 1250;`

11. Что происходит в середине этой анимации?

```
@keyframes border-bulge {  
    from { border-width: 1px; }  
    25% { border-width: 10px; }  
    50% { border-width: 3px; }  
    to { border-width: 5px; }  
}
```

12. Напишите объявление анимации, которое применялось для выполнения следующего:

- a. Заставьте анимацию идти в обратном порядке.
- b. Выполните всю анимацию за последние 5 секунд.
- c. Подождите 2 секунды, прежде чем запустить анимацию.
- d. Повторите анимацию три раза, а затем остановитесь.
- e. Конечное состояние анимации остается видимым после завершения воспроизведения анимации.

ОБЗОР CSS: ПЕРЕХОДЫ, ТРАНСФОРМАЦИИ И АНИМАЦИЯ

В табл. 18.2 в алфавитном порядке приводятся краткие описания рассмотренных в этой главе свойств.

Таблица 18.2. Свойства переходов, трансформации и анимации

Свойство	Описание
animation	Сокращенное свойство, объединяющее свойства анимации
animation-name	Указывает именованную последовательность анимации для применения
animation-duration	Определяет время, которое длится анимация
animation-timing-function	Описывает ускорение анимации
animation-iteration-count	Указывает количество повторений анимации
animation-direction	Определяет, будет ли анимация воспроизведаться вперед, назад или чередоваться вперед/назад
animation-play-state	Указывает, запущена ли анимация, или приостановлена
animation-delay	Указывает количество времени до запуска анимации
animation-fill-mode	Переопределяет ограничения, когда свойства анимации могут быть применены
backface-visibility	Определяет, может ли обратная сторона элемента быть видимой при трехмерной трансформации
perspective	Устанавливает элемент как трехмерное пространство и определяет воспринимаемую глубину
perspective-origin	Определяет положение вашей точки зрения в трехмерном пространстве
transform	Указывает, что визуализация элемента должна быть изменена с помощью одной из двумерных или трехмерных функций трансформации
transform-origin	Обозначает точку, вокруг которой трансформирован элемент
transform-style	Сохраняет трехмерный контекст, когда преобразованные элементы вложены
transition	Сокращенное свойство, объединяющее свойства перехода
transition-property	Определяет, какое свойство CSS будет перенесено
transition-duration	Определяет время, которое длится анимация перехода
transition-timing-function	Описывает способ, которым происходит переход (изменения в ускорении)
transition-delay	Определяет количество времени до начала перехода

ГЛАВА 19

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О CSS-МЕТОДИКАХ

В этой главе...

- ▶ *Стайлинг форм*
- ▶ *Специальные свойства для форматирования таблиц*
- ▶ *CSS reset и Normalize.css*
- ▶ *Подмена текста изображением*
- ▶ *CSS-спрайты*
- ▶ *Проверка поддержки свойств CSS*

К этому моменту у вас уже имеется прочная основа для написания таблиц стилей. Вы научились присваивать стили тексту и блокам элементов, создавать плавающие объекты и перемещать их, формировать, используя Flexbox и Grid, адаптивные (отзывчивые) макеты страниц и даже привносить в свои проекты утонченные эффекты анимации. Но есть и еще несколько достаточно распространенных методик, о которых вам тоже следует знать.

Если вы посмотрите на список «В этой главе», то увидите, что она — своего рода сборник рецептов. Глава начинается с представления общих подходов к стайлингу форм и специальным свойствам для форматирования таблиц. А далее в ней будет рассказано о таких приемах, как очистка стилей браузера с помощью CSS reset (сброс CSS), использование изображений вместо текста (только, когда это необходимо!), уменьшение количества запросов к серверу с использованием CSS-спрайтов и проверка поддержки браузерами тех или иных свойств CSS. Что ж, приступим!

СТАЙЛИНГ ФОРМ

Без применения стилей веб-формы могут выглядеть неаккуратно (рис. 19.1), и вы наверняка захотите придать им более профессиональный вид с помощью CSS. Однако только этого, как правило, недостаточно — исследования показывают, что формами намного проще и быстрее пользоваться, если надписи и поля ввода данных в них правильно скомпонованы. В этом разделе мы рассмотрим, как выполняется стайлинг различных элементов форм.

Custom Sneaker Order Form

- Name:
- Email:
- Telephone:

- Tell us about yourself:
- Size: Sizes reflect standard men's sizes
- Sneaker Color:
 - Red
 - Blue
 - Black
 - Silver

- Add-on Features:
 - Feature
 - Sparkley laces
 - Metallic logo
 - Light-up heels
 - MP3-enabled

-

Рис. 19.1. Если формы созданы исключительно с помощью HTML, они могут выглядеть уродливо и быть неудобными для использования. Однако не все так плохо — обратите внимание, как выглядит эта же форма на рис. 19.2.

КООРДИНАЦИЯ ШРИФТОВ

По умолчанию браузер может применять для тех или иных вариантов надписей на формах различные шрифты разных размеров. Например, для кнопок он может применить системный шрифт, а для вводимого текста — моноширинный шрифт типа Courier. Если вы хотите, чтобы все вводимые на странице данные использовали тот же шрифт, что и шрифт окружающих надписей, можно принудительно наследовать элементы формы:

```
font settings:
button, input, select,
textarea {
    font-family: inherit;
    font-size: 100%
}
```

Здесь я не собираюсь вам лгать — стайлинг форм представляет собой нечто вроде темной магии из-за разнообразия способов, которыми браузеры обрабатывают их элементы. И для создания действительно изящных пользовательских форм вам, как правило, придется обратиться к JavaScript. Однако удобство использования форм можно значительно повысить и с помощью CSS.

Поскольку для стайлинга форм отсутствуют специальные CSS-свойства, воспользуйтесь стандартными свойствами цвета, фона, шрифта, границ, полей и отступов, изученными нами в предыдущих главах. Далее приводится краткий обзор свойств, которые можно применить для каждого типа элемента управления формы:

- поля ввода текста (text, password, email, search, tel, url) — изменение внешнего вида самого блока с помощью свойств: width, height, background-color, background-image, border, border-radius, margin, padding и box-shadow. Вы также можете форматировать текст внутри поля с помощью свойства color и различных свойств шрифта;

- элемент `textarea` — этому элементу можно присваивать стили так же, как и полям ввода текста. Элементы `textarea`, по умолчанию, используют моноширинный шрифт, поэтому вы можете привести его в соответствие с другими полями ввода текста. Поскольку элемент может содержать несколько строк, вы также можете задать высоту строки. Обратите внимание, что некоторые браузеры в правом нижнем углу области `textarea` отображают маркер, с помощью которого можно изменять размер этой области, но вы можете отключить такой маркер, добавив стиль `resize: none`. Области текста отображаются, по умолчанию, как `inline-block`, но вы можете изменить их на значение `block` с помощью свойства `display`;
- кнопки ввода данных (`submit`, `reset`, `button`) — для кнопок отправки и сброса данных можно применить любое из свойств полей: `width`, `height`, `border`, `background`, `margin`, `padding` и `box-shadow`. Стоит отметить, что кнопки по умолчанию настроены на размерную модель `border-box`. Большинство браузеров по умолчанию также добавляют некоторое количество отступов, которые могут быть переопределены с помощью значений отступов, заданных пользователем. Вы также можете присвоить стили тексту, который отображается на кнопках;
- переключатели и флагки — лучше всего к этим элементам управления вовсе не прикасаться. Если же вы, тем не менее, хотите их как-то изменить, следует обратиться к возможностям JavaScript;
- выпадающие меню и меню выбора (`select`) — вы можете задать для элемента `select` значения ширины и высоты, но обратите внимание, что по умолчанию для него используются размерные модели `border-box` `box-sizing`. Некоторые браузеры позволяют применять свойства `color`, `background-color` и свойства шрифтов к элементам `option`, но, вероятно, лучше оставить их для внесения изменений со стороны браузеров и операционной системы;
- наборы полей и надписи — элемент `fieldset` можно воспринимать как любой другой блочный элемент, настраивая его границу, фон, поля и отступы. Полное отключение границ — это один из способов придать форме аккуратный вид, сохраняя при этом семантику и доступность. По умолчанию элементы `legend` располагаются выше верхней границы элементов `fieldset`, и, к сожалению, браузеры обрабатывают их с некоторыми затруднениями. Некоторые разработчики применяют элемент `span` или элемент `b` внутри элемента `legend` и добавляют стили к содержащему элементу для получения более предсказуемых результатов. Некоторые предпочитают скрывать его таким образом, чтобы он все же воспринимался программами чтения с экрана (`legend {width: 1px; height: 1px; overflow: hidden;}`).

Теперь мы знаем, что можем предпринять для стайлинга отдельных элементов управления, но наша главная цель — сделать форму упорядоченной и удобной в использовании. На рис. 19.2 показана

ЭЛЕМЕНТЫ ФОРМ, КОТОРЫЕ НЕЛЬЗЯ ИЗМЕНИТЬ С ПОМОЩЬЮ CSS

Обратите внимание, что следующие элементы формы невозможно изменить с помощью CSS: входные данные для диапазона, цвет, выбор даты, выбор файла, option, optgroup, datalist, progress и meter. Для изменения таких элементов используется JavaScript, подробное описание которого выходит за рамки этой книги.

такая форма, созданная на основе формы, приведенной на рис. 19.1. Как видите, изменился цвет, шрифт, границы и отступы, причем названия полей и элементы ввода данных удачно выровнены. И не только это — сама форма стала адаптивной! Я воспользовалась Flexbox, чтобы на узких экранах надписи, используя свободное пространство, размещались над соответствующими полями ввода и наборами полей.

Широкое окно просмотра

Custom Sneaker Order Form

Name:

Email:

Telephone:

Tell us about yourself:

Size: Sizes reflect standard US men's sizes

Sneaker Color: Red Blue Black Silver

Add-on Features: Sparkley laces Metallic logo Light-up heels MP3-enabled

ENTER

Узкое окно просмотра

Custom Sneaker Order Form

Name:

Email:

Telephone:

Tell us about yourself:

Size: Sizes reflect standard US men's sizes

Sneaker Color: Red Blue Black Silver

Add-on Features: Sparkley laces Metallic logo Light-up heels MP3-enabled

ENTER

Рис. 19.2. Эта адаптивная форма использует Flexbox для изменения размеров полей ввода текста и изменения расположения надписей на узких экранах

Если вы хотите увидеть фактическую разметку и стили такой адаптивной формы, найдите документ `sneakerform.html` в папке материалов для этой главы на сайте по адресу: learningwebdesign.com/5e/materials. Я оставила там тщательные и подробные комментарии, которые точно объясняют, для чего нужен каждый стиль. Мой подход к оформлению пользовательской формы «Custom Sneaker Order» можно резюмировать следующим образом:

- присвоить свойству `border-box` документа в целом значение `box-sizing`. Это сделает изменение размеров элементов формы более предсказуемыми;
- присвоить свойству `form` значение `max-width` (тогда форма сможет уменьшаться, чтобы соответствовать меньшим областям просмотра) и, дополнительно, декоративный стиль — зеленый фон и скругленную границу (как в приведенном на рис. 19.2 примере);
- избавиться от маркеров и отступов вокруг неупорядоченных списков, которые использовались при семантической разметке формы;

- превратить элементы списка (каждый из которых содержит надпись и поле ввода данных или набор полей) в гибкие контейнеры, присвоив их свойствам `display` значения `flex`. Подключить обтекание, позволяющее на маленьких экранах смещать поля ввода под надписи;
- придать надписям фиксированные значения ширины (`flex: 0 0 8em;`), чтобы они получали одинаковый размер независимо от размера экрана. Поскольку надписи для флагков и переключателей работают по-разному, установить их так, чтобы они перекрывали ширину `8em` (`flex: 1 1 auto;`);
- разрешить элементам `input`, `textarea` и `fieldsets` увеличиваться и заполнять свободное пространство (`flex: 1 1 20em;`). Если экран слишком узок, чтобы они могли поместиться рядом с надписями, заработает обтекание;
- установить для полей ввода текста свойство `font-family` в значение `inherit`, чтобы в них — вместо любого шрифта, который браузер использует для форм, — использовался тот же шрифт, что и в остальной части документа. Поля ввода текста также получают высоту, границы и небольшие отступы;
- наборы полей и надписи в процессе стайлинга могут вести себя непредсказуемо. Чтобы исправить ситуацию, следует отключить границы и отступы для элемента `fieldset`, а затем скрыть элемент `legend` так, чтобы он по-прежнему мог считываться вслух средствами чтения с экрана перед каждым флагком или переключателем. Поскольку для каждого набора полей имеется и метка, и надпись, которые, как можно видеть, на совсем одинаковы, не будет лишним их чтение вслух программами чтения с экрана. Надпись должна быть короче, поскольку она повторяется для каждого пункта;
- кнопке ввода (`submit`) придать скругленную границу, цвет фона и стиль шрифта. Установить значение `auto` для боковых полей, чтобы они всегда были центрированы по ширине формы.

Это очень простой пример, но он должен дать вам общее представление о том, каким образом может быть выполнен стайлинг формы. Вы также, для повышения степени интерактивности, можете добавить стили выделения: такие, как `:hover` — для кнопок и `:focus` — для выбранных полей ввода текста.

ВЫРАВНИВАНИЕ С ПОМОЩЬЮ ПЛАВАЮЩИХ ЭЛЕМЕНТОВ

Если вы не желаете применять Flexbox, надписи можно выравнивать с помощью плавающих элементов. Установите для надписей значение `display: block`, придайте им ширину и высоту и заставьте их всплыть влево. Нужно также очистить элементы `li` (`clear: both`), чтобы они начинались ниже предыдущей плавающей пары.

СПЕЦИАЛЬНЫЕ СВОЙСТВА ДЛЯ ФОРМАТИРОВАНИЯ ТАБЛИЦ

Как и любой другой текстовый контент на веб-странице, контент в ячейках таблицы можно отформатировать с различными свойствами шрифта, текста и фона.

Возможно, вы захотите определить пространство внутри таблицы и вокруг нее. Для настройки отступов внутри ячейки (*cell padding*) примените свойство `padding` к элементу `td` или `th`. Пространство между ячейками (*cell spacing*) настроить немного сложнее, и это связано с тем, как CSS обрабатывает границы ячеек. В CSS доступны два способа отображения границ между ячейками таблицы: раздельные (*separated*) или схлопнутые (*collapsed*). Эти параметры указываются с помощью специфичного для таблицы свойства `border-collapse` со значениями `separate` и `collapse` соответственно:

border-collapse

- **Значения:** `separate` | `collapse`.
- **По умолчанию** — `separate`.
- **Применение** — к таблице или строчным элементам таблицы.
- **Наследование** — да.

УСТАРЕВШИЕ АТРИБУТЫ

CELLPADDING И CELLSPECING

Ранее отступы внутри ячейки и пространство между ячейками обрабатывались с помощью атрибутов cellpadding и cellspacing соответственно элемента table, но в HTML5 они признаны устаревшими вследствие их презентационной природы.

Раздельные границы

По умолчанию границы ячеек раздельные, и сама граница обрамляет каждую из ячеек со всех ее четырех сторон. Свойство `border-spacing` позволяет указать пространство между границами ячеек:

border-spacing

- **Значения:** `длина по горизонтали` | `длина по вертикали`.
- **По умолчанию** — 0.
- **Применение** — к таблице или строчным элементам таблицы.
- **Наследование** — да.

Значениями `border-spacing` служат два размера по длине. Горизонтальное значение стоит первым и определяет расстояние между столбцами. Второе значение определяет межстрочное расстояние. Если указать одно значение, оно будет применяться и для горизонтали, и для вертикали. По умолчанию установлено значение 0, в результате чего границы на внутренней сетке таблицы удваиваются.

Таблица, показанная на рис. ЦВ-19.3, установлена в `separate` с 15 пикселями пространства между столбцами и 5 пикселями пространства между строками (фиолетовая рамка применена к ячейкам для лучшей видимости их границ).

УДВОЕНИЕ ГРАНИЦ

Хотя значением свойства `border-spacing` по умолчанию является 0, браузеры обычно по умолчанию добавляют пробел шириной 2 пикселя для устаревшего атрибута `cellspacing`. Если вы хотите увидеть удвоение границ, следует присвоить в элементе `table` атрибуту `cellspacing` значение 0.

Для таблиц с раздельными границами с помощью свойства `empty-cell` можно указать, нужно ли, чтобы пустые ячейки отображались с фоном и границами. Чтобы ячейка считалась «пустой», она не должна содержать текст, изображения или неразрывные пробелы, но может включать символы возврата каретки и пробелов:

empty-cells

- **Значения:** show | hide.
- **По умолчанию** — show.
- **Применение** — к элементам табличных ячеек.
- **Наследование** — да.

На рис. 19.4 показан пример предыдущей таблицы с раздельными границами и пустыми ячейками (это ячейки Cell 14 и Cell 15), которым присвоено значение hide.

`empty-cells: hide;`

Cell 1	Cell 2	Cell 3	Cell 4	Cell 5
Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 11	Cell 12	Cell 13		

Рис. 19.4. Скрытие пустых ячеек с помощью свойства empty-cells

Схлопнутые границы

В модели схлопнутых границ границы смежных ячеек «схлопываются» так, что остается видна только одна из границ, а пространство между ячейками удаляется (рис. 19.5). В этом примере, хотя каждая ячейка таблицы имеет 3-пиксельную границу, границы между ячейками составляют всего 3 пикселя, а не 6. В тех случаях когда соседние ячейки имеют разные стили границ, применяется сложный порядок определения, какая именно граница отображается (об этом можно прочесть в спецификации).

```
td {
  border: 3px solid purple;
}
table {
  border-collapse: collapse;
  border: none;
}
```

Cell 1	Cell 2	Cell 3	Cell 4	Cell 5
Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 11	Cell 12	Cell 13		

граница шириной 3 пикселя

Рис. 19.5. Модель схлопнутых границ

Преимущество применения для таблицы модели схлопнутых границ заключается в том, что можно выполнить стайлинг границ для элементов `tr`, `col`, `rowgroup` и `colgroup`, тогда как при использовании модели раздельных границ этого достигнуть не удается. Осознанное использование горизонтальных и вертикальных границ улучшает удобочитаемость сложных таблиц, делая схлопнутую модель привлекательным выбором.

Макет таблицы

table-layout

- **Значения:** auto | fixed.
- **По умолчанию** — auto.
- **Применение** — к таблице или к строчным элементам таблицы.
- **Наследование** — да.

ВЫБЕРИТЕ МЕСТО ДЛЯ ЗАГОЛОВКА

При использовании в таблице элемента `caption` он по умолчанию появится над таблицей. Если нужно разместить его под таблицей, примените свойство `caption-side` с указанием позиции:

`caption-side`

- **Значения:** `top | bottom`.
- **По умолчанию** — `top`.
- **Применение** — к элементу заглавия таблицы.
- **Наследование** — да.

Свойство `table-layout` позволяет авторам указать один из двух методов расчета ширины таблицы. Значение `fixed`, применяемое при вычислении ширины таблицы, базируется на значениях `width`, предоставленных для таблиц, столбцов или ячеек. Значение `auto` определяет ширину таблицы по минимальной ширине ее контента. Значение `auto` отображается номинально медленнее, поскольку браузер должен для каждой ячейки рассчитать заданную ширину по умолчанию, прежде чем получить значение ширины таблицы.

СВОЙСТВА ОТОБРАЖЕНИЯ ТАБЛИЦЫ

CSS 2.1 включает в себя ряд значений для свойства `display`, которые позволяют разработчикам прикреплять способ отображения таблицы к элементам. Относящимися к таблицам значениями свойства `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`.

Первоначально эти значения служили для предоставления механизма, обеспечивающего отображение таблиц с помощью XML-языков, которые могут не включать в свои словари такие элементы, как `table`, `tr` или `td`.

В наши дни значения отображения таблицы служат еще одним методом достижения таких эффектов для макета страницы, как вертикальное центрирование и применение гибкого значения ширины столбцов. Макет таблицы CSS может представлять значение в качестве запасного варианта для устаревших браузеров, которые не поддерживают CSS Grid или Flexbox. Обратите внимание, что это не идентично использованию табличного макета с HTML-разметкой. При использовании CSS-макета таблицы семантика исходного документа остается неизменной. Если вы хотите узнать об этом больше, я рекомендую статью Массимо Кассандро (Massimo Cassandro) «Layout Secret #1: The CSS Table Property» (www.sitepoint.com/solving-layout-problems-css-table-property/).

Тем не менее я полагаю, что теперь, когда широко распространены Flexbox и Grid, CSS-методы размещения таблиц утрачивают свою популярность.

Итак, мы рассмотрели основы стайлинга форм и форматирования таблиц. И, хотя эта книга ориентирована на начинающих, в следующем разделе я познакомлю вас с некоторыми вспомогательными методиками CSS, которые смогут облегчить вашу работу и сделают ваши страницы быстрее.

ОЧИСТКА СТИЛЕЙ: СБРОС CSS И ТАБЛИЦА СТИЛЕЙ NORMALIZE.CSS

Как известно, для отображения HTML-элементов браузеры располагают собственными встроеннымами таблицами стилей (их называют таблицами стилей агента пользователя). И даже если вы не зададите для элемента `h1` никаких собственных стилей, вы можете быть уверенными, что он станет отображаться в виде крупного и «жирного» текста с отступами сверху и снизу. Но насколько крупным окажется этот шрифт и какого размера отступы его окружат, может варьироваться от браузера к браузеру, что приводит к непредсказуемым результатам. Кроме того, даже если предоставить документу собственную таблицу стилей, элементы в нем могут неявно наследовать определенные стили из таблиц стилей агента пользователя, еще более усиливая непредсказуемость итогового отображения.

К счастью, имеются две методики, позволяющие получить согласованную отправную точку для применения ваших собственных стилей: средство CSS reset (сброс CSS) и таблица стилей **Normalize.css**. Эти методики основаны на разных подходах, поэтому каждая из них может служить наилучшим решением применительно к каждой конкретной ситуации.

Сброс CSS

Сброс CSS (CSS reset) — это более устаревший подход, представляющий собой набор правил стилей, переопределяющих *все* стили пользовательских агентов и формирующих начальный, максимально нейтральный, вариант оформления. Применяя этот метод, вам нужно указать все свойства шрифта и отступов для каждого элемента, который вы используете. Это действительно дает «нулевой» исходный вариант .

Самый популярный код для CSS reset создан Эриком Мейером (Eric Meyer), автором большого числа книг по CSS. Код Мейера представлен здесь далее, и я также включила его копию в папку материалов для этой главы, чтобы вам было удобнее его скопировать и вставить в нужное место.

Просмотрев этот код, можно заметить, что поля, границы и отступы для целого списка элементов блока установлены в 0. В нем также содержатся стили, которые приводят типографику к нейтральному исходному варианту, очищают стили в списках и не позволяют браузерам добавлять символы кавычек в цитаты:

```
/* http://meyerweb.com/eric/tools/css/reset/
v2.0 | 20110126 License: none (public domain)*/
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,
```

ЗАГРУЗКА СРЕДСТВА CSS RESET

Средство CSS reset можно также загрузить с сайта по адресу: meyerweb.com/eric/tools/css/reset/.

```
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;
}

/* HTML5 display-role reset for older browsers */
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;
}
```

Для применения средства CSS reset поместите блок его стилей в верхнюю часть собственной таблицы стилей — чтобы ваши собственные стили их затем переопределяли. Можно использовать это средство точно в том же виде, как оно представлено здесь, или настроить его в соответствии с требованиями своего проекта. Я также рекомендую прочитать посты Эрика о размышленииах, которые навели его на создание CSS reset. Эти посты доступны по адресам: meyerweb.com/eric/tools/css/reset/ и meyerweb.com/eric/thoughts/2007/04/18/reset-reasoning. Впрочем, вы можете провести собственный веб- поиск, который покажет вам и другие, возможно, более легкие, варианты сброса CSS.

Таблица стилей Normalize.css

Более утонченный подход заключается в применении таблицы стилей **Normalize.css**, созданной Николасом Галлахером (Nicolas Gallagher) и Джонатаном Нилом (Jonathan

Neal). Они тщательно изучили стили пользовательских агентов в каждом современном браузере (как настольном, так и мобильном) и создали таблицу стилей, которая согласовывает их стили, а не просто отключает их. Таблица стилей **Normalize.css** предоставляет вам разумный исходный вариант оформления: абзацы по-прежнему имеют некоторое пространство над и под ними, заголовки выделены полужирным шрифтом по убыванию их уровней, списки снабжены логически обоснованными маркерами и отступами. В нее также включены стили, согласующие применяемые формы, что повышает удобство ее использования. На рис. 19.6 показаны различия между документами, оформленными с помощью **CSS reset** и **Normalize.css**.

CSS reset

```
List Item
List Item
List Item
List Item
List Item
List Item
Big letters
Less big or letters
Giga letters
Hello world
Kid Rock
Am man
Lorem ipsum dolor sit amet.
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Fugit dignissimos iusto maxime, quibusdam ut harum consequatur sunt vero!
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam iure, soluta quibusdam ullam error qui atque 7, expedita quod dicta vel laudantium tenetur, quos, voluptatibus aspernatur enim magni velit. Distinctio, adipisci.
```

Singles in your area
Don't click fancy
loud
loud?
copyright forever

Normalize.css

```
• List Item

Big letters
Less big of letters
Getting smaller
Hello world
Kid Rock
Am man
Lorem ipsum dolor sit amet.
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Fugit dignissimos iusto maxime, quibusdam ut harum consequatur sunt vero!
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam iure, soluta quibusdam ullam error qui atque 7, expedita quod dicta vel laudantium tenetur, quos, voluptatibus aspernatur enim magni velit. Distinctio, adipisci.
```

Singles in your area
Don't click fancy
loud
loud?
copyright forever

Рис. 19.6. Различия между документами, оформленными с помощью **CSS reset** (слева) и **Normalize.css** (справа). Экранный снимок Codepen создан Заком Вольфом (Zach Wolf)

Файл с таблицей стилей **Normalize.css** можно загрузить с сайта necolas.github.io/normalize.css/ и вставить его содержимое перед вашими собственными стилями. Этот код весьма длинный, но он хорошо организован и содержит комментарии с четкими пояснениями в каждом разделе. Для того чтобы узнать мнение Николаса об этом проекте, посетите сайт nicolasgallagher.com/about-normalize-css/.

Таблица стилей **Normalize.css** считается лучшим вариантом, чем более «грубый» CSS reset, но я полагаю, что вам следует знать особенности обеих этих методик. Впрочем, если вас устраивает наличие небольших различий в представлении вашей веб-страницы от браузера к браузеру (что характерно и для многих профессиональных разработчиков), этими средствами можно вовсе не пользоваться.

МЕТОДИКА ПОДМЕНЫ ТЕКСТА ИЗОБРАЖЕНИЕМ

Прежде чем веб-шрифты завоевали свое место под солнцем, нам приходилось использовать изображение всякий раз, когда мы хотели оформить текст более привлекательным шрифтом, чем Times или Helvetica. К счастью, сейчас это уже не так,

но время от времени вам может потребоваться заменить текстовый элемент изображением, причем так, чтобы он оставался доступен для средств чтения с экрана. Один из распространенных сценариев — использование в заголовке веб-страницы стилизованного логотипа вместо названия компании.

ПОДМЕНА ТЕКСТА ИЗОБРАЖЕНИЕМ ИЛИ АЛЬТЕРНАТИВНЫЙ ТЕКСТ?

Прежде чем приступить к работе с методикой подмены изображения, подумайте, не достаточно ли вам будет наличия в элементе `img` альтернативного текста `alt`. Если рассматривать именно логотип, то, даже если изображение логотипа по каким-либо причинам не будет отображено, альтернативный текст `alt` все равно представит посетителям сайта название компании. Вообще говоря, пример логотипа, представленный в этом разделе, скорее всего, именно таким образом и должен оформляться. Тем не менее бывают случаи, когда необходимо подменить изображением фактическую текстовую строку, и в этом случае методика подмены изображения окажется весьма полезным компонентом вашего набора инструментов CSS.

ГАЛЕРЕЯ СТАРЫХ МЕТОДИК

Вы можете просмотреть галерею старых методик в музее «The Image Replacement Museum», созданном Мари Мосли (Marie Mosley) (css-tricks.com/the-image-replacement-museum/).

В примере, показанном на рис. ЦВ-19.7, методика Phark применяется для отображения логотипа Jenware вместо текста «Jenware» в элементе `h1` HTML-кода. Вот эта несложная разметка:

```
<h1 id="logo">Jenware</h1>
```

И следующее правило стиля:

```
#logo {  
    width: 450px;  
    height: 80px;  
    background: url(jenware.png) no-repeat;  
    text-indent: -9999px;  
}
```

Здесь следует отметить несколько важных моментов. Во-первых, элемент `h1` по умолчанию отображается как блок, поэтому вы можете просто указать его значения `width` и `height` в соответствии с размерами изображения, используемого

Тем не менее полное удаление текста и замена его элементом `img` — идея неудачная, поскольку текстовый контент исчезает навсегда. Правильное решение состоит в применении методики подмены текста изображением на основе CSS, которая использует изображение в элементе в качестве фона, а затем смещает текст в сторону так, чтобы он не отображался на странице. При этом визуальные браузеры видят фоновое изображение, а текстовый контент остается в файле документа доступным для поисковых систем, программ чтения с экрана и других вспомогательных средств. В результате все остаются в выигрыше!

За истекшие годы было разработано множество методик подмены текста изображением, но наиболее популярной является методика Phark, созданная Майком Рандлом (Mike Rundle). Она основана на применении значительного отрицательного отступа, смещающего текст влево от видимой страницы.

в качестве фона. Свойство `text-indent` сдвигает слово «Jenware» влево на 9999 пикселов. Поэтому браузер должен иметь возможность отображать очень широкий блок элементов, но при минимальном снижении производительности.

Недостатком любого варианта подмены текста изображением является необходимость формирования для каждого используемого изображения дополнительного запроса к серверу. Кроме того, замена заголовка изображением требует дополнительной работы по созданию самого изображения. Так что, еще раз — прежде чем приступить к замене текста изображением, подумайте, сможет ли веб-шрифт или встроенное изображение с альтернативным текстом `alt` справиться с задачей. А в следующем разделе мы рассмотрим способ избавиться от ненужных запросов к серверу.

CSS-СПРАЙТЫ

Когда мы говорили о производительности (см. главу 3), я отметила, что вы можете улучшить производительность сайта, уменьшив количество запросов к серверу, которые делает ваша страница (например, HTTP-запросов). Одна из стратегий уменьшения количества запросов изображений состоит в объединении ваших небольших изображений в один большой файл изображения, чтобы запрашивался именно он и только один раз. Большое изображение, содержащее несколько изображений, известно как спрайт. Этот термин придуман на ранних этапах развития индустрии компьютерной графики и видеоигр. Спрайт позиционируется в элементе с помощью свойства `background-position` так, чтобы была видна только нужная в каждый момент его часть.

Так, если мне нужно отобразить на странице коллекцию из шести pictogramm социальных сетей, можно собрать эти шесть графических изображений в один спрайт, чем соответственно уменьшить количество HTTP-запросов (рис. ЦВ-19.8). Как можно видеть, pictogramмы здесь собраны в один вертикальный рисунок (файл `social.png`). В этом примере также применяется метод замены текста изображением, поэтому текст для каждой ссылки по-прежнему доступен для средств чтения с экрана.

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 dropbox">Dropbox</a></li>
  <li><a href="" class="hide pinterest">Pinterest</a></li>
</ul>
```

Правила стиля

```
.hide {
  text-indent: 100%;
  white-space: nowrap;
  overflow: hidden;
```

```
}

li a {
    display: block;
    width: 40px;
    height: 40px;
    background-image: url(social.png);
}
li a.twitter { background-position: 0 0; }
li a.fb { background-position: 0 -40px; }
li a.gplus { background-position: 0 -80px; }
li a.linkedin { background-position: 0 -120px; }
li a.dropbox { background-position: 0 -160px; }
li a.pinterest { background-position: 0 -200px; }
```

В приведенном макете у каждого элемента имеются два значения класса `class`. Класс `hide` используется в качестве селектора для применения методики замены текста изображением. Эта методика разработана Скоттом Келлумом (Scott Kellum) и задействует левый отступ в 100%, чтобы устраниТЬ текст из поля зрения. Другое значение класса `class` является специфическим для каждой ссылки в социальной сети. Уникальные значения `class` позволяют позиционировать спрайт соответствующим образом для каждой ссылки.

В верхней части таблицы стилей показаны стили замены текста изображениями. Обратите внимание на правило, согласно которому все элементы связей (`a`) используют в качестве фонового изображения файл `social.png`.

Наконец, рассмотрим стили, которые выполняют самую ответственную работу. Свойство `background-position` по-разному задается для каждой ссылки в списке, а поле видимого элемента функционирует как небольшое окно, раскрывающее часть фонового изображения. Первый элемент имеет значение `0, 0` — тогда верхний левый угол изображения помещается в верхний левый угол поля элемента. Для того чтобы

пиктограмма Facebook стала видимой, нужно переместить изображение на 40 пикселов вверх — при этом его вертикальное положение устанавливается равным `-40px` (положение по горизонтали равно 0). Следующие изображения перемещаются с шагом в 40 пикселов для каждой ссылки, открывая области изображения вниз по стеку спайта все дальше и дальше.

В приведенном примере все пиктограммы имеют одинаковые размеры и красиво складываются, но это не является обязательным требованием. На одном спрайте можно комбинировать изображения с различными размерами.

ГЕНЕРАТОРЫ СПРАЙТОВ

Существует большое число интернет-инструментов, которые создают файлы изображений спрайтов и соответствующие им стили автоматически. Просто загрузите или перетащите свои отдельные изображения на страницу инструмента, и он выполнит все остальное. Я нахожу весьма простым в использовании сайт CSSsprites (css.spritegen.com). Если вы хотите, чтобы спрайты получались адаптивными (отзывчивыми), используйте адаптивную версию сайта по адресу: responsive-css.spritegen.com.

Процесс установки размера для элемента, а затем — выравнивания спрайта с помощью свойства `background-position` проходит аналогичным образом.

ПРОВЕРКА ПОДДЕРЖКИ СВОЙСТВ CSS

Одна из основных проблем, с которой сталкиваются веб-дизайнеры и разработчики, связана с неравномерной браузерной поддержкой свойств CSS. Новые полезные свойства CSS появляются регулярно, но им требуется время, чтобы найти свой путь к браузерам, и гораздо больше времени, чтобы старые, не поддерживающие новые свойства CSS браузеры вышли из употребления.

К счастью, существует несколько методик, позволяющих проверить, поддерживает ли браузер ту или иную функцию, что дает нам возможность воспользоваться преимуществами продвинутых CSS-свойств, а также предоставить продуманные альтернативы для браузеров, их не поддерживающих. Использование функции проверки поддержки вкупе с запасными вариантами позволяет обойти доступные неприятные альтернативы: а) не использовать свойство, пока оно не станет универсально поддерживаемым, или б) использовать свойство, что приведет к появлению определенных проблем у пользователей браузеров, которые эти свойства не поддерживают.

Рассмотрим два способа, позволяющих определить, поддерживается ли функция: запросы функций с помощью нового правила CSS (`@supports`) и основанного на JavaScript инструмента под названием Modernizr.

Правило `@supports` (CSS-запросы функций)

Модуль CSS3 Conditional Rules уровня 3 (www.w3.org/TR/css3-conditional/) включает правило `@supports`, обеспечивающее проверку поддержки браузером определенного свойства и объявления значения. Обычно называемый *запросом функции*, он работает как медиапрофиль в том смысле, что запускает тест, и, если браузер выполняет этот тест, задействуются стили, содержащиеся в скобках правила. Для правила `@supports` применяется следующий синтаксис:

```
@supports (свойство: значение) {  
    /* Правила стилей для поддерживающих браузеров */  
}
```

Обратите внимание, что запрос относится ко всему объявлению: как к свойству, так и к значению. Это связано с тем, что иногда можно проверить новое свойство (например, `initial-letter`), а иногда может понадобиться проверить новое значение для существующего свойства.

Например, свойство `display` поддерживается универсально, но новое значение (ключевое слово) `grid` поддерживается не всегда. Учтите также, что в конце запроса отсутствует точка с запятой.

ОТНОСИТЕСЬ К ПОДДЕРЖКЕ ОСТОРОЖНО...

Браузер должен сам разобраться, реализована ли у него та или иная функция. Но, если функция реализована с ошибками, вы все равно можете столкнуться с проблемами даже при использовании запросов функций.

Давайте рассмотрим более конкретный пример (рис. ЦВ-19.9). Я полагаю, что было бы здорово использовать новое свойство `mix-blend-mode`, чтобы фотография арбузов больше гармонировала с фоном (аналогично режиму наложения слоев в Photoshop). На момент подготовки книги это свойство поддерживается только в Firefox, Chrome и Safari. В качестве запасного варианта для не поддерживающих его браузеров я создам менее интересный смешанный эффект, используя свойство `opacity`.

Сейчас, когда я работаю над этой книгой, считается правильным сначала задать стили для запасного варианта, а затем переопределить их набором стилей, предназначенных для браузеров, поддерживающих новую функцию. Обратите внимание, что свойству `opacity` также нужно будет снова присвоить значение 1, чтобы переопределить стиль запасного варианта:

Разметка

```
<div id="container">
    <figure class="blend">
        
    </figure>
</div>
```

Правила стиля

```
#container {
    background-color: #96D4E7;
    padding: 5em;
}
.blend img {
    opacity: .5;
}
@supports (mix-blend-mode: multiply) {
    .blend img {
        mix-blend-mode: multiply;
        opacity: 1;
    }
}
```

Операторы

Правило `@supports` может применяться с тремя операторами, уточняющими возможности проверки: `not`, `and` и `or`.

Оператор `not`

Оператор `not` позволяет убедиться, что определенная пара свойство/значение *не* поддерживается:

```
@supports not (mix-blend-mode: hue) {
    /* Правила стилей для не поддерживающих браузеров */
}
```

Когда-нибудь это будет полезно для предоставления резервных стилей, но с текущей поддержкой браузеров вы рискуете с таким правилом `@supports` потерять для

не поддерживающих браузеров все стили, включая резервные. Вот почему я использовала в предыдущем примере метод переопределения.

Оператор *and*

Применяйте стили только в том случае, если выполняются все условия в серии, состоящей не менее чем из двух условий:

```
@supports (border-radius: 10em) and (shape-outside: circle()) {  
    /* стили, применяемые в том случае, когда их поддерживает браузер  
    shape-outside AND border-radius */  
}
```

Оператор *or*

Используйте оператор *or* для применения стилей, если выполнена любая серия условий. Это особенно полезно для свойств с префиксом поставщика:

```
@supports (-webkit-transform: rotate(10deg)) or  
(-ms-transform: rotate(10deg)) or  
(transform: rotate(10deg))  
/* transform styles */  
}
```

Браузерная поддержка

Запросы свойств начали поддерживаться в Chrome, Firefox и Opera еще в 2013-м году, и они также поддерживаются всеми версиями Microsoft Edge. Браузер Safari начал поддерживать их с 9-й версии, появившейся в 2015-м году. К сожалению, ни одна из версий Internet Explorer не поддерживает запросы свойств, что представляет серьезную проблему до тех пор, пока эти устаревшие браузеры не перестанут использоваться.

Браузеры, которые не поддерживают запросы свойств, применяют резервный вариант дизайна, поэтому убедитесь, что он, как минимум, пригоден для использования. Остерегайтесь, однако, браузеров, которые не поддерживают свойство `@supports`, но поддерживают новые свойства CSS. Отличный пример — это Flexbox. Браузер Safari 8 распознает свойства Flexbox, но не поддерживает запросы `@supports`, так что если все правила размещения Flexbox скрыты в запросе свойств, Safari 8 их не найдет. Поэтому запросы свойств не являются наилучшим инструментом для обнаружения Flexbox или любого свойства, которое имеет большую поддержку, чем `@supports`. Модуль Grid Layout, с другой стороны, полностью поддерживает запросы свойств, поскольку каждый браузер, поддерживающий свойство `display: grid`, также поддерживает `@supports`. Остается только еще раз напомнить вам, что сайт [CanIUse.com](#) является хорошим местом для уточнения браузерной поддержки.

НЕ ВСЕ ВОЗМОЖНОСТИ НУЖДАЮТСЯ В ЗАПРОСЕ СВОЙСТВ

Не каждая новая возможность нуждается в запросе свойства. Некоторые возможности — такие, как `border-radius`, просто не отображаются в неподдерживающих браузерах, и это прекрасно.

Преимущества и недостатки

Запросы свойств — это чудесный новый инструмент для веб-разработки. Они позволяют нам быстро воспользоваться преимуществами новых свойств CSS, чтобы не полагаться на JavaScript (далее мы рассмотрим библиотеку Modernizr — решение на JavaScript), поскольку загрузка и запуск скрипта (даже небольшого) выполняются медленнее, чем использование только CSS.

С другой стороны, ограниченная пока поддержка запросов свойств со стороны браузеров означает, что свойство `@supports` еще не столь всемогуще, как Modernizr. Тем не менее, если оно позволяет вам решить поставленные задачи, рекомендую вам выбирать в первую очередь его. К счастью, браузерная поддержка со временем будет только улучшаться, предоставляя в будущем функциям CSS преимущество перед решением на основе сценариев.

Однако что же это такое — Modernizr, о которой я только что упомянула?

Библиотека Modernizr

Modernizr — это облегченная библиотека JavaScript, которая функционирует «за кулисами» и при загрузке страницы в браузер проверяет множество содержащихся в ней свойств HTML5 и CSS3. Для каждой проверяемой функции в объекте JavaScript сохраняется результат (поддерживает/не поддерживает). Доступ к этому объекту можно получить с помощью сценариев (скриптов), а также — опционально — с помощью имени класса в корневом элементе `html`, который можно использовать в CSS-селекторах. На этом CSS-методе я здесь и сосредоточусь.

Как это работает?

Выполняясь, библиотека Modernizr добавляет элемент `html` с именем класса для каждой проверяемой возможности. К примеру, если проверка ориентирована на

БИБЛИОТЕКА MODERNIZR

Modernizr — это облегченная библиотека JavaScript, которая проверяет различные свойства HTML и CSS.

Flexbox, то при выполнении в браузере, поддерживающем Flexbox, она добавляет имя класса `.flexbox` к элементу `html`:

```
<html class="js flexbox">
```

Если возможность не поддерживается, к названию возможности добавляется имя с префиксом `.no-`. Тогда для неподдерживающего браузера проверка Flexbox будет отображаться так:

```
<html class="js no-flexbox">
```

При наличии имени класса в корневом элементе все элементы на странице становятся частью этого класса. Мы можем применять имя класса как часть селектора для предоставления различных наборов стилей в зависимости от поддержки возможности:

```
.flexbox nav {  
    /* стили flexbox для элемента nav */
```

```
}

.no-flexbox nav {
    /* резервные стили для элемента nav */
}
```

На этом коротком примере удобно продемонстрировать принцип работы Modernizr. Однако, как правило, Modernizr используется для проверки многих возможностей, и элемент `html` заполняется при этом длинным списком имен классов.

Как это применять?

Прежде всего, следует загрузить сценарий **Modernizr.js**. Для этого зайдите на сайт **Modernizr.com** и перейдите по ссылке **Download** (Загрузить). Здесь вы сможете настроить этот сценарий таким образом, чтобы он содержал только те свойства HTML и CSS, которые необходимо проверить, что также позволит сократить размер файла сценария. Щелкните на кнопке **Build** (Сформировать), после чего вам будет предложено несколько вариантов сохранения сценария. Простой щелчок на кнопке **Download** сохраняет на вашем компьютере этот сценарий в файле с расширением `js`.

Обзаведясь своим сценарием, поместите его файл в один каталог с остальными файлами вашего проекта. Добавьте также имя файла сценария в заголовок HTML-документа перед любыми связанными таблицами стилей или другими сценариями, которые его применяют:

```
<head>
    <script src="modernizr-custom.js"></script>
    <!--other scripts and style sheets -->
</head>
```

Наконец, откройте ваш HTML-документ и присвойте имя класса `no-js` элементу `html`:

```
<html class="no-js">
```

Modernizr сразу изменит имя класса на `js`, если убедится, что браузер поддерживает JavaScript. Если же JavaScript (и, соответственно, Modernizr) не запустится, вы не сможете узнать, поддерживаются ли запрошенные возможности.

Преимущества и недостатки

Библиотека Modernizr является одним из самых популярных инструментов в арсенале веб-разработчиков, так как позволяет нам учитывать определенные свойства, а не возможности целых браузеров. Она проста в использовании, и на сайте Modernizr имеется подробная и понятная документация, которая вам в этом поможет. Поскольку это JavaScript, то Modernizr поддерживается подавляющим большинством браузеров. Обратная сторона этой медали заключается в том, что из-за ее опоры на JavaScript нельзя на 100% быть уверенными, что она запустится, что и является ее основным недостатком. Кроме того, выполнение Modernizr осуществляется немного медленнее, чем использование для проверки поддержки свойств только возможностей CSS.

ТАБЛИЦЫ СТИЛЕЙ: ПОСЛЕСЛОВИЕ

На этом завершается наш марш-бросок по каскадным таблицам стилей. Мы прошли длинный путь, начавшийся в главе 11 присвоением стилей элементам `h1` и `p`. Сейчас вы уже можете спокойно выполнять форматирование текста и даже создавать основные макеты страниц. Правила CSS выучить нетрудно, однако освоение их возможностей требует много времени и практики. Если вы начнете испытывать затруднения, к вашим услугам множество ресурсов Интернета, которые помогут вам найти нужные ответы. Весьма удобно, что, работая с CSS, вы можете начать с самых основ, а затем, приобретая навыки веб-разработки, повышать свое мастерство.

В следующей главе я познакомлю вас с инструментами, которые веб-разработчики используют для лучшей организации своего рабочего процесса, инструментами для более эффективного создания правил CSS и оптимизации результатов. Но, если вы сейчас чувствуете, что сыты свойствами CSS по горло, можете вздохнуть с облегчением — вы завершили изучение этого материала!

КОНТРОЛЬНЫЕ ВОПРОСЫ

Ответьте на следующие вопросы, и вы узнаете, насколько хорошо усвоили CSS-методики, описанные в этой главе. Как вы уже догадались, ответы доступны в *приложении 1*.

1. Какова цель сброса CSS (CSS reset)?
 - a. Переопределить настройки браузера по умолчанию.
 - b. Сделать презентацию более предсказуемой в разных браузерах.
 - c. Чтобы элементы не наследовали неожиданные стили.
 - d. Все возможности из здесь перечисленных.
2. Какова цель CSS-спрайта?
 - a. Улучшение производительности сайта.
 - b. Применять небольшие изображения вместо больших, уменьшая размер файла.
 - c. Сократить число HTTP-запросов.
 - d. а и с.
 - e. Все возможности из здесь перечисленных.
3. Какова цель методики подмены текста изображением?
 - a. Создание действительно больших текстовых отступов.
 - b. Использование декоративной графики вместо текста.
 - c. Удаление текста из документа и замена его декоративным изображением.

- d. Поддержка семантического контента документа.
- e. b и d.
- f. Все возможности из здесь перечисленных.
4. Назовите два подхода к выравниванию элементов управления формы и соответствующих надписей без таблиц. Дайте общее описание.
5. Сопоставьте эти правила стиля с соответствующими таблицами на рис. 19.10.
- `table { border-collapse: collapse; }
td { border: 2px black solid; }`
 - `table { border-collapse: separate; }
td { border: 2px black solid; }`
 - `table {
 border-collapse: separate;
 border-spacing: 2px 12px; }
td { border: 2px black solid; }`
 - `table {
 border-collapse: separate;
 border-spacing: 5px;
 border: 2px black solid; }
td { background-color: #99f; }`
 - `table {
 border-collapse: separate;
 border-spacing: 5px; }
td { background-color: #99f;
border: 2px black solid; }`

①	<table border="1"> <tbody> <tr><td>Cell A</td><td>Cell B</td><td>Cell C</td></tr> <tr><td>Cell D</td><td>Cell E</td><td>Cell F</td></tr> </tbody> </table>	Cell A	Cell B	Cell C	Cell D	Cell E	Cell F
Cell A	Cell B	Cell C					
Cell D	Cell E	Cell F					
②	<table border="1"> <tbody> <tr><td>Cell A</td><td>Cell B</td><td>Cell C</td></tr> <tr><td>Cell D</td><td>Cell E</td><td>Cell F</td></tr> </tbody> </table>	Cell A	Cell B	Cell C	Cell D	Cell E	Cell F
Cell A	Cell B	Cell C					
Cell D	Cell E	Cell F					
③	<table border="1"> <tbody> <tr><td>Cell A</td><td>Cell B</td><td>Cell C</td></tr> <tr><td>Cell D</td><td>Cell E</td><td>Cell F</td></tr> </tbody> </table>	Cell A	Cell B	Cell C	Cell D	Cell E	Cell F
Cell A	Cell B	Cell C					
Cell D	Cell E	Cell F					
④	<table border="1"> <tbody> <tr><td>Cell A</td><td>Cell B</td><td>Cell C</td></tr> <tr><td>Cell D</td><td>Cell E</td><td>Cell F</td></tr> </tbody> </table>	Cell A	Cell B	Cell C	Cell D	Cell E	Cell F
Cell A	Cell B	Cell C					
Cell D	Cell E	Cell F					
⑤	<table border="1"> <tbody> <tr><td>Cell A</td><td>Cell B</td><td>Cell C</td></tr> <tr><td>Cell D</td><td>Cell E</td><td>Cell F</td></tr> </tbody> </table>	Cell A	Cell B	Cell C	Cell D	Cell E	Cell F
Cell A	Cell B	Cell C					
Cell D	Cell E	Cell F					

Рис. 19.10. Сопоставьте эти таблицы с примерами кода в вопросе 5

6. Применяя Modernizr, выполните тестирование свойства `border-radius` на предмет того, будет ли `div` отображаться со скругленными углами на основе следующих результатов обобщенного класса:

```
.border-radius div {  
    border: 1px solid green;  
    border-radius: .5em;  
}
```

- a. <html class="js .no-border-radius">
 - b. <html class="js .border-radius">
 - c. <html class="no-js">
7. Как следствие написанного, какое преимущество имеет Modernizr по сравнению с запросом CSS-свойства? Какое преимущество по длительности получит запрос CSS-свойства по сравнению с Modernizr?

ОБЗОР CSS: СВОЙСТВА ТАБЛИЦ

В табл. 19.1 в алфавитном порядке приводятся краткие описания рассмотренных в этой главе свойств таблиц.

Таблица 19.1. Свойства таблиц

Свойство	Описание
<i>border-collapse</i>	Указывает, являются ли границы между ячейками раздельными или схлопнутыми
<i>border-spacing</i>	Обозначает пространство между ячейками, установленными для визуализации, как раздельные
<i>caption-side</i>	Определяет положение заголовка таблицы относительно таблицы (вверху или внизу)
<i>empty-cells</i>	Указывает, должны ли границы и фонны отображаться для пустых ячеек
<i>table-layout</i>	Определяет вычисление ширины таблицы

ГЛАВА 20

СОВРЕМЕННЫЕ ИНСТРУМЕНТЫ ДЛЯ ВЕБ-ДИЗАЙНА

В этой главе...

- ▶ *Введение в командную строку*
- ▶ *Препроцессоры и постпроцессоры CSS*
- ▶ *Инструменты для сборки и обработки задач*
- ▶ *Программа управления версиями Git*

В упражнениях этой книги вы создавали статические HTML-страницы со встроенными таблицами стилей, затем сохраняли их и открывали в браузере. И, хотя это вполне действенный подход, вы, скорее всего, пойдете другим путем, когда решите серьезно заняться веб-дизайном. Так что если вы изучаете веб-дизайн и разработку с дальными целями, то должны быть знакомы с тем, как все это делается в профессиональной среде.

Эта глава познакомит вас со следующими инструментами, используемыми веб-разработчиками для облегчения своей работы и повышения надежности кода:

- CSS-процессорами, помогающими более эффективно создавать CSS-код и оптимизировать результирующий код, с тем чтобы он работал во всех браузерах;
- инструментами, автоматизирующими применение повторяющихся видов задач, с которыми вы сталкиваетесь при создании кода;
- программой управления версиями Git, отслеживающей ваши предыдущие версии кода и облегчающей группам веб-дизайнеров совместную работу над одним и тем же кодом.

Все эти инструменты объединяет то, что они обычно применяются совместно с *интерфейсом командной строки* (ИКС). Так что, прежде чем рассмотреть конкретные инструменты веб-дизайна, освоимся сначала с командной строкой.

ЗНАКОМСТВО С КОМАНДНОЙ СТРОКОЙ

Вероятно, вы сидите за компьютером с графическим пользовательским интерфейсом (GUI), предлагающим вам пиктограммы, обозначающие файлы и папки, полные

опций раскрывающиеся меню, а также интуитивно понятные действия, такие, как перетаскивание файлов из папки в папку или в корзину.

Пользователи компьютеров в 60-е и 70-е годы прошлого века не располагали такой роскошью. В те времена единственным способом заставить компьютер выполнить задачу — было ввести команду. Несмотря на модные графические интерфейсы, ввод команд с помощью терминала командной строки еще достаточно широко применяется. На самом деле, чем опытнее вы становитесь в веб-дизайне, тем больше будете склонны к использованию командной строки для выполнения определенных задач. Если же вы еще и являетесь программистом, то с работой в режиме командной строки наверняка знакомы.

Командная строка по-прежнему популярна в силу ряда причин. Прежде всего, она используется для получения доступа к удаленным компьютерам, а разработчикам часто необходим доступ к файлам и управление ими на удаленных веб-серверах. Кроме того, проще написать программу для командной строки, чем отдельное приложение с графическим интерфейсом, поэтому многие из удобных инструментов для оптимизации рабочего процесса существуют именно в виде программ для командной строки. При этом многие из таких инструментов могут использоваться совместно в конвейере для выполнения сложных задач.

Основанные на здравом смысле преимущества командной строки, экономящие время и усилия, являются мощным стимулом к ее использованию. Проверьте мне: если вы сможете изучить все элементы и свойства стилей, то сможете привыкнуть и к вводу нескольких команд.

Приложение командной строки Terminal

Программа, интерпретирующая вводимые вами команды, называется оболочкой (*shell*) — все визуальные интерфейсы технически также являются оболочками, просто они более изящны. Каждый компьютер, работающий под управлением macOS и Linux, поставляется вместе с приложением Terminal (Терминал), которое использует программу оболочки *bash*. Чтобы получить доступ к приложению Terminal в macOS (рис. 20.1), выполните команду **Applications | Utilities** (Приложения | Утилиты).

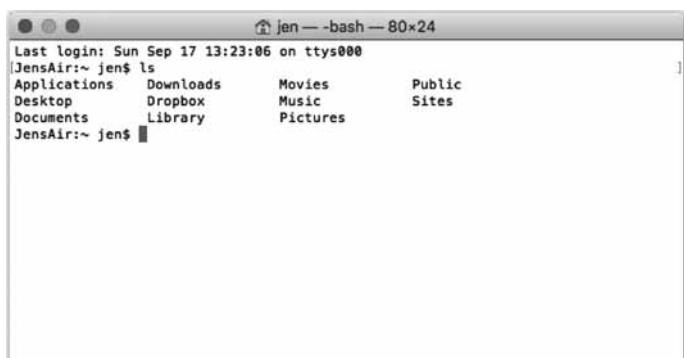


Рис. 20.1. Окно приложения Terminal в macOS

Пользователи Windows располагают несколькими возможностями для работы с командной строкой. Средством командной строки в Windows по умолчанию является приложение Command Prompt (Командная строка), доступ к которому легко получить с помощью поля поиска (**Search**). Это средство может выполнить большое число функций, необходимых вам, как разработчику, но оно не использует bash. Поскольку многие инструменты веб-дизайна основаны на использовании bash, лучше установить эмулятор оболочки на основе bash, например, Cygwin (cygwin.com) или cmder (cmder.net). Если вы работаете в Windows 10, рекомендуется установить на свой компьютер с помощью подсистемы Windows для Linux (msdn.microsoft.com/en-us/commandline/wsl/about) среду Linux — например, Ubuntu, доступную в магазине Windows (www.microsoft.com/en-us/store/p/ubuntu/9nblggh4msv6).

Выдаем команды

Открыв окно приложения Terminal, первое, что вы видите, это *приглашение* командной строки, представляющее собой строку символов, которая указывает, что компьютер готов принять вашу команду:

```
$: _
```

В строке приглашения обычно отображается знак доллара, но в вашей терминальной программе может отображаться и другой символ. Подчеркивание в примере обозначает позицию курсора, которая может выглядеть как маленький прямоугольник или мигающая линия.

Полный запрос, отображаемый в командной строке Terminal, на моем компьютере начинается с имени компьютера (*JensAir*) и указания на рабочий каталог, а именно — на каталог, который в текущий момент просматривается оболочкой. В терминах графического интерфейса пользователя рабочим каталогом является папка, в которой «вы находитесь». В следующем примере тильда (~) указывает, что я нахожусь в своем корневом каталоге пользователя.

Префикс *jep* (Джен) перед символом подсказки отображает мое имя как пользователя командной строки (в последующих примерах приглашение сокращено до \$:):

```
JensAir:~ jen$: _
```

Если вы видите приглашение командной строки, введите команду и нажмите клавишу <Enter> — компьютер выполнит команду и после ее завершения выдаст новое приглашение. Впрочем, для некоторых команд до появления следующего приглашения может также отображаться ответная реакция или еще какая-либо информация. Иногда же все происходит «за кулисами», и все, что вы увидите, это новое приглашение.

НАСТРОЙКА ВНЕШНЕГО ВИДА ПРИЛОЖЕНИЯ TERMINAL

Внешний вид приложения Terminal можно подстроить под себя, выполнив команду **Preferences | Profile** (Настройки | Профиль) и изменяя настройки. Если вы желаете развлечься, можете заменить символ подсказки с \$ на символ по вашему выбору, включая эмодзи (osxdaily.com/2013/04/08/add-emojis-command-line-bash-prompt/).

При изучении командной строки обычно начинают со встроенных команд для навигации по файловой системе — задач, обычно выполняемых с помощью средства Finder на компьютере macOS и приложения Мой компьютер (My Computer) в Windows. Поскольку эти методы интуитивно понятны, именно с них я и собираюсь начать простой урок, посвященный командной строке.

Хорошей небольшой утилитой для начинающих является `pwd` (сокращение от слов Print Working Directory, печать рабочего каталога), которая отображает полный путь к рабочему (текущему) каталогу. Для применения этой команды просто введите `pwd` в строке приглашения. Не бойтесь, эта команда ничему не повредит, а может просто подсказать, где именно вы находитесь. В следующем примере косая черта (слэш)

КОРНЕВОЙ КАТАЛОГ ПОЛЬЗОВАТЕЛЯ

Пользовательский каталог в Terminal является корневым каталогом по умолчанию и представлен в приглашении тильдой (~), как показано в ранее приведенном примере.

означает, что путь к моему каталогу начинается с корневого каталога всего компьютера:

```
$: pwd  
/Users/jen
```

Рассмотрим еще один простой (и безопасный) пример. После ввода в командной строке команды `ls` возвращается список (list) файлов и каталогов, вложенных в рабочий каталог (`/Users/jen`):

```
JensAir:~ jen$ ls  
Applications      Downloads      Movies        Public  
Desktop          Dropbox       Music         Sites  
Documents        Library      Pictures  
JensAir:~ jen$
```

Можно сравнить этот список с представлением Finder той же папки, показанном на рис. 20.2. Это два способа взглянуть на одну и ту же сущность: каталог и папку — два термина, обозначающих один и тот же объект в зависимости от того, каким способом вы к нему обращаетесь.



Рис. 20.2. Представление Finder для домашней папки `jen`

Некоторые утилиты, такие, как `pwd`, требуют для выполнения указания только своего имени, но чаще бывает нужно предоставить утилите дополнительную информацию в виде флагов и аргументов. *Флаг* изменяет работу утилиты, задавая ей, например, какой-либо дополнительный параметр или настройку. Он следует за именем команды и обозначается одинарной (-) или двойной (--) чертой. Во многих случаях флаги сокращаются до первой буквы их имени, поскольку применяются в контексте с определенной утилитой. Например, можно изменить представление утилиты `ls` с флагом `-l`, который указывает компьютеру на необходимость отображения контента каталога в «длинном» (long) формате, включая настройки разрешений и даты создания:

```
JensAir:~ jen$ ls -l
total 0
drwxr-xr-x 5 jen staff 170 Jul 8 2016 Applications
drwx----- 57 jen staff 1938 Sep 11 09:47 Desktop
drwx----- 26 jen staff 884 May 18 11:34 Documents
drwx-----+ 151 jen staff 5134 Sep 3 15:47 Downloads
drwx-----@ 48 jen staff 1632 Aug 16 16:34 Dropbox
drwx-----@ 72 jen staff 2448 Jul 15 11:21 Library
drwx----- 22 jen staff 748 Oct 6 2016 Movies
drwx----- 12 jen staff 408 Sep 29 2016 Music
drwx----- 14 jen staff 476 Oct 13 2016 Pictures
drwxr-xr-x 6 jen staff 204 May 6 2015 Public
drwxr-xr-x 11 jen staff 374 Jul 10 2016 Sites
JensAir:~ jen$
```

Аргумент предоставляет утилите конкретную информацию, необходимую для конкретной функции. Например, если я хочу перейти в другой каталог, я ввожу `cd` (сокращение от слов «change directory», изменение каталога), а также — имя каталога, куда необходимо перейти.

Чтобы сделать каталог **Dropbox** новым рабочим каталогом, я ввожу следующую команду:

```
JensAir:~ jen$: cd Dropbox
```

По нажатию клавиши `<Enter>` приглашение изменяется на:

```
JensAir:Dropbox jen$
```

показывая, что я нахожусь в каталоге **Dropbox**. Если ввести сейчас команду `ls`,

ДОТ-ФАЙЛЫ

На компьютере есть некоторые файлы, которые в представлении Finder не выводятся. Эти файлы, известные как *дот-файлы*, начинаются (как вы уже догадались) с точки и, как правило, обрабатывают предназначенную для работы «за кулисами» информацию. Если ввести команду `ls -a` (`-a` — сокращение от слова «all»), можно обнаружить скрытые в каталоге дот-файлы. В macOS можно настроить Finder для отображения дот-файлов, но большинству пользователей текущее скрытое состояние гораздо удобнее.

ВОСПОЛЬЗУЙТЕСЬ СРЕДСТВОМ FINDER

На компьютерах с macOS утилита Terminal хорошо связана со средством Finder. Если нужно указать путь к каталогу или файлу, можно перетащить значок этого файла или папки из Finder в окно Terminal, после чего будет автоматически получен требуемый путь.

будет выведен список содержащихся в папке **Dropbox** файлов и папок (определенного слишком длинный, чтобы отображать его здесь).

Для перехода на уровень выше и возвращения в домашний каталог пользователя (~) я могу применить сокращение UNIX для команды «подняться на уровень выше»: .. (помните, оно рассматривалось в уроке, посвященном URL-пути). Полученное в ответ приглашение показывает, что я вернулась в корневой каталог (~):

```
JensAir:Dropbox jen$ cd ..  
JensAir:~ jen$
```

Некоторые другие полезные команды для работы с файлами включают: mv (перемещает файлы и папки), cp (копирует файлы) и mkdir (создает новый пустой каталог). Команда rm удаляет файл или папку в рабочем каталоге. Будьте осторожны при работе с этой командой, поскольку она не просто перемещает файлы в корзину, а полностью удаляет их с компьютера.

НЕ ВВОДИТЕ КОМАНДУ, ЕСЛИ НЕ ЗНАЕТЕ, ЧТО ОНА ДЕЛАЕТ И КАК РАБОТАЕТ

Командная строка позволяет вам проникнуть в критические области вашего компьютера, которые его GUI любезно защищает от вас. Лучше не вводить команду, если вы не знаете точно, что она делает и как она работает. Сделайте полную резервную копию своего компьютера, прежде чем начать играть с командной строкой, чтобы вы могли быть спокойны, что ваши файлы по-прежнему останутся доступными, если что-то пойдет не так.

возврата на страницу вверх следует нажимать клавиши $<Fn>+<\downarrow>$ или $<Shift>+<\downarrow>$, соответственно. Наконец, для выхода из справочной страницы и возврата к строке приглашения введите команду q.

Дополнительные источники

Не будет особым секретом, если сказать, что рассмотренные нами команды служат лишь верхушкой айсберга, если речь идет об утилитах командной строки. Полный список команд, которые можно применять при работе с оболочкой bash, находится на сайте «An A-Z Index of the Bash Command Line for Linux», доступном по адресу: ss64.com/bash/. Вы будете обращаться к этому списку по мере необходимости, так что не перегружайте себя лишней информацией сейчас. Кроме того, при установке и применении новых инструментов, подобных тем, что рассматриваются в этой главе, вы постепенно изучите необходимые команды, а также флаги и аргументы для них. Поэтому не спешите, всему свое время!

ВОЗВРАЩЕНИЕ В ДОМАШНИЙ КАТАЛОГ

Ввод команды cd с последующим пробелом всегда возвращает вас в ваш домашний каталог.

Как бы там ни было, но у меня нет здесь места (и, если честно, опыта), чтобы предложить вам в этой главе всеобъемлющее руководство по командной строке, однако вы найдете в Интернете книги и множество учебных пособий, которые могут вам помочь. Я полагаю, что руководство Майкла Хартла (Michael Hartl) «Learn Enough Command Line to Be Dangerous» (www.learnenough.com/command-line-tutorial#sec-basics) является достаточно полным и доступным, даже если начинать изучение этой темы «с нуля». Также я рекомендую серию пособий от Envato Tuts+ «The Command Line for Web Design» (webdesign.tutsplus.com/series/the-command-line-for-web-design--cms-777). Если вы заинтересованы в видеоуроках, обратитесь к курсу Рэми Шарпа (Remy Sharp) «Command Line for Non-Techies» (terminal.training).

Теперь, когда вы получили общее представление о работе с командной строкой, давайте рассмотрим инструменты, которые вы могли бы использовать в своей работе, начав с инструментов для создания и оптимизации CSS.

УНИВЕРСАЛЬНЫЕ CSS-ИНСТРУМЕНТЫ (ПРОЦЕССОРЫ)

Хотя вы лишь начинаете работать с CSS, было бы непростительно не познакомить вас с некоторыми универсальными CSS-инструментами, которые являются центральными в рабочем процессе профессионального веб-дизайнера. Эти инструменты делятся на две основные категории:

- созданные на основе CSS языки, использующие синтаксические особенности традиционных языков программирования и экономящие за счет этого время разработчиков. Эти языки принято называть *препроцессорами*. На момент подготовки книги самыми популярными препроцессорами являются Sass, LESS и Stylus. Если вы станете создавать стили на одном из этих языков, вам придется применять программу или сценарий для преобразования полученного файла в стандартный CSS-документ, понятный браузерам;
- инструменты оптимизации CSS, использующие ваши чистые стан-

НЕСКОЛЬКО СЛОВ ОБ ИНСТРУМЕНТАХ РАЗРАБОТКИ...

Имейте в виду, что перечень инструментов разработки постоянно меняется. Причем весьма быстро, и все сообщество разработчиков, кинувшись использовать один какой-либо фреймворк, мгновенно бросает его в пользу нового. Поэтому сложно описывать конкретные инструменты в книге, которая рассчитана на применение в течение достаточно долгого времени, и я представляю вам здесь наиболее надежные и стабильные инструменты, актуальные на начало 2018-го года, но следует знать, что существует множество подобных инструментов, и, когда вы станете читать эти строки, может получить популярность какой-либо совершенно иной инструмент. Так что при чтении этой главы сосредоточьтесь на функциях, которые выполняют инструменты, упомянутые здесь, но учитывайте спектр новых возможностей.

дартные таблицы CSS и усовершенствующие их, повышая согласованность их восприятия браузерами и уменьшая размеры файлов для увеличения производительности, а также улучшающие многие их характеристики. Инструменты, которые оптимизируют готовые для браузера CSS, широко известны как *постпроцессоры*.

Прежде чем вы освоитесь с терминами *препроцессор* и *постпроцессор*, следует знать, что различие между ними нечеткое. Препроцессорам всегда удавалось выполнять некоторые задачи оптимизации, относящиеся к сфере постпроцессоров, а постпроцессоры начинают выполнять некоторые функции, которые обычно присущи препроцессорам. Разделительные линии между ними размыты, поэтому некоторые разработчики называют все подобные инструменты просто *CSS-процессорами*, включая

НАДЕЮСЬ, ЭТО ПОМОЖЕТ И ВАМ!

Большой привет Штефану Баумgartнеру (*Stefan Baumgartner*), чья статья «*Deconfusing Pre- and Post-Processing*» (medium.com/@ddprrt/deconfusing-pre-and-post-processingd68e3bd078a3) помогла мне разобраться во всем, что касается CSS-обработки, и Дэвиду Кларку (*David Clark*) за его обстоятельную статью «*It's Time for Everyone to Learn About PostCSS*» (davidtheclark.com/its-time-for-everyone-to-learn-about-postcss/).

сюда как специальный синтаксис, увеличивающий скорость разработки, а также CSS-оптимизаторы. Многие функции процессоров CSS также встроены в приобретаемые отдельно сторонние инструменты, такие, как CodeKit (codekitapp.com, только Mac-версия). тем не менее я полагаю, что вам будет полезно познакомиться с традиционной терминологией, поскольку она широко используется, и я собираюсь использовать ее здесь для простоты изложения.

Введение в препроцессоры (на основе Sass)

Препроцессоры представляют возможность разрабатывать стилевые таблицы с использованием особого синтаксиса и переводить (или *компилировать*, говоря языком специалистов) файлы, написанные в этом синтаксисе, в простые традиционные файлы CSS, которые могут использовать браузеры. Такая компиляция осуществляется с помощью специальной программы, включенной в препроцессор (рис. 20.3). Например, в Sass вы пишете на языке синтаксиса Sass и сохраняете файлы с расширением **scss**, указывая тем самым, что они написаны на этом языке, а не с применением CSS-свойств. Программа Sass, изначально написанная на языке Ruby, преобразует SCSS-синтаксис в стандартный CSS-синтаксис и сохраняет

полученный файл с расширением **css**. Инструменты LESS и Stylus (см. далее *врезку «LESS и Stylus»*) функционируют одинаково, но используют для конвертации JavaScript. Эти инструменты устанавливаются и запускаются с помощью командной строки.

NODE SASS

Проект Sass создал на языке C++ более новую версию этого препроцессора, которую можно применять с другими языками. Большинство разработчиков компилируют файлы ***.scss** на Node Sass, поскольку этот компилятор плавно интегрируется в рабочий процесс с другими инструментами Node.js.

Безусловно, самым популярным препроцессором является Sass (сокращение от Syntactically awesome style sheets, синтак-

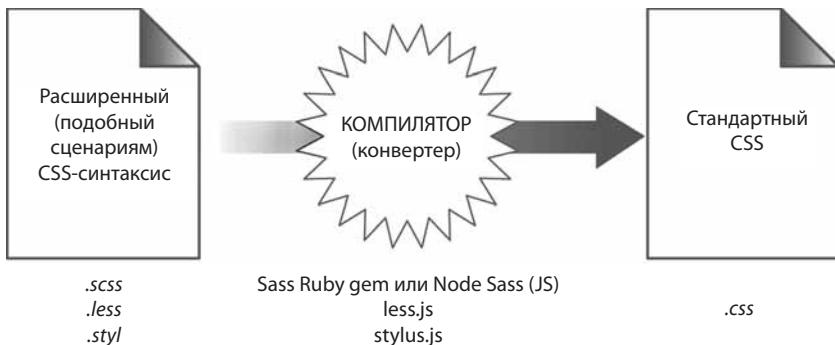


Рис. 20.3. Упрощенное представление препроцессора

LESS И STYLUS

Sass — наиболее широко применяемый препроцессор, но это не единственный инструмент, пригодный для работы с вложениями, переменными, миксинами (об этих понятиях рассказано чуть далее), а также со многими другими объектами.

- LESS (lesscss.org) — еще одно расширение CSS с возможностями, подобными сценариям. Он напоминает Sass, но в нем отсутствуют расширенные функции логики программирования (такие, как операторы `if/else`), а также имеются небольшие различия в синтаксисе. Например, переменные в LESS обозначаются символом `@` вместо `$`. Другое важное отличие заключается в том, что файл LESS преобразуется в обычную CSS-разметку с применением JavaScript ([less.js](#)) вместо Ruby. Обратите внимание, что компиляция LESS-файла в CSS-разметку требует значительных ресурсов процессора и может привести к зависанию браузера. По этой причине лучше перед отправкой на сервер выполнить преобразование LESS-файлов в CSS. LESS поддерживает очень активное сообщество разработчиков, имеется даже смешанная библиотека миксинов «[LESShat](#)».
- Stylus (stylus-lang.com) — относительно новое явление в препроцессорном хозяйстве. Он сочетает логические возможности Sass с удобством компилятора на основе JavaScript ([stylus.js](#)). В нем также предлагается наиболее гибкий синтаксис: можно включать необходимый объем «пунктуации» CSS (скобки, двоеточия и точки с запятыми), добавлять переменные с `$` или нет и обрабатывать имена миксинов как обычные свойства. Разработчикам, применяющим Stylus, нравится, что на нем легко писать и компилировать. Для Stylus также доступны две библиотеки миксинов: Nib и Axis.

Если вы готовы вывести свои усилия по использованию CSS на новый уровень, можете попробовать каждый из этих программных продуктов. Инструмент, который вы выберете, — это вопрос ваших личных предпочтений, однако, если вы работаете в составе профессиональной команды разработчиков, этот выбор может от вас и не зависеть.

тически потрясающие таблицы стилей), созданный Хэмптоном Кэтлином (Hampton Catlin) и Натаном Вейзенбаумом (Nathan Weizenbaum), уставшими от повторяющейся блоков в обычных CSS. Их новый синтаксис позволил CSS-разработчикам использовать варианты сокращений, характерные для сценариев. Первоначально

в нем применялся синтаксис без скобок (он по-прежнему доступен), но более поздняя его версия, известная как SCSS (от Sassy CSS, нахальные CSS), основана на скобочном ({}) формате CSS, который мы знаем и любим. При этом действительный CSS-документ фактически также будет и действительным SCSS-документом. Подобный подход значительно облегчает начало работы с Sass, потому что он оказывается вам знаком и вы можете использовать Sass в таблицах стилей, созданных вами так, как описано в этой книге.

Далее мы рассмотрим несколько примеров применения синтаксиса Sass, что даст вам общее о нем представление. А когда вы будете готовы приступить к серьезному изучению Sass, отличным пособием вам в этом станет книга Дэна Седерхольма (Dan Cederholm) «Sass for Web Designers». Я также приведу в конце этого раздела еще и перечень дополнительных ресурсов.

Сейчас же мы начнем с рассмотрения трех наиболее популярных функций Sass: вложение, переменные и миксины.

Вложение

ВЛОЖЕНИЕ СТИЛЕЙ

Sass позволяет вам вкладывать стили, соответствующие структуре разметки.

вам вложить для элементов `nav`, `ul` и `li` правила стиля, отражающие структуру HTML-разметки. Это избавляет от необходимости многократно выводить селекторы — компилятор Sass выполнит это вместо вас.

В следующем примере показаны вложенные стили в том виде, как они могут быть написаны с применением синтаксиса Sass:

```
nav {  
    margin: 1em 2em;  
  
    ul {  
        list-style: none;  
        padding: 0;  
        margin: 0;  
  
        li {  
            display: block;  
            width: 6em;  
            height: 2em;  
        }  
    }  
}
```

Когда Sass скомпилирует такой SCSS-файл в стандартную CSS-разметку, она будет выглядеть так:

```
nav {  
    margin: 1em 2em;
```

Допустим, у нас имеется HTML-документ с элементом `nav`, который содержит неупорядоченный список для нескольких пунктов меню. Sass позволяет

```
}

nav ul {
    list-style: none;
    padding: 0;
    margin: 0;
}

nav ul li {
    display: block;
    width: 6em;
    height: 2em;
}
```

Переменные

Переменная представляет собой значение, которое можно определить один раз, а затем использовать несколько раз во всей таблице стилей. Например, издательство O'Reilly неоднократно применяет на своем сайте один и тот же оттенок красного, поэтому разработчики могут создать переменную с именем `$oreilly-red` и использовать это имя для задания значений цвета. При этом, если позже потребуется изменить этот оттенок, надо будет изменить значение переменной (фактический цвет RGB) только в одном месте. Продемонстрируем, как выглядит настройка и применение переменных в Sass:

```
$oreilly-red: #900;

a {
    border-color: $oreilly-red;
}
```

Когда этот код компилируется в стандартный CSS, значение переменной вставляется туда, где оно вызывается:

```
a {
    border-color: #900;
}
```

Преимущество переменной заключается в том, что можно изменить ее значение в одном месте вместо поиска и замены по всему документу. Если команды таким образом используют имена переменных, поддерживается согласованность стилей на всем сайте в целом.

Миксины

Sass позволяет повторно применять ценные наборы стилей, используя соглашение под названием **миксины**. В следующем примере комбинация стилей фона, цвета и границ сохраняется в виде миксина под названием `special`. Для применения этой комбинации стилей включите ее в объявление с помощью ключевого слова `@include` и вызовите по имени:

ИСПОЛЬЗОВАНИЕ ПЕРЕМЕННЫХ

Переменная — это значение, которое определяется один раз, после чего может использоваться в таблице стилей многократно.

ПРИМЕНЕНИЕ МИКСИНОВ

Миксины — это наборы правил, которые можно использовать повторно.

```
@mixin special {
    color: #fff;
    background-color: #befc6d;
    border: 1px dotted #59950c;
}
a.nav {
    @include special;
}
a.nav: hover {
    @include special;
    border: 1px yellow solid;
}
```

После компиляции окончательная CSS-разметка выглядит следующим образом:

```
a.nav {
    color: #fff;
    background-color: #befc6d;
    border: 1px dotted #59950c;
}
a.nav: hover {
    color: #fff;
    background-color: #befc6d;
    border: 1px dotted #59950c;
    border: 1px yellow solid;
}
```

Обратите внимание, что при наведении (состояние `hover`) мыши на объект имеется второе объявление границы, которое переопределяет значения с применением миксина, что весьма удобно. Миксины также являются популярными решениями для работы с префиксами поставщиков. Далее приводится миксин для `border-radius`, включающий *аргумент* (заполнитель для предоставляемого значения обозначается знаком \$):

```
@mixin rounded($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    border-radius: $radius;
}
```

При включении миксина в правило стиля укажите значение для `$radius`, и оно будет добавлено в каждый экземпляр объявлений:

```
aside {
    @include rounded(.5em);
    background: #f2f5d5;
}
```

Этот SCSS-код компилируется в CSS следующим образом:

```
aside {
    -webkit-border-radius: .5em;
    -moz-border-radius: .5em;
    border-radius: .5em;
    background: #f2f5d5;
}
```

Построение миксина вокруг заполняемых пустых аргументов делает их многократно применяемыми и даже разделяемыми. Многие разработчики формируют собственные библиотеки миксинов и используют их во многих проектах. Вы также можете воспользоваться преимуществами существующих библиотек миксинов в таких инструментах, как Compass (среда разработки CSS с открытым исходным кодом, которая доступна по адресу compassstyle.org) или Bourbon (bourbon.io). К тому времени, как вы будете читать эти строки, могут появиться и другие варианты, поэтому не пренебрегайте возможностями поиска, чтобы их заметить.

Ресурсы Sass

Вложения, переменные и миксины — это лишь малая часть того, что делает Sass. С его помощью могут выполняться математические операции, математическое «затемнение» и «осветление» цвета «на лету», обрабатываться операторы `if` / `else` и еще многое другое.

Когда вы, приобретя некоторый опыт, почувствуете, что готовы перейти на новый уровень, обратитесь к следующим статьям и ресурсам Sass и LESS:

- Сайт Sass (sass-lang.com)
- Дэвид ДеМари (David Demaree), «Getting Started with Sass» (alistapart.com/articles/getting-started-with-sass);
- Джереми Хиксон (Jeremy Hixon), «An Introduction to LESS, and LESS Vs. Sass» (www.smashingmagazine.com/2011/09/an-introduction-to-less-and-comparisonto-sass/)

Введение в постпроцессоры (на основе PostCSS)

Как уже упоминалось ранее, постпроцессоры — это сценарии, которые оптимизируют стандартный код CSS, улучшая его (рис. 20.4). «Улучшенный» код получает последовательную и безошибочную поддержку браузеров, но существуют сотни сценариев постпроцессоров, которые выполняют еще множество более интересных действий.



Рис. 20.4. Постпроцессоры оптимизируют имеющиеся стандартные CSS-файлы

Основой постпроцессоров является инструмент Autoprefixer, который просматривает написанные вами CSS-стили, сканирует их на предмет свойств, которым требуются префиксы поставщика, а затем автоматически вставляет нужные префиксные свойства. В результате обеспечивается значительная экономия времени и трудозатрат!

В главе 16 мы использовали Autoprefixer для генерирования требуемых префиксов через интерфейс его веб-страницы (autoprefixer.github.io). Несмотря на удобство работы с веб-страницей (особенно в период обучения), постпроцессоры чаще всего реализуются с помощью обработчиков задач типа Grunt или Gulp. Мы кратко рассмотрим их в следующем разделе этой главы.

ЕСТЬ И ДРУГИЕ ПОСТПРОЦЕССОРЫ...

PostCSS — это не единственный постпроцессор. Другие фреймворки включают постпроцессоры Rework (github.com/reworkcss/rework) и Pleeease (pleeease.io), но их функциональность не столь полна. Ко времени, когда вы будете читать эти строки, могут появиться и иные средства. Мир инструментов веб-разработки находится в процессе постоянного развития.

На момент подготовки книги в области постпроцессоров доминирует PostCSS (postcss.org) — «инструмент для преобразования CSS с помощью JavaScript», созданный Андреем Ситником (Andrey Sitnik), который также является и автором Autoprefixer. Инструмент PostCSS представляет собой как программу на основе JavaScript (точнее, модуля Node.js), так и экосистему создаваемых сообществом плагинов, которые решают все CSS-проблемы.

Инструмент PostCSS исследует CSS (или CSS-подобный синтаксис, создаваемый инструментами Sass или LESS), анализирует ее структуру и формирует результирующее «дерево», делая его доступным для плагинов, манипулирующих кодом.

Этот открытый API-интерфейс позволяет каждому пользователю легко создавать плагины PostCSS, в результате чего появились буквально сотни созданных и совместно используемых разработчиками плагинов (их можно найти на сайте www.postcss.parts).

Они могут быть как жизненно важными для всех, так и известными лишь посвященным, но, поскольку система эта модульная, вы можете выбирать только те плагины, которые считаете полезными или даже создавать свои собственные. Упомянем далее некоторые из них:

- Stylelint (stylelint.io) — проверяет CSS-файл на наличие синтаксических ошибок (этот процесс называется линтингом) и присутствие избыточности;
- CSSNext (cssnext.io) — позволяет уже сегодня применять будущие функции CSS уровня 4, создавая резервные варианты, работающие в браузерах, в которых эти функции еще не реализованы;
- PreCSS (github.com/jonathantneal/precss) — это набор плагинов, который позволяет создавать Sass-подобный синтаксис (циклы, условные выражения, перемен-

ные, миксины и т. д.) и преобразовывать его в стандартную CSS-разметку. Это пример постпроцессора, используемого для облегчения разработки кода, когда грань между пре- и постобработкой размыта;

- Fixie (github.com/tivac/fixie) — вставляет в код дополнения, необходимые для реализации эффектов при работе с устаревшими версиями Internet Explorer (по сути, это «заплатка» для IE);
- преобразователи формата цвета преобразуют альтернативные форматы цвета (такие, как HWB, HCL и hex + альфа-канал) в стандартный RGB или шестнадцатеричный формат;
- плагин Pixrem — конвертирует единицы измерения rem в пиксели для не поддерживающих их браузеров;
- плагин List-selectors — выводит список и классифицирует селекторы, которые вы использовали в своей таблице стилей для проверки кода. Это пример плагина, который не изменяет ваш файл, но дает вам полезную информацию о нем.

Из этого небольшого списка инструментов вам, вероятно, станет понятно, почему постпроцессоры столь популярны. Они освобождают вас от необходимости писать CSS с нужным вам синтаксисом, основанном на передовых свойствах и значениях, но при этом гарантируют, что все это будет хорошо работать в браузерах. Используя их, вам также можно не вдаваться в подробности относительно всех особенностей браузеров, их прошлом и настоящем состояниях. И, даже если вы не готовы взять эти инструменты на вооружение сейчас, узнать о них подробнее все же стоит. Так что познакомьтесь со следующими источниками для получения дополнительной информации:

- Статья Дрю Миннса (Drew Minns) «PostCSS: A Comprehensive Introduction» для журнала Smashing Magazine (www.smashingmagazine.com/2015/12/introduction-to-postcss/);
- Учебник по Envato Tuts+ «PostCSS Deep Dive» (webdesign.tutsplus.com/series/postcss-deep-dive--cms-889).

GRUNT И GULP: ИНСТРУМЕНТЫ ДЛЯ СБОРКИ КОДА

В мире программного обеспечения для тестирования исходного кода и его компиляции в исполняемые программы нельзя обойтись без *процесса сборки*. Поскольку сайты из наборов статических HTML-файлов превратились в сложные JavaScript-зависимые приложения, часто генерируемые из шаблонов, инструменты сборки также стали неотъемлемой частью рабочего процесса веб-разработки.

Такие инструменты веб-сборки, как Grunt и Gulp, обычно называются *обработчиками задач*. Они применяются для определения и запуска различных задач из файлов рабочего кода HTML, JavaScript, CSS и изображений при подготовке их к публикации (впрочем, все это вы можете сделать и вручную из командной строки).

Автоматизация

Вы можете автоматизировать свои задачи, чтобы они выполнялись в фоновом режиме без необходимости ввода каких-либо команд. Для этого вы поручаете инструменту сборки «следить» за изменениями в ваших файлах и папках. При обнаружении изменения инструмент согласно своей настройке автоматически запустит соответствующие задачи.

ПРЕОБРАЗОВАНИЕ SCSS-ФАЙЛОВ

Существует плагин Grunt, предназначенный для преобразования SCSS-файлов, но он не так полнофункционален, как Ruby.

Представьте, что вы вносите изменения в свой файл Sass и сохраняете его. Grunt сразу же увидит, что файл SCSS изменился, автоматически преобразует его в CSS, после чего перезагрузит браузер, чтобы отразить ваши изменения.

Настроив обработчик задач на просмотр своих файлов, вы можете заняться написанием CSS, а все остальное будет происходить для вас незаметно, даже не затрагивая приложение терминала.

Некоторые общие задачи

В предыдущем разделе вы получили представление о некоторых моментах, которые хорошо бы автоматизировать. Позвольте мне привести еще несколько примеров задач, которые помогут сформировать у вас полное впечатление о том, каким образом обработчики задач облегчат вашу работу:

- **конкатенация** — обычно веб-разработчики делят таблицы стилей и сценарии на небольшие специализированные части в файлах `*.css` и `*.js`. Однако, когда приходит время публикации, в целях повышения производительности необходимо использовать как можно меньше обращений к серверу, вследствие чего эти небольшие части нужно конкатенировать (объединить) в мастер-файлы;
- **сжатие и «минимизация» кода** — это еще один способ повышения производительности: насколько возможно уменьшить файлы, удалив из них ненужные пробелы и символы возврат строк. Инструменты сборки могут сжать CSS и минимизировать код JavaScript;
- **проверка на наличие ошибок** ваших HTML, CSS и JavaScript (линтинг);
- **оптимизация изображений** — с помощью инструментов, которые уменьшают размер файлов всех изображений в каталоге;
- **помощь в фиксации или отправке изменений** в репозиторий контроля версий (Git);
- **перезагрузка браузера** — для отражения любых изменений, которые вы только что внесли в файл (плагин LiveReload);
- **создание окончательных HTML-файлов** на основе шаблонов и данных контента (см. врезку «*Создание сайтов с помощью данных и шаблонов*»);
- **запуск пре- и постпроцессоров CSS**.

СОЗДАНИЕ САЙТОВ С ПОМОЩЬЮ ДАННЫХ И ШАБЛОНОВ

В этой книге HTML-код для веб-страниц создавался вручную путем заключения в теги элементов содержимого в логическом порядке следования кода. Весь контент страницы при этом содержится непосредственно в документе *.html. Конечно, вполне допустимо создавать целые сайты из подобных статических веб-страниц, но в реальном мире, где сайты могут включать тысячи страниц с предназначенным для отдельных пользователей контентом, требуется более надежное решение.

В наши дни более распространено использование системы шаблонов или фреймворков для создания веб-страниц из хранящегося в виде данных контента. Шаблоны применяют обычную HTML-разметку, поэтому все изученное вами на этот момент ценности не теряет, но вместо конкретного контента — для извлечения контента из базы данных или файла данных — между тегами размещаются специальные маркеры данных.

Существует огромное число различных инструментов, применяемых для создания сайтов, но рассмотрение всех их выходит за рамки этой книги. Однако, как обычно, мне хочется дать вам представление о процессе создания шаблонов.

Когда-то мне пришлось работать с сайтом, при создании которого применялся шаблонный инструмент Handlebars (handlebarsjs.com), извлекающий контент из файлов данных, написанных на языке YAML (www.yaml.org/start.html). Рассмотрим небольшой пример того, как шаблон и данные применялись для сборки показанного на рис. 20.5 веб-контента.



Рис. 20.5. Небольшая часть веб-страницы, созданной с помощью Handlebars и YAML

Вот небольшой фрагмент данных в YAML-файле (*.yml):

```
speaker--name: "Jennifer Robbins"
speaker--description: "Designer, Author, ARTIFACT
Co-founder"
speaker--photo: "/img/speakers/jennifer-robbins.
jpg"
#HTML
speaker--biography: |
    <p>Jennifer has been designing for the web since
1993 when she worked on the first commercial web
site, GNN, from O'Reilly Media. Since then she has
gone on to write several books on web design for
O'Reilly, including <i>Web Design in a Nutshell
```

▶ </i>, <i>Learning Web Design</i>, and the <i>HTML5 Pocket Reference</i>. More recently, Jennifer's days are filled with organizing the ARTIFACT Conference. ...</p>
speaker--links:
 - link--label: "Website"
 link--target: "http://www.jenville.com"
 link--title: "jenville.com"
 - link--label: "Twitter"
 link--target: "http://www.twitter.com/jenville"
 link--title: "@jenville"

А теперь рассмотрим разметку в документе шаблона Handlebars (`speaker.hbs`) — я немного подправила его для краткости. Взглянув на выделенный код, вы заметите, что вместо фактического содержимого в фигурных скобках содержатся метки данных, которые используются в файле YAML. (Если повернуть фигурную скобку набок, она станет похожа на усы, подкрученные вверх, отсюда и название — handlebars, что по-английски это и означает). Обратите внимание, что шаблон имеет разметку для одной пары метка/ссылка, но он организован в цикл и отображает в файле данных все ссылки `speaker--links`:

```
<div class="layout--container">  
  <div class="speaker--photo-container">  
      
    <div class="speaker--biography">  
      {{{page-data.speaker--biography}}}  
    </div>  
    <ul class="speaker--links">  
      {{#each page-data.speaker--links}}  
        <li class="speaker--link-item">{{link--label}}:  
          <a href="http://{{link--target}}" class="speaker--link">{{link--title}}</a></li>  
        {{/each}}  
    </ul>  
  </article>  
</div>
```

Это только один пример того, как шаблоны сокращают избыточность в разметке. Прямо на главной странице сайта Handlebars (handlebarsjs.com) представлено хорошее описание семантических шаблонов, если вам необходима дополнительная информация о принципах их работы.

Конечно, браузеры не имеют представления о том, что делать с этими форматами файлов, поэтому, прежде чем сайт будет опубликован, его необходимо собрать или слить, объединяя все данные в модули шаблона, а все модули — в целые веб-страницы. Это работа сценариев и инструментов сборки, подобные представленным в этом разделе. Надеюсь, этот краткий пример даст вам представление о принципах работы созданных с использованием шаблонов и данных сайтов.

Инструменты Grunt, Gulp и другие

Первым и наиболее признанным инструментом для веб-сборки является Grunt (gruntjs.com), который, предположительно, назван так, поскольку выполняет для вас «тяжелую работу» (grunt по-английски «ворчание»). Это инструмент JavaScript, сформированный на основе Node.js с открытым исходным кодом, управляемый с помощью командной строки. Самое замечательное в Grunt — это то, что сообщество разработчиков создало для него буквально тысячи плагинов, которые выполняют практически любую задачу, о которой вы только можете мечтать. Просто скачайте этот инструмент, настройте его и начните использовать. Вам не нужно быть гуру JavaScript, чтобы сразу приступить к делу.

Еще одним популярным инструментом считается Gulp (gulpjs.com), преимущество которого в том, что работает он немного быстрее, но требует и больших технических знаний, чем Grunt, поскольку его нужно конфигурировать с применением реального кода JS. Другими претендентами на эту роль на момент подготовки книги могут служить весьма популярный Webpack, а также Browserify, Brunch (по-английски «поздний завтрак») и Broccoli («брокколи») — новые инструменты с подобными забавными названиями появляются регулярно. Некоторые разработчики просто используют скрипты на основе Node.js, не используя программы для запуска задач в качестве посредника, поэтому встречается большое число вариантов таких скриптов.

Когда вы считете себя готовыми автоматизировать свой рабочий процесс, то найдете в Интернете много онлайн-уроков, посвященных тому, как загрузить и настроить выбранный вами инструмент сборки. Я надеюсь, что ознакомила вас с их возможностями, и когда ваш собеседник на работе упомяннет Grunt и Gulp (по-английски «давиться»), вы поймете, что он не страдает расстройством желудка.

КОНТРОЛЬ ВЕРСИЙ С ПОМОЩЬЮ GIT И GITHUB

Работая на компьютере, вы, вероятно, используете какую-нибудь систему для отслеживания версий своей работы. Вы могли придумать систему именования черновиков, пока не доберетесь до «окончательной» версии (и «окончательно-финальной» версии, и «окончательно-окончательно-финальной» версии и т. д.). Вы также можете воспользоваться средством Time Machine компьютеров на macOS для сохранения версий, к которым можно вернуться в случае чрезвычайной ситуации. Или же вы могли использовать одну из профессиональных систем контроля версий, которые использовались командами разработчиков на протяжении десятилетий.

Королем систем контроля версий (Version Control Systems, VCS) для веб-разработки признана мощная программа под названием Git (git-scm.com). Умение работы с Git является обязательным требованием, если вы входите в команду разработчиков, но это также просто хороший навык, который можно применять при работе над своими проектами.

ЛИНУС ТОРВАЛЬДС — РАЗРАБОТЧИК GIT

Система контроля версий Git была разработана Линусом Торвальдсом, создателем операционной системы Linux, когда ему понадобился способ, позволяющий сообществу разработчиков вносить свой вклад в проект Linux.

ницу между терминами «ветвь» и «развилка», что весьма полезно для начинающих, я здесь уточню. Ну, а теперь приступим к работе.

И начну я с основной посылки: Git — это программа контроля версий, которая запускается на компьютере пользователя, а GitHub (github.com) — это сервис, который размещает проекты Git либо бесплатно, либо за плату. С помощью Git вы взаимодействуете с GitHub — либо с помощью командной строки и пользовательского интерфейса на веб-сайте GitHub, либо с помощью автономного приложения, которое предлагает графический интерфейс для команд Git. Поскольку в свое время эти вопросы вызывали у меня затруднения, остановимся на них подробнее. GitHub и подобные ему сервисы — это веб-оболочки для Git, предлагающие такие функции, как отслеживание ошибок, инструментарий для проверки кода и веб-интерфейс для просмотра файлов и предысторий. Эти функции весьма удобны, но имейте в виду, что вы также можете настроить Git на собственном сервере и поделиться им с членами своей команды, не привлекая сторонние сервисы, такие, как GitHub.

Зачем нужен Git?

Сервисы Git и GitHub обладают рядом возможностей, позволяющих сделать их частью рабочего процесса. Прежде всего, если возникнут проблемы, можно легко вернуться к более ранней версии проекта.

СЕРВИСЫ GIT-ХОСТИНГА

Сервисы Beanstalk (beanstalkapp.com), GitLab (gitlab.com) и Bitbucket (bitbucket.org) — это еще варианты сервисов Git-хостинга, предназначенные для проектов масштаба предприятия. Сервис GitLab включает бесплатные возможности для публичных проектов, аналогичных GitHub, и, поскольку имеет открытый исходный код, вы можете развертывать его самостоятельно.

изменения в одной основной версии документа. В качестве дополнительного преимущества процесс такого обмена служит способом получения предварительной информации о работе, прежде чем она будет включена в проект. Вы также можете организовать свободное сотрудничество в публичном проекте, поощряя вклад людей, которых вы даже не знаете, безопасным и управляемым способом. Git —

В этом разделе я познакомлю вас с терминологией и некоторыми моментами, которые помогут вам начать работать с Git. Однако знакомство со всеми особенностями настройки и применения Git с помощью командной строки — это тема для другой книги и учебников в Интернете (некоторые из них упоминаются в конце этого раздела). Но раз-

ницу между терминами «ветвь» и «развилка», что весьма полезно для начинающих, я здесь уточню. Ну, а теперь приступим к работе.

И начну я с основной посылки: Git — это программа контроля версий, которая запускается на компьютере пользователя, а GitHub (github.com) — это сервис, который размещает проекты Git либо бесплатно, либо за плату. С помощью Git вы взаимодействуете с GitHub — либо с помощью командной строки и пользовательского интерфейса на веб-сайте GitHub, либо с помощью автономного приложения, которое предлагает графический интерфейс для команд Git. Поскольку в свое время эти вопросы вызывали у меня затруднения, остановимся на них подробнее. GitHub и подобные ему сервисы — это веб-оболочки для Git, предлагающие такие функции, как отслеживание ошибок, инструментарий для проверки кода и веб-интерфейс для просмотра файлов и предысторий. Эти функции весьма удобны, но имейте в виду, что вы также можете настроить Git на собственном сервере и поделиться им с членами своей команды, не привлекая сторонние сервисы, такие, как GitHub.

Зачем нужен Git?

Сервисы Git и GitHub обладают рядом возможностей, позволяющих сделать их частью рабочего процесса. Прежде всего, если возникнут проблемы, можно легко вернуться к более ранней версии проекта.

И, поскольку каждое внесенное в проект изменение регистрируется и записывается, легко определить, в какой момент произошел сбой в работе.

Git также облегчает совместную работу над общим исходным кодом. С его помощью вы можете тесно сотрудничать с одним или несколькими разработчиками в частном проекте, объединяя все

любимый всеми инструмент для такого рода совместной работы над всевозможными проектами с открытым исходным кодом.

И особенно важно освоить GitHub, потому что этот сервис используют все разработчики. Если ваш проект является публичным (доступен каждому), хостинг будет бесплатным. Но для частных и коммерческих проектов GitHub взимает плату за хостинг. Помимо хостинга проектов сервис предоставляет инструменты для совместной работы — например, обеспечивающие отслеживание проблем.

Возможно, вы заметили, что некоторые ссылки на инструменты, которые упоминались в книге, ведут в репозиторий GitHub. Вы должны знать, что можете получить, обратившись к этому репозиторию.

Как работает Git?

Git сохраняет копию каждой версии ваших файлов и папок по мере их создания, при этом каждое изменение (называемое *commit*, фиксацией) регистрируется с уникальным идентификатором (сгенерированным Git), написанным вами сообщением, описывающим изменения, и другими метаданными. Все версии и журнал фиксации хранятся в репозитории, часто называемом «репо».

Установленная на компьютере программа Git каждый раз при создании нового репозитория или клонировании существующего наряду с другими файлами в папку проекта добавляет каталог и файлы, представляющие метаданные репо. После инициализации репозитория Git можно фиксировать изменения и воспользоваться функцией «машины времени», если надо вернуться к более ранней версии документов. Таким образом, Git является хорошим инструментом для рабочего процесса, выполняемого в одиночку.

Скорее всего, над проектом вы будете работать в команде. В таком случае используется модель концентратора (*hub*), когда на центральном сервере имеется официальный репозиторий, в котором каждый член группы создает для работы свою локальную копию. При этом он работает на своем компьютере, а его работа фиксируется в его локальном репозитории и через определенные интервалы загружается в центральное хранилище.

Именно поэтому Git является распределенной системой контроля версий по сравнению с другими системами, такими, как SVN, которые требуют фиксировать каждое изменение непосредственно на сервере. То есть с помощью Git вы можете работать как локально, так и в автономном режиме.

Первая часть изучения Git — это освоение его словаря. Так что давайте рассмотрим некоторые термины, которые пригодятся вам в процессе изучения Git и сервиса GitHub. На рис. 20.6 представлена упрощенная схема, помогающая визуализировать соединение частей сервиса в единое целое.

GIT — НАИБОЛЕЕ УДАЧНЫЙ ИНСТРУМЕНТ СОВМЕСТНОЙ РАБОТЫ

Сервис Git — это наиболее удачный инструмент, применяемый при совместной работе над проектами с открытым исходным кодом.

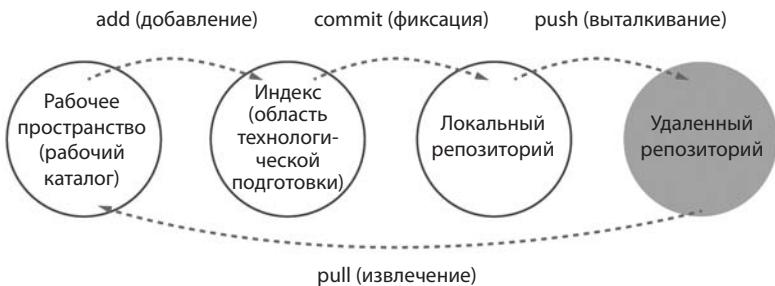


Рис. 20.6. Визуализация структуры Git

РЕСУРСЫ GIT-ВИЗУАЛИЗАЦИИ

Требуется дополнительная помощь, чтобы представить, каким образом эти части структуры и команды функционируют вместе? Обратитесь к следующим ресурсам визуализации:

- Git Cheatsheet от NDP Software — предоставляет полное интерактивное отображение того, каким образом различные команды Git соответствуют рабочей области, локальным и удаленным репозиториям. Этот ресурс доступен по адресу: ndpsoftware.com/gitcheatsheet.html#loc=workspace;
- визуальная Git-ссылка (marklodato.github.io/visual-git-guide/indexen.html) — представляет набор диаграмм, демонстрирующих наиболее распространенные команды Git;
- «Understanding the GitHub Flow» (guides.github.com/introduction/flow/) — описывает типичный поток в GitHub.

Рабочий каталог

Вашим *рабочим каталогом* является каталог файлов на вашем компьютере, в котором вы выполняете реальную работу. Ваша рабочая копия файла — это файл, в который можно вносить изменения, или, иначе говоря, это файл, который можно открыть с жесткого диска компьютера, используя средства Finder или My Computer.

Репозиторий

Ваш локальный *репозиторий* Git находится рядом с файлами в вашем рабочем каталоге. В нем хранятся копии (снимки) всех файлов проекта на каждом из этапов его разработки, хотя они и остаются скрытыми. Он также содержит сохраненные после каждого изменения метаданные. Кроме того, может существовать и центральный репозиторий (хранилище) проекта, расположенный на удаленном сервере, таком, как GitHub.

Фиксация

Фиксация (commit) — это наименьшая единица взаимодействия с Git и основная часть того, что вы будете выполнять при работе с Git. Git трактует «фиксацию» и как глагол, и как имя существительное. В процессе работы вы можете «сохранять» рабочий документ так часто, как пожелаете, но лишь при намеренном добавлении текущей версии файла в репозиторий выполняется ее фиксация (как глагол). Обычно это происходит тогда, когда вы фиксируете в рабочем процессе логическую паузу — например, исправили ошибку или завершили изменение набора стилей.

При выполнении фиксации Git записывает состояние всех файлов проекта, определяет метаданные изменения, включая имя пользователя, адрес его электронной почты, дату и время, уникальный многозначный идентификационный номер (см. врезку «Хеши»), и создает сообщение, описывающее изменения. Эти сохраненные записи называются **фиксациями** (как существительное). Такая фиксация представляет собой снимок вашего репозитория — каждого файла, который он содержит, — на тот момент, когда вы произвели фиксацию.

Фиксации добавляются всегда, поэтому даже при удалении файла Git добавляет фиксацию в свой стек. Список фиксаций доступен для ознакомления в любое время. Для просмотра списка изменений файла или папки на GitHub применяйте кнопку **History** (Журнал).

Уровень детализации в фиксациях позволяет вам просматривать репозиторий (проект) в любом состоянии, в котором он когда-либо был. То есть вы никогда не потеряете сделанную работу, даже если продолжаете работать над ней все дальше и дальше. Это отличная сеть безопасности. Косвенно это также означает, что вы ничего не можете сделать с Git, чего нельзя было бы отменить — поэтому вы никогда не попадете в невозможную ситуацию.

Технологическая подготовка

Прежде чем вы зафиксируете изменение, необходимо сначала сообщить Git о файле (или *отследить* его, если использовать технический термин). Этот процесс называется *технологической подготовкой файла* (*staging*) для добавления его в Git. В командной строке для этого нужно выполнить команду `git add имя_файла`, а визуальные инструменты могут содержать кнопку **Add** (Добавить), позволяющую выполнить подготовку файла. В результате создается локальный *индекс* файлов, которые вы намереваетесь зафиксировать в своем локальном хранилище, однако не сама их фиксация. Стоит отметить, что перед фиксацией вам нужно «добавить» любые файлы, который вы изменили, а не только вновь созданные. Поначалу вам может потребоваться некоторое время, чтобы привыкнуть к этой процедуре, поскольку она не носит интуитивного характера.

Ветка

Ветка (*branch*) — это последовательный ряд фиксаций, также иногда называемый *стеком* фиксаций. Самая последняя фиксация в любой ветке называется ее *заголовком* (*head*).

ХЕШИ

Уникальный идентификатор, который генерирует Git для каждого подтверждения, технически называется *хешем SHA-1*, но в мире разработчиков он более известен как просто *хеш*. Это строка, включающая 40 символов, записанная в шестнадцатеричном формате (используются символы 0–9 и A–F), поэтому вероятность дублирования хеша является ничтожно малой. Обычно в проектах применяются короткие хеши вместо полных, 40-символьных. Например, на GitHub короткие хеши имеют длину, равную семи символам, — их можно увидеть в таких местах, как страница **Commits** проекта. Даже при наличии всего семи символов шансы на возникновение коллизий ничтожны.

Вы можете представить себе ветку как «ось» развития проекта. Проекты обычно имеют основную ветку (ветку по умолчанию), часто (хотя и не обязательно) называемую *мастер*, которая является официальной версией проекта. Чтобы работать в ветке, вам нужно ее *проверить* (checked out).

ЗАГОЛОВОК — ЭТО САМАЯ ПОСЛЕДНЯЯ ФИКСАЦИЯ

Имеют место исключения, поскольку порядок следования фиксаций можно изменить, однако почти всегда заголовок окажется и самой последней фиксацией.

Ветви часто используются для таких небольших специфических задач, как эти, но создать новую ветку вы можете для любых целей.

Например, если вы работаете с веткой «мастер», но хотите исправить ошибку, можно создать новую ветку вне ветки «мастер» и дать ей содержательное имя — например: `bugfix`. Вы можете рассматривать ветку `bugfix` как копию ветки «мастер» в той точке, где было создана ветка `bugfix` (рис. 20.7), хотя это не вполне точно отражает то, что происходит при этом «за кулисами».

Чтобы поработать с веткой `bugfix`, сначала нужно ее проверить (командой `git checkout bugfix`), после чего вы можете заняться внесением в нее изменений, сохранением их, добавлением в Git и фиксацией. В конечном счете новая ветвь заканчивается историей фиксаций, отличной от исходной ветки.



Рис. 20.7. Создание и объединение новой ветки

Закончив работу с новой веткой, вы можете объединить (слить) выполненные вами изменения с тем, что находится в исходной ветке, и удалить новую ветку. Если вам не понравился результат работы в новой ветке, удалите ее без слияния.

Слияние

Слияние (merge) — замечательная функция Git, обеспечивающая общий доступ к коду. Вы можете выполнять слияние фиксаций из одной ветки с фиксациями другой (например, всех раздельных веток с мастер-веткой) или выполнять слияние различных версий одной и той же ветки, находящихся на разных компьютерах. Согласно документации Git, слияние «включает изменения именованных фиксаций (с тех пор как их истории расходятся с текущей веткой) в текущую ветку». Иными словами, Git рассматривает слияние как «объединение двух историй в одну», поэтому полезно воспринимать слияние как получение финальной фиксации.

Работая в ветке, вы в любой момент можете запустить новую ветку, чтобы выполнить небольшую работу, не затрагивая исходную ветку. Начать новую ветку можно, чтобы, например, поэкспериментировать с новой функцией, отладить или «поиграть» с презентацией.

Git пытается объединить каждую фиксацию одну за другой в целевую ветку. Если изменилась только одна ветвь, другая ветвь может просто выполнить *ускоренную перемотку вперед*, чтобы отследить изменения. Если в обеих ветвях есть фиксации, которых нет в другой ветке, — то есть если в обеих ветвях есть изменения, — Git просматривает каждую из имеющихся фиксаций и пытается построчно объединить различия. И при этом автоматически изменяет код внутри файлов, чтобы вам не приходилось самим отыскивать то, что изменилось.

Однако, если Git обнаруживает *конфликты*, такие, как два разных изменения, произведенных в одной строке кода, он предоставляет вам отчет о конфликтах, а не пытается изменить сам код. Информация о конфликте вписывается в исходные файлы между символами ===== и <<<<< (рис. 20.8). В случае возникновения конфликтов пользователь должен просмотреть их список и вручную изменить файл, сохраняя требуемое изменение и удаляя лишнее. После разрешения конфликтов файлы необходимо добавить и оформить фиксацией снова.

```
55 p {  
56   margin-top: 0;  
57   margin-bottom: 1rem;  
58 }  
59 .container > p {  
60   <<<<< big-load-comin-through-container  
61   margin: .6rem auto 1rem;  
62   max-width: 880px;  
63   ======  
64   margin: .5rem auto 1rem;  
65   max-width: 900px;  
66   >>>> gh-pages  
67 }  
68  
69 hr {  
70   max-width: 100px;  
71   margin: 3rem auto;  
72   border: 0;  
73   border-top: .1rem solid #eee;  
74 }
```

Рис. 20.8. Отчет о конфликтах в GitHub

Удаленные репозитории

Все рассмотренные до сих пор функции (фиксации, ветки, слияния) можно реализовать на вашем локальном компьютере, но гораздо чаще Git используется при работе с одним или несколькими *удаленными* репозиториями. Удаленный репозиторий может находиться на другом компьютере в вашей организации, но, скорее всего, он размещается на удаленном сервере, таком, как GitHub. Координация с удаленным репозиторием приоткрывает несколько дополнительных ключевых функций Git.

Клон

Клонирование создает точную копию репозитория и всего, что в нем содержится. Распространено клонирование репо с удаленного сервера на ваш собственный компьютер, но также возможно локальное клонирование в другой каталог. Если вы приступаете к работе над существующим проектом, имеет смысл начать с создания локального клона репозитория проекта.

Выталкивание/извлечение

Когда вы работаете с удаленным репозиторием, вам, несомненно, требуется выполнить выгрузку своих изменений на сервер и загрузку материалов с сервера на свой локальный компьютер. Процесс перемещения данных из локального репозитория в удаленный называется *выталкиванием* (push). Отправленные на удаленный сервер фиксации автоматически объединяются с находящейся на сервере текущей версией. Для обновления локальной версии версией, которая находится на сервере, вы *извлекаете ее* (pull), получая метаданные об изменениях и применяя изменения к своим рабочим файлам. Вы можете воспринимать выталкивание и извлечение как удаленную версию слияния.

Хорошей практикой принято считать частое извлечение удаленного мастера, чтобы обновлять с его помощью свою локальную копию. Это помогает устранивать конфликты, особенно если над кодом работают несколько программистов. Многие визуальные инструменты GUI Git предоставляют кнопку синхронизации, которая выполняет функцию «извлекай/выталкивай» за одно нажатие.

СНАЧАЛА ВЫТАЛКИВАНИЕ, А УЖЕ ЗАТЕМ ИЗВЛЕЧЕНИЕ

Всегда перед выталкиванием выполняйте сначала извлечение — чтобы избежать конфликтов.

Развилка

Вы, наверное, слышали разговоры о *развилке* (forking) репо на GitHub. В результате развилки создается копия (fork) репозитория GitHub для учетной записи GitHub — и вы становитесь обладателем своей собственной копии репозитория, с которой можно экспериментировать. Наличие репо в учетной записи — не идентично наличию рабочей копии на вашем компьютере, поэтому после развилки необходимо клонировать (скопировать) копию репозитория на свой компьютер (рис. 20.9).

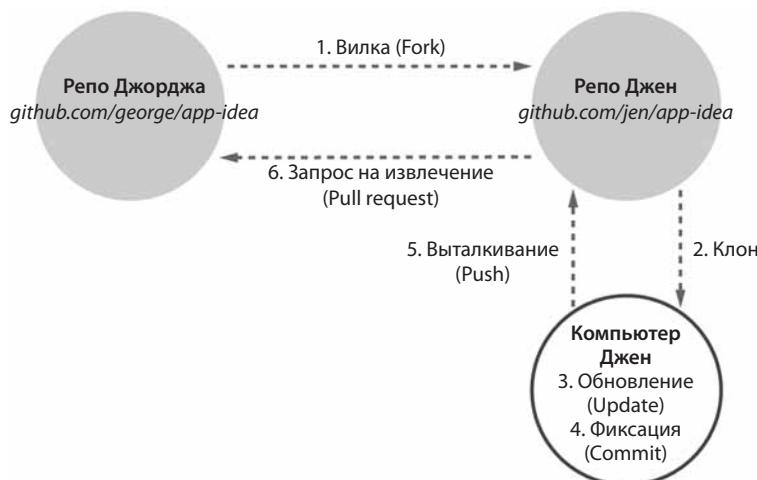


Рис. 20.9. Выполнив развилку репозитория на GitHub, для получения локальной рабочей копии нужно его клонировать. Основано на диаграмме Кевина Маркхэма (Kevin Markham)

Пользователи делают развлечки (форки) репозиториев проектов на GitHub по разным причинам. Возможно, им просто хочется «заглянуть под капот». Возможно, они хотят повторить проект или придать ему новое звучание. Возможно, они стремятся внести свой вклад в проект в форме запроса на извлечение. В любом случае развлечка является гарантией сохранности репозитория для его владельцев, поэтому они могут сделать проект общедоступным, контролируя при этом, что в него возвращается.

Запрос на извлечение

Важно помнить, что ваша «развилочная» копия больше не связана напрямую с исходным хранилищем, из которого она была извлечена, и вы не сможете «вытолкнуть» из нее данные обратно в оригиналный репозиторий. Если вы придумали что-то, что считаете ценным для исходного проекта, то можете выполнить так называемый *запрос на извлечение* (pull request), то есть попросить владельца репозитория перенести ваши изменения в исходный «мастер».

Запрос на извлечение вы также можете сделать и к тому репозиторию, к которому у вас есть доступ, а не только к тому, чей форк вы сделали. Например, если вы создали ветку из основной ветки проекта, то можете выполнить запрос на извлечение, чтобы команда, в составе которой вы работаете, проверила ваши результаты и предоставила отзыв, прежде чем объединить ваши изменения со своими. То есть запросы на извлечение могут использоваться для начала обсуждения возможных доработок проекта.

Инструменты и ресурсы Git

Большинство пользователей Git полагают, что лучший способ для работы с Git — это использование командной строки. Как утверждает Дэвид Демари (David Demaree) в книге «Git for Humans»: «Интерфейс командной строки Git — это его родной язык». Он рекомендует вводить команды и наблюдать за происходящим, что и составляет наилучший способ изучения Git. Недостаток командной строки состоит в том, что необходимо изучить все команды Git и, возможно, решать возникающие при этом проблемы. Следующие ресурсы помогут вам освоить новый материал:

- Дэвид Демари (David Demaree), «Git for Humans» — отличная отправная точка для начала изучения Git в режиме командной строки (или в том виде, в котором вам удобно его использовать!);
- руководство «Pro Git» от Скотта Чакона (Scott Chacon) и Бена Штрауба (Ben Straub) (Apress) — доступно в Интернете на бесплатной основе (git-scm.com/book/en/v2);
- «Git Cheat Sheet» от GitHub — список наиболее часто используемых (services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf);
- Руководство «Git Reference Manual» на официальном сайте Git — содержит подробный список команд и функций (git-scm.com/docs).

ФОРКИНГ — ПРИЕМ РАБОТЫ НАД ПРОЕКТАМИ С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

Форкинг чаще всего применяется для работы над проектом с открытым исходным кодом. В коммерческих или личных проектах фиксации обычно напрямую отправляются в репозиторий, с которым работает ваша команда.

Существует также несколько графических приложений Git, доступных для тех, кто предпочитает пиктограммы, кнопки и меню для взаимодействия со своими репозиториями — у каждого свои предпочтения. Мне известны многие разработчики, которые попеременно применяют графическое приложение и приложение Terminal, выбирая тот инструмент, который легче всего позволяет им выполнять текущую задачу. Если вам удобнее работать с графическим инструментом Git, я рекомендую следующие инструменты:

- GitHub Desktop (от GitHub) — бесплатен и доступен для Mac и Windows (desktop.github.com);
- Git Tower 2 (Mac и Windows) — требует оплаты, но зато вы сможете воспользоваться плодами более универсального подхода, и в вашем распоряжении окажется продуманный интерфейс, который, в том числе, предусматривает визуализацию ветвей и слияний (www.git-tower.com).

Встроенную поддержку Git и подключаемых модулей Git/GitHub имеют также многие редакторы кода.

На сайте GitHub.com доступны простые руководства, которые помогут вам выполнить с их помощью процесс установки. Вы сможете создать учетную запись и получить основные навыки работы с GitHub за считанные минуты. Их онлайн-документация превосходна, и у них даже есть канал на YouTube с видеоуроками, предназначеными для начинающих (www.youtube.com/githubguides).

И если уже говорить о GitHub, то для хорошего знакомства со всеми тонкостями его интерфейса я рекомендую книгу «Introducing GitHub: A Non-Technical Guide», написанную Брентом Биром (Brent Beer), которая увидела свет в издательстве O'Reilly.

В общем, когда вы будете готовы начать использовать Git для контроля версий, то найдете всю необходимую поддержку.

ЗАКЛЮЧЕНИЕ

На этом глава, посвященная универсальным инструментам для веб-дизайна, завершается. Мы начали со знакомства с командной строкой, и я привела вам ряд важных аргументов в пользу изучения принципов ее использования. Благодаря инструментам, описанным в этой главе, вы сможете создавать CSS-разметку быстрее, чем раньше, причем делать ее более совместимой с браузерами. Вы сможете воспользоваться преимуществами обработчиков задач и создавать инструменты, которые автоматизируют большую часть повторяющейся нудной работы, с которой вы сталкиваетесь как разработчик. И, наконец, хотя командная строка не обязательна для использования Git, она значительно упростит вам изучение Git и обеспечит супервозможности репо, на которые вы сможете опереться в дальнейшем.

В этой главе мы много говорили о возможностях языка JavaScript. И в *части IV* книги я уступаю место за клавиатурой специалисту в области JavaScript Мэтту Маркусу, который познакомит вас с JavaScript и с его синтаксисом (а также попытается вызвать у вас к нему живой интерес). Я же вернусь к вам в *части V*, чтобы поговорить о веб-изображениях.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Пришло время проверить ваши знания по представленным в этой главе темам. И, как всегда, ответы на вопросы вы найдете в *приложении 1*.

1. Что такое оболочка в компьютерном мире?
2. Почему нужно учиться применять командную строку?
 - a. Это хороший способ манипулировать файлами и папками на вашем компьютере.
 - b. Это хороший способ манипулировать файлами и папками на удаленном сервере.
 - c. Это требуется для применения многих полезных инструментов веб-дизайна.
 - d. Все здесь перечисленное.
3. Что такое *приглашение*?
4. Что именно ожидается, если после приглашения командной строки ввести `mkdir newsite`?
5. Назовите две основные функции процессоров CSS.
6. Назовите одно преимущество изучения Sass.
7. Назовите две функции, для которых можно применять постпроцессор CSS.
8. Что такое задача (по отношению к инструменту сборки/обработчику задач)?
9. Что означает фраза «Grunt просматривает этот файл»?
10. Что делает Git *распределенной* системой контроля версий?
11. Что означает, что файл находится в стадии *подготовки* (в Git)?
12. В чем разница между веткой и развилкой?
13. Почему сначала нужно извлекать, чтобы потом выталкивать?
14. В чем состоит запрос на извлечение?

ЧАСТЬ IV

ПРИМЕНЕНИЕ

JAVASCRIPT

ДЛЯ УПРАВЛЕНИЯ

ПОВЕДЕНИЕМ

ВЕБ-СТРАНИЦ

*Обе главы этой части — 21-я и 22-я —
написаны Мэттом Маркусом (Mat Marquis)*

ГЛАВА 21

ВВЕДЕНИЕ В JAVASCRIPT

В этой главе...

- ▶ *Что может сделать JavaScript, а что ему не под силу*
- ▶ *Переменные и массивы*
- ▶ *Инструкции if/else и циклы*
- ▶ *Собственные и пользовательские функции*
- ▶ *Объекты браузера*
- ▶ *Обработчики событий (Event handlers)*

В этой главе я собираюсь познакомить вас с JavaScript. Полагаю, вы сейчас немного оторопели, и вас можно понять — мы забираемся на суверенную территорию «языка программирования», и это вполне может серьезно напугать... Но, обещаю, вы с этим справитесь!

Мы начнем с обсуждения того, чем является JavaScript, а чем — нет, и рассмотрим некоторые способы его использования. Большая часть главы состоит из введения в синтаксис JavaScript: переменные, функции, операторы, циклы и тому подобное. Будете ли вы программировать к концу главы? Возможно, нет. Но у вас сформируется хотя бы начальное понимание того, что происходит в сценарии, когда вы на него посмотрите. В заключение я расскажу о некоторых способах управления окном браузера и привязки сценариев к действиям пользователя — таким, как нажатие кнопки или отправка формы.

ЧТО ЕСТЬ JAVASCRIPT?

Если вы, работая с книгой, добрались до этого места, то, несомненно, уже знаете, что JavaScript — это язык программирования, который добавляет нашим сайтам интерактивность и нестандартное поведение. Его также можно назвать *языком сценариев на стороне клиента*, и это означает, что он работает на компьютере пользователя, а не на сервере, как делают другие языки веб-программирования, такие, как PHP и Ruby. Это означает также, что JavaScript (и способ его использования) зависят от возможностей и настроек браузера. Он может даже не быть доступным вообще — либо потому что пользователь решил его отключить, либо потому что устройство его не поддерживает, что должны всегда иметь в виду хорошие

разработчики. JavaScript также известен как *динамический и слабо типизированный* язык программирования. Не пытайтесь сейчас вникнуть в эти определения — что все они означают, я объясню чуть позже.

Прежде всего, я хочу развеять бытоущее неверное понимание того, чем является JavaScript.

Чем JavaScript не является?

Надо сразу же отметить путаницу с его названием. Несмотря на наличие в названии «JavaScript» фрагмента «Java», JavaScript с языком Java не имеет ничего общего. Он был создан Брэнданом Айком (Brendan Eich) в Netscape в 1995-м году и первоначально назывался «LiveScript». Но в то время Java был в моде, поэтому в маркетинговых целях «LiveScript» стал «JavaScript». Или просто «JS», если вы захотите продемонстрировать знание компьютерного жаргона, разговаривая о JavaScript.

Репутацию JS также подмочил в некотором роде полукриминальный аспект его использования. Некоторое время назад это название было синонимом всевозможных недобросовестных интернет-махинаций: нежелательных перенаправлений, неприятных всплывающих окон и множества неясных «уязвимостей безопасности», и это лишь некоторые из них. JavaScript позволял неразборчивым в средствах разработчикам пользоваться этим неблаговидным арсеналом, до тех пор пока современные браузеры не обратили внимание на «темную» сторону JavaScript-разработок и не заблокировали ее. Нам не следует винить сам JavaScript за неконвенциальное использование его возможностей. Как гласит мудрость, «с большой силой приходит и большая ответственность». JavaScript всегда предоставлял разработчикам значительную степень контроля над отображением страниц и поведением браузеров, и нам надо уметь ответственно пользоваться этим контролем.

Чем JavaScript является?

Теперь мы знаем, что такое JavaScript: он не имеет отношения к Java, и это не усатый злодей, который прячется в вашем браузере, выкручивает вам руки и извещает вас о «горячих штучках в вашем регионе». Давайте теперь поговорим подробнее о том, что же такое JavaScript.

JavaScript — это легкий, но невероятно мощный язык сценариев. Мы чаще всего сталкиваемся с ним в наших браузерах, но JavaScript проник во все: от встроенных приложений до PDF-файлов и электронных книг. Даже сами веб-серверы могут работать на JavaScript.

Поскольку JavaScript является *динамическим языком программирования*, для запуска программы на этом языке не нужен какой-либо компилятор, интерпретирующий понятный человеку код в код, понятный браузеру. Браузер эффективно читает код так же, как и мы и интерпретирует его «на лету».

JavaScript также является языком *свободно типизированным*. Это означает, что в JavaScript нет необходимости обязательно определять переменные. Чтобы при-

своить переменной значение 5, не нужно программно описывать это значение как число, 5 — это и есть число, и JavaScript так его и распознает.

Вам не обязательно сейчас запоминать эти термины, чтобы начать писать программы на языке JS, и, честно говоря, я тоже сначала не запоминал их. Они приведены здесь лишь для того, чтобы познакомить вас с некоторыми из терминов, с которыми вам придется часто встречаться при изучении JavaScript, и впоследствии они приобретут больший смысл по мере вашего углубления в материал. Кроме того, знание терминологии JS позволит вам поддержать разговор во время следующей коктейльной вечеринки! «О, да, в последнее время я действительно свободно использую динамические языки сценариев». Люди просто молча кивнут вам, что, на мой взгляд, и означает, что вы — «свой парень» в их компании. Но, честно говоря, я — не любитель коктейльных вечеринок.

ECMASCIPT

Язык JavaScript был стандартизирован в 1996-м году Европейской ассоциацией производителей компьютеров (European Computer Manufacturers Association, ECMA), поэтому его иногда называют ECMAScript.

Что может делать JavaScript?

Чаще всего мы используем JavaScript в качестве способа добавить на страницу интерактивность. В то время как «структурным» слоем страницы является наша HTML-разметка, а «презентационный» слой страницы состоит из CSS, третий — «поведенческий» — слой сформирован с помощью JavaScript. Все элементы, атрибуты и текст на веб-странице могут быть доступны с помощью сценариев с использованием DOM (Document Object Model, объектная модель документа), которую мы рассмотрим в главе 22. Мы также можем написать сценарии, реагирующие на ввод пользователя, «на лету» изменяя содержимое страницы, стили CSS или поведение браузера.

Скорее всего, вы видели это в действии, если когда-либо пытались зарегистрироваться на веб-сайте, вводили имя пользователя и сразу получали ответ о том, что введенное вами имя пользователя уже занято кем-то другим (рис. 21.1). Красные

Упс! Произошли некоторые ошибки.

- Это имя пользователя уже используется.
- Подтверждение электронной почты не совпадает.

Username	wilto	Must be at least 4 characters
Email	sample@email.com	
Confirm Email	sampel@email.com	
Password	*****	
Confirm Password	*****	

Рис. 21.1. JavaScript вставляет сообщение, изменяет стили, чтобы сделать ошибки очевидными, и блокирует отправку данных формы. Он также может определить, совпадают ли записи электронной почты, но имя пользователя, скорее всего, будет обнаружено программой на сервере

рамки вокруг полей ввода текста и появление сообщения «Извините, но это имя пользователя уже используется» являются примерами JavaScript, изменяющего содержимое страницы. Блокировка отправки формы — это также пример того, как JavaScript изменяет заданное по умолчанию поведение браузера. В конечном счете, проверка этой информации является задачей сервера, но JavaScript позволяет веб-сайту сделать этот запрос и предложить немедленную обратную связь без необходимости перезагрузки страницы.

Короче говоря, JavaScript позволяет создавать весьма отзывчивые интерфейсы, улучшающие пользовательское восприятие и обеспечивающие динамическую функциональность, не дожидаясь, пока сервер загрузит обновленную страницу. Например, вы можете применять JavaScript для выполнения любого из следующих действий:

- предложение полной поисковой фразы по мере ввода пользователем в поле поиска ее начальных слов. Вы можете увидеть это в действии на сайте **Google.com** (рис. 21.2);

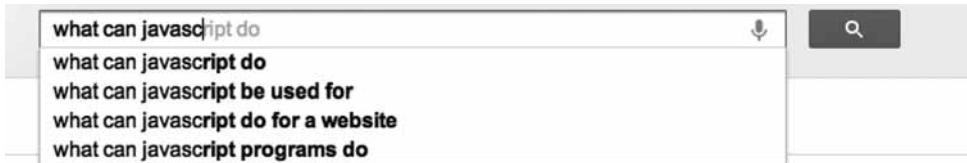


Рис. 21.2. Сайт **Google.com** использует JavaScript для автоматического завершения поисковой фразы по мере ввода ее начальных слов

- запрос контента и информации с сервера и вставка их в текущий документ по мере необходимости без перезагрузки всей страницы — эта возможности обычно называется «Ajax»;
- создание «свертывающейся» области контента, отображающейся и скрываемой по щелчку пользователя на ссылке или заголовке (рис. 21.3);

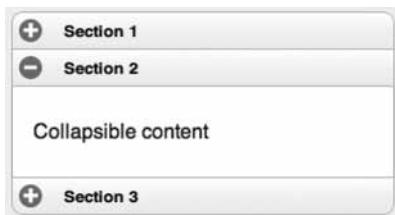


Рис. 21.3. Возможности JavaScript можно использовать для раскрытия и скрытия частей контента

- проверка отдельных функций и возможностей браузеров. Например, можно проверить наличие «событий касания», указывающих, что пользователь взаимодействует со страницей через браузер мобильного устройства, и внести в интерфейс больше сенсорных стилей и методов взаимодействия;

- восполнение недостающих встроенных функций браузера или добавление в старые браузеры некоторые функций, присущих более новым браузерам. Эти виды сценариев обычно называют *прокладками* или *полифилами* (*polyfills*);
- загрузка изображения или контента по щелчку пользователя на уменьшенной версии изображения в пользовательский «лайтбокс», внедренный на страницу с помощью CSS (рис. 21.4).



Рис. 21.4. JavaScript может использоваться для загрузки изображений в галерею в стиле лайтбокса

И этот список далеко не исчерпывающий!

ДОБАВЛЕНИЕ КОДА JAVASCRIPT НА ВЕБ-СТРАНИЦУ

Как и в случае с CSS, можно встроить сценарий прямо в документ или сохранить его во внешнем файле и связать со страницей. Оба метода предусматривают применение элемента `script`.

Вложенный сценарий

Чтобы встроить сценарий на веб-страницу, добавьте код JavaScript в качестве контента элемента `script`:

```
<script>
... Код JavaScript
</script>
```

Внешние сценарии

Этот метод использует атрибут `src`, указывающий на файл сценария (с расширением `js`) с помощью URL-ссылки. В таком случае элемент `script` не имеет контента:

```
<script src="my_script.js"></script>
```

ЕЩЕ О СИНТАКСИСЕ XHTML-ДОКУМЕНТОВ...

Для написанных в более строгом синтаксисе XHTML-документов необходимо идентифицировать контент элемента `script` как раздел `CDATA`, поместив код JavaScript в следующую оболочку:

```
<script type="text/javascript">  
// <![CDATA[  
...Код JavaScript  
// ]]>  
</script>
```

Преимущество внешних сценариев заключается в возможности применять к нескольким страницам один и тот же сценарий (аналогичное преимущество предлагают и внешние таблицы стилей). Недостатком, конечно, является необходимость для каждого внешнего сценария выполнять дополнительный HTTP-запрос сервера, что снижает производительность страницы.

Размещение сценария

Элемент `script` можно разместить в любом месте документа, но наиболее приемлемые места для сценариев находятся в заголовке документа (`head`) и в самом конце раздела `body`. Не рекомендуется разносить его по всему документу, поскольку его будет сложно находить и поддерживать.

Для большинства сценариев конец документа, непосредственно перед тегом `</body>`:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="utf-8">  
</head>  
<body>  
    содержимое страницы ...  
    <script src="script.js"></script>  
</body>  
</html>
```

является наиболее предпочтительным местом размещения сценария, поскольку это позволит браузеру выполнить синтаксический анализ документа и его DOM-структуры. В результате эта информация будет готова и доступна к тому времени, когда она попадет в сценарии, и они смогут выполняться быстрее. Кроме того, загрузка и выполнение сценария блокирует отображение страницы, поэтому перемещение сценария в нижнюю часть страницы улучшит воспринимаемую производительность.

Однако в некоторых случаях вам может понадобиться, чтобы сценарий что-либо выполнял до полной загрузки раздела `body`, и тогда его размещение в разделе `head`

приведет к повышению производительности. Например, Modernizr (инструмент запроса функций, рассматриваемый в главе 19) рекомендует размещать свой сценарий в разделе `head`, чтобы тесты запроса функций могли быть запущены первыми.

АНАТОМИЯ СЦЕНАРИЯ

Книга Дэвида Флэнагана (David Flanagan) «JavaScript: The Definitive Guide» (издательство О’Рейли) не зря имеет объем в 1100 страниц. Причина этого — в богатстве языка JavaScript! В этой главе описанию основных строительных блоков JavaScript, знание которых позволяет понимать суть сценариев, посвящены всего лишь несколько страниц. Многие разработчики научились программированию, подбирая уже существующие сценарии и выполняя их адаптацию под свои нужды. После небольшой подобной практики они начинают создавать и собственные сценарии «с нуля». Распознавание частей сценария — это первый шаг в верном направлении, так давайте выполним его.

Первоначально функциональность JavaScript была, в основном, ограничена общими методами взаимодействия с пользователем. Для обеспечения обратной связи с пользователем мы могли применить несколько встроенных в JavaScript функций (рис. 21.5), таких, как `alert()` — для направления пользователю уведомления и `confirm()` — чтобы пользователь одобрил или отклонил действие. Для запроса пользователя о вводе данных применялась встроенная функция `prompt()`. Несмотря на то что подобные методы применяются и сегодня, они считаются раздражающими, навязчивыми и, согласно общему мнению, весьма неприятными способами взаимодействия с пользователями. С течением времени JavaScript успешно развивался и был обогащен более изящными способами управления поведением веб-страниц, которые формируют вполне приемлемый для пользователей способ общения.

Чтобы воспользоваться этими методами взаимодействия, мы должны сначала понять основную логику, на которой построены сценарии. Она опирается на логические шаблоны, общие для всех типов языков программирования, хотя их синтаксис от языка к языку может различаться. Проводя параллель между языками программирования и разговорными языками, отметим, что хотя словари могут варьироваться от одного языка к другому, большинство их поддерживают одни и те же грамматические правила.



Рис. 21.5. Встроенные функции JavaScript: `alert()` (вверху), `confirm()` (посередине) и `prompt()` (внизу)

К концу этого раздела вы получите представление о переменных, массивах, операторах сравнения, инструкциях `if / else`, циклах, функциях и многом другом. Вы готовы? Тогда вперед.

Основы

Имеется несколько общих синтаксических правил, которые распространяются на весь JavaScript.

Важно знать, что JavaScript *чувствителен к регистру символов* — переменные с именами `myVariable`, `myvariable` и `MYVariable` обрабатываются как три разных объекта.

ЗАПОМНИТЕ!

JavaScript чувствителен к регистру символов.

Кроме того, символы табуляции и пробелы игнорируются, если они не являются частью текстовой строки и не заключены в кавычки. Все символьные пространства, имеющиеся в представленных в этой главе сценариях, предназначены для удобства пользователей — они облегчают процесс чтения кода, но JavaScript их не воспринимает.

Инструкции

Сценарий состоит из серии *инструкций*. Инструкция — это команда, которая говорит браузеру, что ему следует сделать. Вот, например, простая инструкция, которая заставляет браузер отобразить сообщение со словами: «`Thank you.`»:

```
alert( "Thank you." );
```

Точка с запятой в конце инструкции сообщает JavaScript, что это конец команды — так же, как точка заканчивает предложение. Согласно стандарту JavaScript, разрыв строки также воспринимается как конец команды, но рекомендуется заканчивать инструкцию символом «точка с запятой».

Комментарии

Язык JavaScript позволяет оставлять комментарии, которые игнорируются при выполнении сценария, так что вы можете сопровождать свой код напоминаниями и пояснениями. Это особенно полезно, если код впоследствии будет изменяться другим разработчиком, что вполне вероятно.

Имеются два способа добавления комментариев. Для одностroчных комментариев применайте два символа косой черты (`//`) в начале строки. Вы можете разместить односточный комментарий в той же строке, что и инструкцию, — при условии, что комментарий следует за инструкцией. Его не нужно закрывать точкой с запятой, поскольку маркером его закрытия фактически служит разрыв строки:

```
// Это односточный комментарий.
```

Многострочные комментарии используют тот же синтаксис, который применяется и в CSS, — браузером игнорируется все, что содержится между символами `/ * * /`.

Этот синтаксис можно применять для «закомментирования» примечаний и даже фрагментов сценария при устраниении неполадок:

```
/* Это многострочный комментарий.  
Все, что находится между этими наборами символов, будет полностью  
игнорироваться при выполнении сценария.  
Этот вид комментария должен быть закрыт. */
```

Я буду применять однострочные комментарии для добавления небольших пояснений к примерам кода, а также приведенную ранее на рис. 21.5 функцию `alert()`, чтобы быстро показать вам результаты работы.

Переменные

Если вы чем-то похожи на меня, то термин «переменные» должен вызывать у вас кошмарные воспоминания об уроках математики в средней школе. Но сейчас вам этого термина можно более не бояться.

Переменная представляет собой информационный контейнер. Вы даете ему имя, а затем присваиваете значение, которое может быть числом, текстовой строкой, элементом DOM или функцией — чем угодно. В результате мы получаем удобный способ для последующей ссылки на это значение по его имени. Само значение можно изменять и переназначать любым способом, который диктует логика сценария.

Следующее объявление создает переменную с именем `foo` и присваивает ей значение 5:

```
var foo = 5;
```

Мы здесь начали объявление переменной с помощью ключевого слова `var`. Единственный знак равенства (`=`) означает, что ей присваивается значение. Поскольку на этом наша инструкция завершена, заканчиваем строку точкой с запятой. Переменные также могут объявляться и без ключевого слова `var`, влияющего на то, какая часть вашего скрипта будет иметь доступ к информации, которую переменная содержит. Мы обсудим это далее в разд. «Область действия переменных и ключевое слово `var`».

В качестве имени переменной вы можете использовать все что угодно, но убедитесь, что это имя будет впоследствии иметь для вас смысл. Не следует называть переменную просто `data`, поскольку название переменной должно описывать содержащую в ней информацию. В только что приведенном примере название `productName` было бы более приемлемым именем переменной, чем `foo`. Существует несколько правил именования переменных:

- имя переменной должно начинаться с буквы или символа подчеркивания;
- оно может содержать буквы, цифры и символы подчеркивания в любой комбинации;

ЗАПОМНИТЕ!

Переменная — это как бы информационный контейнер.

- оно не может содержать символы пробелов. Вместо них используйте символы подчеркивания или примените «СтильВерблюда» (от англ. CamelCase) — например: `my_variable` или `myVariable`;
- оно не может включать специальные символы: `! . , / \ + * =`.

Вы можете в любой момент изменить значение переменной, повторно указав ее в любом месте сценария. Напомню: JavaScript восприимчив к регистру символов, и имена переменных ведут себя точно так же.

Типы данных

Значения, которые назначаем переменным, относятся к нескольким различным *типам данных*:

- неопределенные типы данных (undefined)* — самый простой из всех типов данных, вероятно, и есть `undefined`. Если объявить переменную, присвоив ей имя, но не присвоив значение, эта переменная будет содержать значение `undefined`:

```
var foo;  
alert(foo); // Откроется диалоговое окно, содержащее "undefined".
```

Скорее всего, вам не часто придется использовать этот тип данных, но о нем следует знать для исправления некоторых ошибок, с которыми вы, вероятно, столкнетесь на начальном этапе изучения JavaScript. Если переменная неожиданно для вас приобретет значение `undefined`, следует ее перепроверить и уточнить, правильно ли она объявлена и нет ли опечатки в ее имени. (Все мы через это проходили.)

- нуль (null)* — аналогично `undefined`, присваивание переменной значения `null` (опять же, с учетом регистра символов) означает: «Определите эту переменную, но не присваивайте ей значения»:

```
var foo = null;  
alert(foo); // Откроется диалоговое окно, содержащее "null".
```

- числовые значения (numbers)* — переменным можно присваивать числовые значения:

```
var foo = 5;  
alert(foo); // Откроется диалоговое окно, содержащее "5".
```

Слово `foo` теперь с точки зрения JavaScript означает то же самое, что и число 5. Поскольку JavaScript является *свободно типизированным*, нет необходимости указывать сценарию, что переменная `foo` должна рассматриваться как *число 5*. Переменная ведет себя как число, поэтому можно поступать с ней точно так же, как и с любым другим числом, используя классические математические операторы: `+`, `-`, `*` и `/` — для сложения, вычитания, умножения и деления соответственно. В следующем примере мы используем знак плюс (`+`) для прибавления значения переменной `foo` к самой себе (`foo + foo`):

```
var foo = 5;  
alert(foo + foo); // Отображается "10".
```

- *строки (strings)* — другим типом данных, которые можно сохранять в переменной, служит *строка (string)*, которая, как правило, представляет собой строку текста. Заключение группы символов в одинарные или двойные кавычки означает, что это — строка:

```
var foo = "five";  
alert( foo ); // Отображается "five"
```

Переменная `foo` теперь трактуется точно так же, как и слово `five`. Это относится к любой комбинации символов: букв, цифр, пробелов и т. п. Если какое-либо значение заключено в кавычки, оно рассматривается в дальнейшем как текстовая строка. Если бы мы заключили число 5 в кавычки и присвоили это переменной, такая переменная перестала бы быть числом — вместо этого речь шла бы о строке текста, содержащей символ «5».

Ранее мы видели знак плюс (+), использованный для суммирования чисел. Если же знак плюс применяется со строками, он объединяет строки в одну длинную строку (этот процесс называется *конкатенацией*):

```
var foo = "bye"  
alert(foo + foo); // Отображается "byebye"
```

Обратите внимание, что отображается в следующем примере, где значение 5 заключено в кавычки и рассматривается из-за этого как строка, а не как число:

```
var foo = "5";  
alert( foo + foo ); // Теперь отобразится "55"
```

В случае конкатенации строки и числа JavaScript предполагает, что число также является строкой, поскольку иначе конкатенация окажется невозможной:

```
var foo = "five";  
var bar = 5;  
alert( foo + bar ); // Отображается "five5"
```

- *булевы значения (booleans)* — вы также можете присвоить переменной значения истина (`true`) или ложь (`false`). Эти значения называются *булевыми*, и они служат основой для логических построений всех видов. В булевых значениях используются встроенные в JavaScript ключевые слова `true` и `false`, поэтому кавычки не нужны:

```
var foo = true; // Теперь переменная "foo" имеет значение true
```

Как и в случае с числами, если в этом примере значение заключить в кавычки, в переменной вместо присущего ей значения `true` (то есть «не ложь») сохранилось бы слово `true`.

В определенном смысле все в JavaScript имеет либо истинное, либо ложное значение. Например, `null`, `undefined`, 0 и пустые строки (« ») — все они, по своей сути, ложны, в то время как все остальные значения, по своей сути, истинны. Такие значения, хотя они и не совпадают с булевыми значениями `true` и `false`, обычно называют истинными» (`truthy`) и «ложными» (`falsy`). Прошу прощения, не я придумал это.

Массивы

Массив — это группа из нескольких значений (называемых *элементами*), которые могут назначаться одной переменной. О значениях в массиве говорят, что они являются *индексированными*, то есть вы можете ссылаться на них по номерам в соответствии с порядком их появления в списке. Первому элементу присваивается индекс 0, второму — 1 и т. д. Мы часто слышим, как всякие умники начинают считать объекты с нуля — это потому, что именно так ведет JavaScript счет объектов, да и многие языки программирования действуют так же. Если об этом помнить, можно избежать многих проблем с программированием.

Итак, допустим, что нашему сценарию нужны все определенные ранее переменные. Мы могли бы определить их три раза и назвать: `foo1`, `foo2` и т. д., или же можно сохранить их в указанном в квадратных скобках (`[]`) массиве:

```
var foo = [5, "five", "5"];
```

Теперь, когда бы вам ни понадобилось получить доступ к любому из этих значений, вы можете извлечь его из массива `foo`, сославшись на его порядковый номер:

```
alert( foo[0] ); // Отображает "5"  
alert( foo[1] ); // Отображает "five"  
alert( foo[2] ); // Опять отображает "5"
```

Операторы сравнения

Теперь, когда мы узнали, как сохранять значения в переменных и массивах, следующий логический шаг — научиться сравнивать эти значения. Существует набор специальных символов, называемых *операторами сравнения*, которые различными способами оценивают и сравнивают значения:

- `==` — равно;
- `!=` — не равно;
- `===` — идентично (равно и одинакового типа данных);
- `!==` — не идентично;
- `>` — больше чем;
- `>=` — больше или равно;
- `<` — меньше чем;
- `<=` — меньше или равно.

Имеет смысл считать все эти определения частью инструкции. Сравнивая значения, мы предполагаем некое утверждение, и цель состоит в том, чтобы получить результат, который либо истинен, либо ложен. То есть, когда мы сравниваем два значения, JavaScript оценивает наше утверждение и возвращает нам логическое значение в зависимости от того, является ли это утверждение истинным или ложным:

```
alert( 5 == 5 ); // Отображается "true"  
alert( 5 != 6 ); // Отображается "true"  
alert( 5 < 1 ); // Отображается "false"
```

Равенство или идентичность?

Важно понять разницу между «равно» (`==`) и «идентично» (`===`). Мы уже знаем, что все значения относятся к какому-либо определенному типу данных. При этом строка «5» и цифра 5 похожи, но это совершенно не одно и то же.

И именно оператор `==` предназначен для выполнения такой проверки:

```
alert( "5" == 5 ); // Отображается "true". Оба равны "5".  
alert( "5" === 5 );  
/* Отображается "false". Оба равны "5", но тип данных  
различный. */  
alert( "5" !== 5 );  
/* Отображается "true", поскольку здесь не один и тот же тип  
данных. */
```

Даже если вам придется прочитать этот раздел пару раз, понимание различий между «равным» и «идентичным» означает, что вы уже начали осваивать особый вид сумасшествия, который нужен программисту. Добро пожаловать! Вы попали в хорошую компанию.

ВНИМАНИЕ!

Будьте внимательны, чтобы не указать один знак равенства, иначе вы переназначите значение одной переменной на значение другой переменной!

Математические операторы

Другой тип оператора — *математический оператор*, выполняющий математические функции над числовыми значениями (и, конечно же, переменными, содержащими числовые значения). Ранее мы уже кратко упомянули простые математические операторы, применяемые для сложения (+), вычитания (-), умножения (*) и деления (/). Имеется также несколько полезных сокращений, о которых следует знать:

- `+=` — добавляет значение к самому себе;
- `++` — увеличивает значение числа (или переменной, содержащей числовое значение) на 1;
- `--` — уменьшает значение числа (или переменной, содержащей числовое значение) на 1.

Инструкции *if/else*

Инструкции *if/else* применяются в JavaScript, чтобы задать вопрос: «истинно или ложно». Они являются основой всей продвинутой логики, которая может быть написана на JavaScript, и они так же просты, как программирование. Да и написаны они почти простым английским языком. Структура условной инструкции имеет следующий вид:

```
if( true ) {  
    // Выполнение каких-либо операций.  
}
```

Такая инструкция сообщает браузеру, «Если это условие выполнено, тогда выполните команды, приведенные в фигурных скобках ({ })». JavaScript не учитывает в коде пробелы, поэтому пробелы по обе стороны от (true) вставлены исключительно для улучшения читабельности кода.

А теперь рассмотрим простой пример использования массива, который объявлен ранее:

```
var foo = [5, "five", "5"];
if( foo[1] === "five" ) {
    alert("Это слово five, написанное простым английским языком.");
}
```

Поскольку здесь выполняется сравнение, JavaScript выдаст значение `true` или `false`. Выделенная строка кода говорит: «истина или ложь: значение переменной `foo` с индексом 1 совпадает со словом «`five`»?»

В нашем случае оповещение `alert` сработает, поскольку переменная `foo` с индексом 1 (вторая в списке, если вы помните) идентична слову «`five`». Это действительно так, и оповещение об этом будет выведено.

Мы также можем проверить, является ли что-либо ложным, используя оператор сравнения `!=`, который означает «не равно»:

```
if( 1 != 2 ) {
    alert("Если вы не увидите это сообщение, значит, у вас
больше проблем, чем у JavaScript.");
    // 1 не равно 2, поэтому мы всегда видим это сообщение.
}
```

ИДИОМАТИЧЕСКИЙ JAVASCRIPT

В сообществе пользователей JavaScript предпринимаются попытки создать руководство по стилю написания кода JavaScript. В документе «Principles of Writing Consistent, Idiomatic JavaScript» утверждается следующее: «Весь код в любом блоке кода должен выглядеть так, как будто его ввел один человек, независимо от количества разработчиков этого кода». Для достижения этой цели группа разработчиков создала `Idiomatic Style Manifesto`, описывающий, каким образом для получения «красивого кода» должны применяться пробелы, разрывы строк, кавычки, функции, переменные и многое другое. Вы можете более подробно познакомиться с этим манифестом на сайте, доступном по адресу: github.com/rwldrn/idiomatic.js/.

Я не очень хорошо разбираюсь в математике, но, насколько я могу судить, 1 никогда не будет равно 2. JavaScript говорит: «Строка “1 не равно 2” является истинным утверждением, поэтому я выполню код в фигурных скобках».

А вот если значение следующей инструкции не окажется равно `true`, код внутри фигурных скобок будет полностью пропущен:

```
if( 1 == 2 ) {
    alert("Если вы не видите это
сообщение, у вас больше проблем,
чем JavaScript.");
    // 1 не равно 2, поэтому этот
код никогда не выполняется.
}
```

Это относится к *if*, но для чего *else*?

Наконец — и я обещаю, что мы почти закончили, — что если мы хотим сделать что-либо одно, если что-то верно, и что-либо другое, если это что-то ложно? Мы могли бы написать два заявления *if*, но это как-то... неуклюже. Вместо этого мы можем просто сказать: *else, do something...else* («еще, сделай что-нибудь ... еще»).

```
var test = "testing";
if( test == "testing" ) {
    alert( "Вы ничего не изменили." );
} else {
    alert( "Вы что-то изменили!" );
}
```

При изменении значения переменной *test* на какое-либо иное значение, отличное от слова *testing*, появится оповещение: «Вы что-то изменили!».

Упражнение 21.1 предоставляет вам возможность самостоятельно создать кусочек кода JavaScript.

УПРАЖНЕНИЕ 21.1. ТРАНСЛЯЦИЯ АНГЛИЙСКОГО НА ЯЗЫК JAVASCRIPT

При выполнении этого небольшого упражнения можно поближе познакомиться с переменными, массивами и операторами *if/else*, переводя написанные на английском языке операторы в строки кода JavaScript. Подсказки для этого упражнения можно найти в *приложении 1*.

1. Создайте переменную под названием *friends* и присвойте ее массиву с четырьмя или пятью именами друзей.
2. Покажите пользователю диалоговое окно, которое отображает третье имя в вашем списке *friends*.
3. Создайте переменную с именем *name* и присвойте ей соответствующее вашему имени строковое значение.
4. Если значение *name* идентично *Jennifer*, покажите пользователю диалоговое окно, в котором отображается сообщение "That's my name too!"
5. Создайте переменную под названием *myVariable* и присвойте ей число из интервала от 1 до 10. Если *myVariable* больше пяти, покажите пользователю диалоговое окно с надписью "upper". Если это не так, отобразите пользователю диалоговое окно с надписью "lower".

Циклы

Бывают ситуации, когда нам требуется обратиться к каждому элементу массива и что-нибудь с ним сделать, но при этом мы не хотим десятки и более раз прописывать это обращение по всему списку элементов. Внимание, читатели, вы собираетесь освоить технику сокрушительной силы — циклы.

Возможно, я несколько преувеличил впечатление от циклов, но они, действительно, невероятно полезны. С помощью тех возможностей JavaScript, которые мы уже

рассмотрели, нам удается хорошо справляться с отдельными переменными, циклы же позволяют нам легко манипулировать огромными наборами данных.

Предположим, у нас есть форма, поля которой заполнены разными значениями. Мы можем воспользоваться DOM для извлечения значения каждого поля, и DOM создаст массив из таких значений (в следующей главе я подробно расскажу вам о том, как DOM это делает). Разумеется, мы можем проверять значения, хранящиеся в этом массиве по одному элементу за раз, но для этого потребуется написать много кода, и трудно даже представить себе, в какой кошмар выльется его обслуживание. Если же мы для проверки каждого значения используем цикл, нам не понадобится изменять наш скрипт, независимо от того, сколько полей добавлено или удалено со страницы. Циклы позволяют нам взаимодействовать с каждым элементом массива, независимо от размера этого массива.

Имеется несколько способов создания цикла, но метод `for` является одним из наиболее популярных. Основная структура цикла `for` имеет следующий вид:

```
for( инициализируемая переменная; проверяемое условие; изменяемое значение; ) {
    // выполнение каких-либо действий
}
```

Рассмотрим пример цикла `for` в действии:

```
for( var i = 0; i <= 2; i++ ) {
    alert( i );
    // Этот цикл вызовет три сообщения – при чтении
    // "0", "1" и "2" соответственно.
}
```

Этот код может показаться немного сложным, поэтому разобьем его на части:

- `for()` — здесь мы вызываем встроенную в JavaScript инструкцию `for()`. Она полагает: «Каждый раз, когда это выполняется (`true`), делайте так». Далее, нужно предоставить этой инструкции некоторую информацию;
- `var i = 0;` — будет создана новая переменная: `i`, значение которой равно нулю. Можно утверждать, что это — переменная, поскольку используется один знак равенства. Чаще в качестве имени переменной используется буква «`i`» (от «`index`»), но можно задать вместо нее любое имя переменной. Это — лишь общее соглашение, а не правило.

Начальное значение мы установили равным 0, потому что остаемся верными привычке вести счет от нуля. Теперь JavaScript приступает к вычислениям;

- `i <= 2;` — при `i <= 2` выполняется инструкция: «Пока меньше или равно 2, продолжайте цикл». Поскольку счет ведется с нуля, это означает, что цикл выполнится три раза;
- `i++` — наконец, запись `i++` означает действие «каждый раз при выполнении этого цикла добавьте единицу к значению `i`» (`++` — это один из сокращенных математических операторов, он уже встречался нам ранее). Без этого шага значение `i` всегда было бы равным нулю, и цикл выполнялся бы вечно! К счастью, современ-

ные браузеры этого не допусят. И если один из трех частей цикла отсутствует, цикл просто не запустится вообще;

- { *сценарий* } — все, что находится внутри этих фигурных скобок, выполняется один раз при каждом запуске цикла — в нашем случае три раза.

Что переменная *i* доступна для использования в коде, выполняющем цикл, мы увидим далее.

Вернемся к примеру «проверить каждый элемент в массиве». Как же следует написать цикл?

```
var items = ["foo", "bar", "baz"]; // Сначала мы создаем массив.  
for( var i = 0; i < items.length; i++ ) {  
    alert( items[i] ); // Отображение сообщения для каждого  
                      // элемента массива.  
}
```

Этот пример отличается от первого цикла двумя ключевыми моментами:

- *items.length* — вместо задания числа, ограничивающего количество выполняемых циклов, мы используем встроенное в JavaScript свойство, позволяющее определить «длину» (*length*) нашего массива, то есть количество включенных в него элементов. Свойство *.length* — это одно из стандартных свойств и методов объекта *Array* в JavaScript. В нашем примере массив состоит из трех элементов, так что цикл будет выполнен три раза;
- *items[i]* — ранее я упоминал, что мы можем использовать переменную *i* внутри цикла? Так и воспользуемся ей для ссылки на каждый индекс массива. Хорошо, что мы начали считать с нуля, — если бы мы установили начальное значение *i* в 1, первый элемент в массиве был бы пропущен. В результате выполнения нашего цикла *for* каждый элемент в массиве (текстовые строки *foo*, *bar* и *baz*) возвращается после каждого цикла и подается в сообщение.

Теперь, независимо от того, насколько большим или маленьким может оказаться массив, цикл будет выполняться столько раз, сколько элементов содержится в массиве, и всегда будет содержать удобную ссылку на каждый элемент массива.

Существуют буквально десятки способов написания цикла в JavaScript, и это — один из наиболее распространенных шаблонов, с которыми вы можете встретиться. Разработчики применяют циклы при выполнения ряда задач — например, таких:

- перебор в цикле списка элементов, имеющихся на странице, и проверка значения каждого из них. Применение стиля к каждому элементу, а также добавление, удаление или изменение атрибутов каждого элемента. Например, можно проверить каждый элемент формы и убедиться перед продолжением работы, что пользователи ввели правильное значение для каждого из них;
- создание в исходном массиве нового массива элементов, которые имеют определенные значения. Для этого мы в цикле сверяем значение каждого элемента исходного массива с заданными значениями и, если значения совпадают, заполняем новый массив только этими элементами. Это превращает цикл в своего рода фильтр.

Функции

По ходу изложения материала я уже познакомил вас с несколькими функциями. Вот пример функции, с которой вы уже не раз встречались:

```
alert("Я все время была функцией!");
```

СТРУКТУРА ФУНКЦИИ:

```
function() {  
}
```

Функция представляет собой часть кода для выполнения задачи, которая не запускается без ссылки на нее или при отсутствии вызова.

Например `alert()` — это встроенная в браузер функция. Блок кода функции запускается только в случае, когда мы явно сообщаем ему об этом. В некотором смысле можно рассматривать функцию как содержащую логику переменную, причем ссылка на эту переменную запускает весь хранящийся в ней код. Функции позволяют повторно использовать свой код всякий раз, когда на него ссылаются, поэтому вам не придется записывать его снова и снова.

Все функции имеют общую структуру (рис. 21.6). За именем функции всегда без пробела следует пара круглых скобок, а за ними — пара фигурных скобок, содержащая код. Круглые скобки иногда содержат используемую функцией дополнительную информацию, называемую *аргументами*. Аргументы — это данные, влияющие на поведение функции. Например, уже хорошо вам известная функция `alert()` воспринимает строку текста как аргумент и использует эту информацию для заполнения выводимого диалогового окна.

```
addNumbers( a, b ) {  
    return a + b;  
}
```

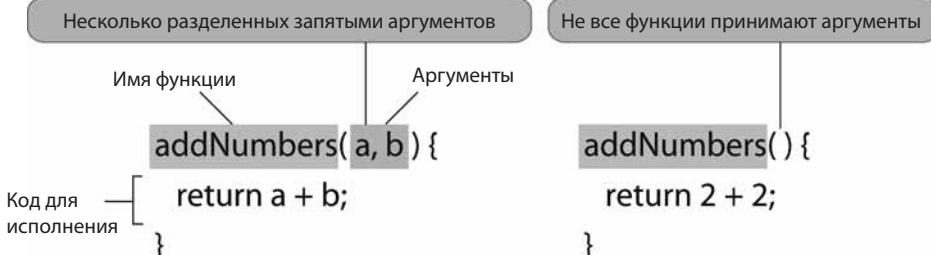


Рис. 21.6. Структура функции

Существуют два вида функций: те, которые поставляются «из коробки» (собственные функции JavaScript), и те, которые вы создаете сами (пользовательские функции). Давайте рассмотрим каждый их вид подробнее.

Собственные функции

В JavaScript встроены сотни предопределенных, в том числе:

- `alert()`, `confirm()` и `prompt()` — эти функции открывают диалоговые окна на уровне браузера;
- `Date()` — возвращает текущие дату и время;

- `parseInt("123")` — эта функция, помимо прочего, обращается к содержащему числа строковому типу данных и превращает его в числовой тип данных. Стока данных передается функции как аргумент;
- `setTimeout(имя_функции, 5000)` — выполняет функцию после задержки. Функция указывается в первом аргументе, а задержка задается в миллисекундах во втором аргументе (в примере 5000 миллисекунд, что равно 5 секундам).

Разумеется, имеются и другие функции. Обратите внимание, что имена функций чувствительны к регистру символов, поэтому обязательно пишите `setTimeout`, а не `SetTimeout`.

Пользовательские функции

Для создания пользовательской функции мы вводим ключевое слово `function`, за которым следует имя функции, за ним — открывающая и закрывающая круглые скобки, а затем открывающая и закрывающая фигурные скобки:

```
function name() {  
    // Код функции.  
}
```

Как в случае с переменными и массивами, имя функции может выбираться произвольно, но с применением тех же правил синтаксиса.

Если вы хотите создать функцию, выводящую некоторое сообщение (код ее здесь несколько избыточен), она может выглядеть так:

```
function foo() {  
    alert("Наша функция только что выполнилась!");  
    // Этот код не выполняется до тех пор, пока мы не вызвали  
    // функцию "foo()"  
}
```

Затем вы можете вызвать эту функцию и выполнить ее код в любом месте сценария, записав следующее:

```
foo(); // Сообщение: "Наша функция только что выполнилась!"
```

Эту функцию можно вызывать в сценарии любое число раз. Это экономит время и позволяет избежать избыточного кодирования.

Аргументы

Наличие функции, которая выполняет в вашем сценарии один и тот же код, вряд ли будет таким уж полезным. Но мы можем «передавать аргументы» (предоставлять данные) собственным и пользовательским функциям, что позволяет применять логику функции к различным наборам данных и в разное время. Чтобы сохранить место для аргументов, создайте в скобках после имени функции при ее определении имя переменной (или последовательность имен, разделенных запятыми).

АРГУМЕНТ

Аргумент — это значение или данные, которые функция использует при запуске.

Пусть нам, например, нужно создать простую функцию оповещения о количестве содержащихся в массиве элементов. Известно, что для получения количества элементов в массиве можно применить свойство `.length`, поэтому нам необходимо определить способ передачи в нашу функцию массива, количество элементов в котором будет вычисляться. Сделаем это, предоставив в качестве аргумента передаваемый массив. В следующем коде создаются новая функция с именем `alertArraySize()` и переменная `arr`, которая резервирует место для аргумента. Затем эта переменная станет доступной внутри функции и сможет содержать любой аргумент, который передается функции при ее вызове:

```
function alertArraySize(arr) {  
    alert(arr.length);  
}
```

При показанном далее вызове этой функции все данные, заключенные в скобки после имени функции (в нашем случае — переменная `test`), передаются в аргумент с помощью заполнителя `arr`. Переменная `test` здесь определяется как массив из пяти элементов. Эта переменная и передается функции, после чего массив `test` подключается к функции и возвращается его длина:

```
var test = [1,2,3,4,5];  
alertArraySize(test); // Выводится "5"
```

Возврат значения

(эта информация для вас новая, но зато невероятно полезная)

Весьма часто функцию используют для вычисления чего-либо, а затем возвращают вам значение, которое вы можете использовать в другом месте своего сценария. Мы могли бы, используя уже полученные знания, сделать это с помощью осознанного применения переменных, но есть гораздо более простой способ.

Ключевое слово `return` внутри функции эффективно превращает эту функцию в переменную с динамическим значением! Намного легче показать это, чем объяснить, так что перейдем к рассмотрению примера:

```
function addNumbers(a,b) {  
    return a + b;  
}
```

Здесь мы имеем функцию, принимающую два аргумента и складывающую их. Вряд ли она была бы хоть как-то полезна, если бы результат всегда оставался внутри нее и его нельзя было бы использовать в другом месте сценария. Но для передачи результата из функции мы применили ключевое слово `return`. Теперь любая ссылка, которую мы сделаем на эту функцию, выдаст нам ее результат в качестве переменной:

```
alert( addNumbers(2,5) ); // Выводится "7"
```

В некотором смысле, функция `addNumbers ()` теперь играет роль переменной, которая содержит динамическое значение — вычисленную нами величину. Если бы мы не использовали внутри нашей функции ключевое слово `return`, приведенный

скрипт `alert` выдал бы сообщение `undefined`, — точно так же, как его выдает переменная, которой мы не дали значение.

Ключевое слово `return` имеет одну особенность. Как только JavaScript видит, что пришло время вернуть значение, функция завершается. Вот соответствующий пример:

```
function bar() {  
    return 3;  
    alert("Мы никогда не увидим это сообщение.");  
}
```

При вызове этой функции обращением к `bar()` сообщение из второй строки никогда не отображается. Функция завершает выполнение, как только приходит время вернуть значение.

Область видимости переменных и ключевое слово `var`

Бывает необходимо, чтобы определенная в функции переменная была доступна в любом месте вашего сценария. В иных ситуациях желательно ограничить ее применение и сделать доступной только для содержащей ее функции. Понятие доступности переменной называется ее *областью видимости*. Переменная, которая может быть использована в любом из сценариев на вашей странице, имеет *глобальную* область видимости, а переменная, доступная только в пределах ее родительской функции, имеет *локальную* область видимости.

Переменные JavaScript используют функции для управления своими областями видимости. Если переменная определена вне функции, она будет иметь глобальную область видимости и доступна для всех сценариев. Когда же вы определяете переменную внутри функции и хотите, чтобы она использовалась только этой функцией, вы можете пометить ее как локальную область, добавив перед именем переменной **ключевое слово `var`**:

```
var foo = "value";
```

Чтобы использовать переменную, определенную внутри функции, в глобальной области видимости, мы при определении этой переменной просто не указываем ключевое слово `var`:

```
foo = "value";
```

В табл. 21.1 представлены данные о зависимости получаемой переменной области видимости от способа ее объявления.

Таблица 21.1. Зависимость получаемой переменной области видимости от способа ее объявления

Переменная	Место нахождения	Область видимости
Наличие ключевого слова <code>var</code>	Вне функции	Глобальная
Наличие ключевого слова <code>var</code>	Внутри функции	Локальная
Отсутствие ключевого слова <code>var</code>	Внутри функции	Глобальная

При определении переменных в функциях вам следует проявлять осторожность, иначе можно получить неожиданные результаты. Рассмотрим следующий фрагмент кода JavaScript:

```
function double( num ){
    total = num + num;
    return total;
}
var total = 10;
var number = double( 20 );
alert( total ); // Выводится 40.
```

Можно было бы ожидать, что, раз вы специально присвоили переменной `total` значение 10, функция `alert(total)` в конце сценария вернет значение 10. Но поскольку мы не охватили переменную `total` в функции ключевым словом `var`, она приобретает глобальную область видимости. Поэтому, хотя переменная `total` установлена в 10, следующий оператор запускает функцию и получает значение для переменной `total`, определенной там. Без ключевого слова `var` переменная «утекла».

Как можно видеть, проблема с глобальными переменными в том, что они становятся общими для всех сценариев на странице. Чем больше переменных попадает в глобальную область, тем больше шансов на появление «коллизий», когда названная в другом месте (и даже в другом сценарии) переменная совпадет с одной из ваших переменных. Это может приводить к непреднамеренному переопределению переменных с неожиданными значениями и, как следствие, к ошибкам в вашем сценарии.

УДЕРЖАНИЕ ПЕРЕМЕННЫХ ВНЕ ГЛОБАЛЬНОЙ ОБЛАСТИ ВИДИМОСТИ

Если нужно, чтобы ни одна переменная не попала в глобальную область видимости, можно весь код JavaScript поместить в следующую оболочку:

```
<script>
(function() {
    // Здесь находится весь код!
}());
<script>
```

Это скромное «карантинное» решение называется IIFE (Immediately Invoked Functional Expression, немедленно вызываемое функциональное выражение), и мы обязаны этим методом и связанным с ним запоминающимся термином Бену Алману (Ben Alman) (benalman.com/news/2010/11/immediately-invoked-functionexpression/).

Запомните, что вы не всегда можете проконтролировать весь код, размещенный на вашей странице. Очень часто на страницах встречается код, написанный третьими лицами, — например:

- сценарии для отображения рекламы;
- сценарии отслеживания пользователей и аналитические сценарии;
- кнопки «Поделиться» в социальных сетях.

Лучше всего во избежание коллизий переменных не рисковать, поэтому, если вы начинаете самостоятельно писать сценарии, задавайте своим переменным локальную область видимости, если это возможно (см. врезку «Удержание переменных вне глобальной области видимости»).

На этом завершим небольшой вводный обзор синтаксиса JavaScript. Это далеко не все, но уже имеется достойная основа

для самостоятельного изучения и интерпретации существующих сценариев. Существует ряд дополнительных, связанных с JavaScript, функций, поэтому сначала рассмотрим их, а затем перейдем к примерам.

ОБЪЕКТ БРАУЗЕРА

Помимо возможности управления элементами на веб-странице, JavaScript также предоставляет вам доступ и возможность манипулировать частями самого окна браузера. Например, можно получать или заменять интернет-адрес (URL) в адресной строке браузера, открывать или закрывать окно браузера.

В JavaScript браузер известен как объект `window`. Этот объект имеет ряд свойств и методов, которые мы можем использовать для взаимодействия с ним. Кстати, известный нам метод `alert()` является одним из стандартных методов объекта браузера. В табл. 21.2 приведены лишь некоторые свойства и методы, которые можно применять с объектом `window`, чтобы дать представление о его возможностях. Полный их список вы найдете в справочнике Window API на сайте MDN Web Docs (developer.mozilla.org/en-US/docs/Web/API/Window).

Таблица 21.2. Свойства и методы браузеров

Свойство/метод	Описание
<code>event</code>	Представляет состояние события
<code>history</code>	Содержит URL-адреса, которые пользователь посетил в окне браузера
<code>location</code>	Предоставляет доступ на чтение/запись к URI в адресной строке
<code>status</code>	Устанавливает или возвращает текст в строке состояния окна
<code>alert()</code>	Отображает окно с указанным сообщением и кнопкой OK
<code>close()</code>	Закрывает текущее окно
<code>confirm()</code>	Отображает диалоговое окно с указанным сообщением, кнопкой OK и кнопкой Cancel
<code>focus()</code>	Устанавливает фокус на текущее окно

СОБЫТИЯ

JavaScript может получать доступ к объектам на странице и в окне браузера, но известно ли вам, что он также «прослушивает» определенные события? *Событие* — это действие, которое можно обнаружить с помощью JavaScript, — например, загрузка документа или щелчок пользователем на элементе, или просто наведение им на него указателя мыши. HTML 4.0 позволил привязывать скрипт к событиям на странице независимо от того, инициировано ли оно пользователем, браузером или другими скриптами. Это известно как *привязка событий*.

ОБРАБОТЧИКИ СОБЫТИЙ

Обработчики событий «прослушивают» определенные действия документа, браузера или пользователя и связывают сценарии с этими действиями.

В сценариях событие идентифицируется *обработчиком события*. Например, обработчик события `onload` запускает сценарий при загрузке документа, а обработчики `onclick` и `onmouseover` запускают сценарий при щелчке мышью пользователя или при наведении мыши на элемент, соответственно. В табл. 21.3 приведены некоторые из наиболее распространенных обработчиков событий. Учтите, что они также восприимчивы к регистру символов.

Таблица 21.3. Распространенные обработчики событий

Обработчик события	Описание события
<code>Onblur</code>	Элемент теряет фокус
<code>Onchange</code>	Содержимое поля формы изменяется
<code>Onclick</code>	Щелчок мышью на объекте
<code>Onerror</code>	Произошла ошибка при загрузке документа или изображения
<code>Onfocus</code>	Элемент получает фокус
<code>Onkeydown</code>	Клавиша на клавиатуре нажата
<code>Onkeypress</code>	Клавиша на клавиатуре нажата или удерживается
<code>Onkeyup</code>	Клавиша на клавиатуре отпускается
<code>Onload</code>	Страница или изображение загружены
<code>Onmousedown</code>	Кнопка мыши нажата
<code>onmousemove</code>	Мышь перемещена
<code>onmouseout</code>	Мышь отодвигается от элемента
<code>onmouseover</code>	Мышь перемещается над элементом
<code>onmouseup</code>	Кнопка мыши отпущена
<code>onsubmit</code>	В форме нажата кнопка отправки

Существуют три широко используемых метода применения обработчиков событий к элементам на страницах:

- как атрибут HTML;
- как метод, прикрепленный к элементу;
- метод `addEventListener()`.

В рассматриваемых далее примерах двух последних подходов используется объект `window`. Любые события, прикрепляемые к объекту `window`, применяются ко всему документу. Мы будем использовать событие `onclick` во всех этих случаях.

В качестве атрибута HTML

Вы можете указать в атрибуте разметки запускаемую функцию, как показано в следующем примере:

```
<body onclick="myFunction();> /* Функция myFunction выполнится, когда пользователь щелкнет внутри "body" */
```

Хотя этот способ прикрепления событий к элементам на странице еще достаточно часто применяется, тем не менее он считается устаревшим. Его следует избегать по той же причине, по какой мы избегаем использования в разметке атрибутов `style` для применения стилей к отдельным элементам. В этом случае стирается грань между семантическим и поведенческим уровнями страниц, что может привести к сложностям при их обслуживании.

В качестве метода

Этот подход к прикреплению событий также устарел, хотя он и сохраняет объекты строго в соответствии с нашими сценариями. Мы можем прикреплять функции с помощью уже встроенных в JavaScript помощников:

```
window.onclick = myFunction; /* Функция myFunction выполнится,  
когда пользователь щелкнет в окне браузера */
```

Мы также можем вместо предопределенной применять анонимную функцию:

```
window.onclick = function() {  
    /* Любой расположенный здесь код выполнится, когда пользователь  
    щелкнет внутри окна браузера */  
};
```

Этот подход имеет преимущество по простоте применения и обслуживания, но имеет весьма существенный недостаток — с его помощью за один раз можно привязывать только одно событие:

```
window.onclick = myFunction;  
window.onclick = myOtherFunction;
```

В этом примере вторая привязка перезаписывает первую, поэтому, когда пользователь щелкнет внутри окна браузера, выполнится только функция `myOtherFunction`, а ссылка на `myFunction` будет проигнорирована.

Применение метода `addEventListener()`

Хотя этот подход, на первый взгляд, немного сложнее и синтаксис его несколько более многословный, именно он позволяет нам сохранять логику в сценариях и выполнять несколько привязок к одному объекту. В следующем примере мы начнем с вызова метода `addEventListener()` для целевого объекта, а затем укажем рассматриваемое событие и функцию, которая выполняется как два аргумента:

```
window.addEventListener("click", myFunction);
```

Обратите внимание, что префикс `on` в обработчике событий в этой записи опущен.

Как и предыдущий метод, метод `addEventListener()` также может применяться с анонимной функцией:

```
window.addEventListener("click", function(e) {  
});
```

Здесь я мог привести только краткое введение в этот метод, поэтому рекомендую вам посетить страницу «`eventTarget.addEventListener`» MDN Web Docs (developer.mozilla.org/en/DOM/element.addEventListener), где вы найдете исчерпывающую информацию по этой теме.

ОБЪЕДИНИЯ ВСЕ ВМЕСТЕ

Вы познакомились со многими важными строительными блоками JavaScript: рассмотрели переменные, типы данных и массивы, узнали об инструкциях `if/else`, циклах и функциях, а также разобрались с объектами браузера и с обработчиками событий. Но это все разрозненные фрагменты информации. Давайте пройдемся по нескольким простым примерам сценариев, чтобы увидеть, как они могут быть собраны воедино.

Пример 1: Повесть о двух аргументах

Вот простая функция, принимающая два аргумента и возвращающая большее из двух значений:

```
greatestOfTwo( first, second ) {
    if( first > second ) {
        return first;
    } else {
        return second;
    }
}
```

Название этой функции — `greatOfTwo`. Она настроена для принятия двух аргументов, которые я здесь назвал `first` (первый) и `second` (второй) по причине отсутствия более содержательных определений. Функция содержит инструкцию `if/else`, которая возвращает первый аргумент, если он больше второго, и возвращает второй, если это не так.

Пример 2: Самое длинное слово

Вот функция, которая принимает в качестве аргумента массив строк и возвращает самую длинную строку этого массива. А если несколько самых длинных строк имеют одинаковую длину, она возвращает первое вхождение одной из них:

```
longestWord( strings ) {
    var longest = strings[0];

    for( i = 1; i < strings.length; i++ ) {
        if ( strings[i].length > longest.length ) {
            longest = strings[i];
        }
    }
    return longest;
}
```

Сначала мы присваиваем функции имя и разрешаем ей иметь один аргумент. Затем устанавливаем самую длинную переменную (`longest`) в качестве начального значения первого элемента массива `strings[0]`. После чего начинаем цикл со значения 1, а не с 0, так как у нас уже имеется первое значение представленного массива. И каждый раз при повторении цикла сравниваем длину текущего элемента в массиве с длиной значения, сохраненного в переменной `longest` (самой длинной). Если текущий элемент массива содержит больше символов, чем текущее значение переменной `longest`, изменяем значение `longest` на этот элемент. Если же нет, ничего не делаем. После завершения цикла возвращаем значение `longest`, которое теперь уже содержит самую длинную строку массива.

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О JAVASCRIPT

После знакомства с основными строительными блоками и несколькими простыми примерами JavaScript, появилось ли у вас желание продвинуться в этой теме дальше? Если да, вот вам дополнительные ресурсы для дальнейшего изучения:

- ресурсы JavaScript на MDN Web Docs (developer.mozilla.org/en-US/docs/Web/JavaScript) — сотрудники MDN Web Docs собрали необходимые учебные пособия, а также подробную документацию по всем компонентам JavaScript. Посетите этот сайт, если вы только начинаете изучение, но он также будет служить вам справочником и при наличии у вас многолетнего опыта;
- «JavaScript for Web Designers» от Мэта Маркуса (Mat Marquis) — в книге я смог привести гораздо больше полезного материала, чем в одной главе, поэтому, если вы заинтересованы в серьезном руководстве для начинающих, эта книга — для вас;
- «Learning JavaScript» от Итана Брауна (Ethan Brown) (O'Reilly) — при более глубоком погружении в JavaScript эта книга выведет вас на следующий уровень.

В следующей главе будет показано, как применять рассмотренные здесь инструменты в контексте веб-дизайна. А пока проверьте свои знания в области JavaScript, выполнив *упражнение 21.2* и ответив на контрольные вопросы.

УПРАЖНЕНИЕ 21.2. ПРОВЕРЬТЕ СВОИ ЗНАНИЯ

При выполнении этого упражнения мы напишем сценарий, который обновляет заголовок страницы в окне браузера с помощью счетчика «новых сообщений». Возможно, вам приходилось сталкиваться с подобным сценарием. Предположим, что со временем результат выполнения этого упражнения станет частью более крупного веб-приложения, причем вам лишь понадобится обновить заголовок страницы с текущим счетчиком «непрочитанных сообщений».

Необходимый для работы документ (`title.html`) я уже создал заранее — он доступен в папке материалов для этой главы на сайте: learningwebdesign.com. Код сценария, который будет получен в результате выполнения этого упражнения, приводится в *приложении 1*.

1. Начните с открытия в окне браузера файла **title.html**. Отобразится пустая страница с заполненным элементом `title`. Посмотрев на верхнюю часть окна браузера, вы увидите надпись «*Million Dollar WebApp*».
2. Откройте этот документ в текстовом редакторе. Вы найдете в нем элемент `script`, содержащий комментарий непосредственно перед закрывающим тегом `</body>`. Удалите этот комментарий.
3. Если вы собираетесь изменить заголовок страницы, сохраните сначала оригинал. Создайте переменную с именем `originalTitle`. Для нее с помощью метода DOM браузер получит заголовок документа `document.title`. Теперь во время загрузки страницы будет доступна сохраненная ссылка на заголовок страницы. Эта переменная должна быть глобальной, поэтому объявите ее вне каких-либо функций:

```
var originalTitle = document.title;
```

4. Теперь надо определить функцию для повторного применения сценария в случае необходимости. Назовите функцию так, чтобы ее имя легко запоминалось и ее можно было бы сразу узнать в коде. На мой взгляд, вполне подойдет имя `showUnreadCount()`, но можно выбрать и другое:

```
var originalTitle = document.title;
function showUnreadCount() {
}
```

5. Следует подумать о том, что именно выполняет эта функция. Она обрабатывает количество непрочитанных сообщений, поэтому ее аргумент — число, представленное в следующем коде как `unread`:

```
var originalTitle = document.title;
function showUnreadCount( unread ) {
}
```

6. Добавьте код, который запускается для этой функции. Желательно, чтобы заголовок документа на странице отображал заголовок документа плюс количество непрочитанных сообщений. Похоже, нашлась работа для конката-нации (+)! Установите `document.title`, чтобы он был (=) всякий раз, когда строка сохраняется для `originalTitle` плюс число в `showUnreadCount`. Как известно, JavaScript объединяет строку и число, как будто они оба пред-ставлены строками:

```
var originalTitle = document.title;
function showUnreadCount( unread ) {
    document.title = originalTitle + unread;
}
```

7. Опробуем сценарий, прежде чем идти дальше. Ниже определения функции и переменной `originalTitle` впишите `showUnreadCount(3);`. Сохраните эту страницу и перезагрузите ее в браузере (рис. 21.7).

```
var originalTitle = document.title;
function showUnreadCount( unread ) {
    document.title = originalTitle + unread;
}
showUnreadCount(3);
```

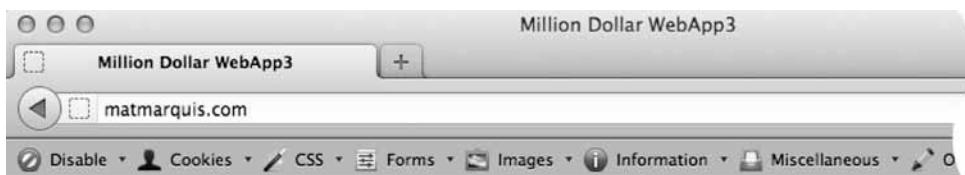


Рис. 21.7. Наш заголовок изменился! Однако есть еще над чем поработать

8. Наш сценарий работает, но заголовок выглядит суховато... К счастью, у нас нет ограничений на количество строк, объединяемых одновременно. Добавьте новые строки для переноса значения счетчика и слов «new messages» (новые сообщения) в скобках (рис. 21.8):

```
var originalTitle = document.title;
function showUnreadCount( unread ) {
    document.title = originalTitle + "(" + unread + " new
messages!)";
}
showUnreadCount(3);
```

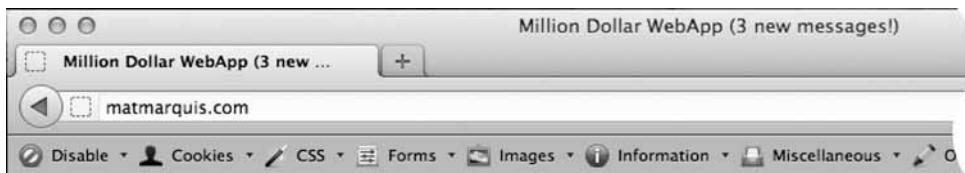


Рис. 21.8. Теперь значительно лучше!

КОНТРОЛЬНЫЕ ВОПРОСЫ

В этой главе мы рассмотрели много нового материала, и теперь пришло время проверить, что именно осталось у вас в памяти. Ответы на вопросы приведены в *приложении 1*.

1. Назовите один положительный и один отрицательный моменты, связанные со ссылками на внешние файлы *.js.
2. Рассмотрите следующий массив:

```
var myArray = [1, "two", 3, "4"]
```

и запишите, что будет сказано в сообщении для каждого из следующих примеров:

- a. alert(myArray[0]);
- b. alert(myArray[0] + myArray[1]);
- c. alert(myArray[2] + myArray[3]);
- d. alert(myArray[2] - myArray[0]);

3. Что будет сказано в каждом из этих сообщений?

- a. var foo = 5;
foo += 5;
alert(foo);
- b. i = 5;
i++;
alert(i);
- c. var foo = 2;
alert(foo + " " + "remaining");
- d. var foo = "Mat";
var bar = "Jennifer";
if(foo.length > bar.length) {
 alert(foo + " is longer.");
} else {
 alert(bar + " is longer.");
}
- e. alert(10 === "10");

4. Напишите, какие действия здесь выполняются:

```
for( var i = 0; i < items.length; i++ ) { }
```

5. В чем состоит потенциальная проблема, связанная с глобальными переменными?

6. Сопоставьте каждый обработчик событий с его триггером:

- | | |
|----------------|---|
| a. Onload | 1. Пользователь завершает форму и нажимает кнопку отправки. |
| b. Onchange | 2. Страница завершает загрузку. |
| c. Onfocus | 3. Указатель находится над ссылкой. |
| d. Onmouseover | 4. Поле для ввода текста выбрано и подготовлено для ввода. |
| e. onsubmit | 5. Пользователь изменяет имя в поле формы. |

ГЛАВА 22

ИСПОЛЬЗОВАНИЕ JAVASCRIPT И DOM

В этой главе...

- ▶ Использование DOM для получения доступа к элементам, атрибутам и контенту и изменения их
- ▶ Использование полифилов для согласованной работы версий браузеров
- ▶ Использование библиотек JavaScript
- ▶ Краткое введение в Ajax

Теперь, когда вы получили представление о языке JavaScript, давайте рассмотрим некоторые способы его применения в современном веб-дизайне. Сначала я расскажу вам о сценариях DOM, которые позволяют нам манипулировать элементами, атрибутами и текстом на странице. Затем познакомлю вас с некоторыми готовыми ресурсами для сценариев JavaScript и DOM, чтобы вам не пришлось искать их самостоятельно. Вы узнаете о *полифилах*, благодаря которым устаревшие браузеры приобретают современные возможности и получают новую функциональность. Я также познакомлю вас с библиотеками JavaScript, которые облегчают жизнь разработчиков, предоставляя им коллекции полифилов и ярлыков для решения их задач.

ВСТРЕЧАЙТЕ DOM

В этой книге несколько раз встречались упоминания про *объектную модель документа* (Document Object Model, DOM), и сейчас настало время уделить этой теме внимание, которого она заслуживает. Модель DOM предоставляет нам возможность доступа к содержимому документа и управления им. Мы обычно используем DOM для HTML, но ее также можно с успехом применять при работе с любым языком XML. И, хотя мы сосредоточены на связи DOM с JavaScript, стоит отметить, что к этой модели могут обращаться и другие языки, такие, как PHP, Ruby, C++ и пр. Несмотря на то что DOM Level 1 была выпущена W3C в 1998-м году, лишь почти пять лет спустя сценарии DOM начали приобретать популярность.

МОДЕЛЬ DOM

Модель DOM позволяет получать доступ к контенту документа и управлять им.

Модель DOM — это программный интерфейс (API) для страниц HTML и XML. Она предоставляет структурированную карту документа, а также набор методов для взаимодействия с содержащимися в нем элементами. По сути, DOM преобразует нашу разметку в понятный JavaScript (и другим языкам) формат. Это заявление звучит весьма прозаично, но основной его смысл в том, что DOM служит схемой расположения всех элементов на странице и позволяет нам что-то *делать* с ними. Мы можем использовать DOM для поиска элементов по их именам или атрибутам, а затем добавлять, изменять или удалять элементы и их содержимое.

Без DOM у JavaScript не было бы никакого смысла для применения к содержимому документа — я имею в виду содержимое документа во *всей* его совокупности. Все — от типа страницы (`doctype`) до каждой отдельной буквы в тексте — можно получить через DOM и манипулировать этим с помощью JavaScript.

Дерево документа

Рассмотрим в качестве примера HTML-разметку несложного документа:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document title</title>
    <meta charset="utf-8">
  </head>
  <body>
    <div>
      <h1>Heading</h1>
      <p>Paragraph text with a <a href="foo.html">link</a> here.</p>
    </div>
    <div>
      <p>More text here.</p>
    </div>
  </body>
</html>
```

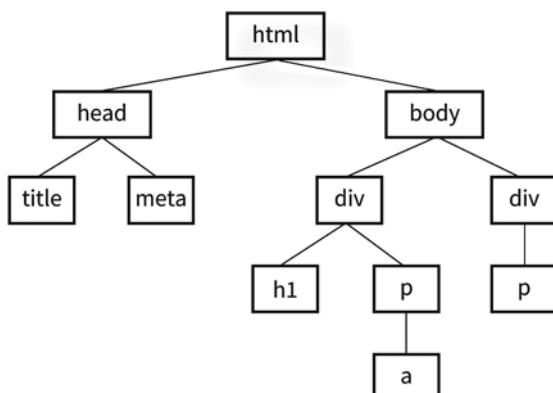


Рис. 22.1. Структура несложного документа

Простой способ представления DOM этого документа в терминах *дерева документа* показан на рис. 22.1. Вы уже видели документы, схематически изображенные таким образом, когда изучали селекторы CSS.

Каждый элемент на странице рассматривается как *узел*. Если представлять DOM как дерево, то каждый узел — это его отдельная ветвь, которая может содержать дополнительные ветви. Но DOM

обеспечивает более глубокий доступ к контенту документа, чем CSS, потому что трактует как узлы фактический контент документа. На рис. 22.2 показана структура первого элемента `p`. Этот элемент, его атрибуты и его контент в дереве узлов DOM и служат узлами.

```
<p>Paragraph text with a <a href="foo.html">link</a> here.</p>
```

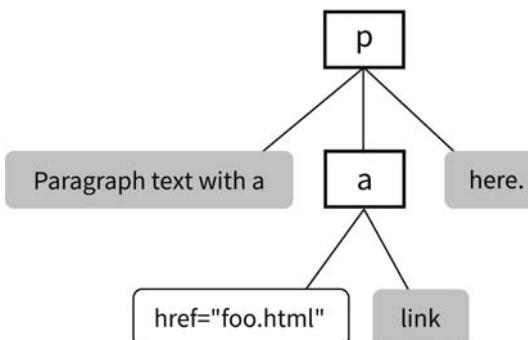


Рис. 22.2. Узлы в первом элементе `p` примера документа

Модель DOM также предоставляет стандартизованный набор методов и функций, с помощью которых JavaScript может взаимодействовать с элементами на нашей странице. Большинство сценариев DOM предусматривают операции чтения и записи в документ.

DOM — ЭТО НАБОР УЗЛОВ:

- узлов элементов
- узлов атрибутов
- текстовых узлов

Есть несколько способов использовать DOM, чтобы найти в документе то, что нам нужно. Давайте рассмотрим некоторые конкретные методы, которые мы можем использовать для доступа к объектам, определенным DOM (мы, JS-люди, называем эту процедуру «сканированием DOM» или «обходом DOM»), а также ряд методов управления этими элементами.

Доступ к узлам DOM

Объект `document` в DOM идентифицирует страницу в целом и чаще всего является отправной точкой для сканирования DOM. Он характеризуется рядом стандартных свойств и методов для доступа к коллекциям элементов. Этот объект может напомнить нам о свойстве `length`, которое рассматривалось в главе 21. Так же как `length` является стандартным свойством всех массивов, объект `document` характеризуется рядом содержащих информацию о документе встроенных свойств. Для нахождения пути к необходимому элементу мы проходим по цепочке через разделенные точками свойства и методы, что и позволяет нам сформировать своего рода маршрут через документ.

Для получения общего представления о том, что это означает, взгляните на наш пример документа: инструкция в примере говорит о необходимости просмотра страницы (`document`), обнаружения элемента, `id` которого имеет значение "beginner", затем — HTML-контента в этом элементе (`innerHTML`) и сохранения этого контента в переменной (`foo`):

```
var foo = document.getElementById("beginner").innerHTML;
```

Поскольку такие цепочки бывают весьма длинными, обычно можно видеть, что каждое свойство или метод выносится на отдельную строку, что дает нам возможность их легко читать:

```
var foo = document
    .getElementById("beginner")
    .innerHTML;
```

При этом не забывайте, что пробелы в JavaScript игнорируются и никак не влияют на результат синтаксического анализа инструкции.

Существуют несколько способов доступа к узлам в документе.

По имени элемента

`getElementsByName()`

Получать доступ к отдельным элементам мы можем с помощью тегов, применяя метод `document.getElementsByTagName()`. Суть работы этого метода заключается в выборке любого элемента или указанных в качестве аргумента элементов.

Например, метод `document.getElementsByTagName("p")` возвращает все имеющиеся на странице абзацы, упакованные в то, что мы называем *коллекцией* или *списком узлов*, в том порядке, как они отображаются в документе по направлению сверху вниз.

ОСТЕРЕГАЙТЕСЬ БЕСКОНЕЧНЫХ ЦИКЛОВ

Список узлов (nodeLists) представляет собой реальную коллекцию. Когда вы организуете в документе цикл по списку узлов — например, при просмотре всех абзацев и добавлении новых, — вы можете угодить в бесконечный цикл. Удивительно, не правда ли?!

Списки узлов очень похожи на массивы. Для получения доступа к определенным абзацам в списке узлов мы ссылаемся на них по индексу, как это происходит при обработке массива:

```
var paragraphs = document.getElementsByTagName("p");
```

На основе этой инструкции `paragraphs[0]` будет служить ссылкой на первый абзац документа, `paragraphs[1]` — на второй абзац и т. д. Если приходится обращаться отдельно к каждому элементу в списке узлов, по одному за один раз, сразу вспоминаем о возможности циклического перемещения по массивам, речь о котором шла ранее. Циклы точно так же работают и со списками узлов:

```
var paragraphs = document.getElementsByTagName("p");
for( var i = 0; i < paragraphs.length; i++ ) {
    // выполнение каких-либо действий
}
```

Теперь мы можем получить доступ к каждому абзацу на странице, ссылаясь внутри цикла на элемент `paragraphs[i]`, так же, как это происходит при работе с массивом, но вместо значений на странице мы получаем элементы.

По значению атрибута *id*

`getElementById()`

Этот метод возвращает отдельный элемент на основе ID этого элемента (значения его атрибута `id`), который предоставляется методу в качестве аргумента. Например, при получении доступа к этому конкретному изображению:

```

```

значение `id` включается в качестве аргумента метода `getElementById()`:

```
var photo = document.getElementById("lead-photo");
```

По значению атрибута *class*

`getElementsByClassName()`

Доступ к узлам в документе можно получить и на основе значения атрибута `class`. Следующая инструкция присваивает переменной `firstColumn` любой элемент, значением `class` которого является «`column-a`», что позволяет легко получить к нему доступ из сценария:

```
var firstColumn = document.getElementsByClassName("column-a");
```

Так же, как и в методе `getElementsByTagName()`, здесь возвращается список узлов, на который можно ссылаться по индексу или применять цикл.

С помощью селектора

`querySelectorAll()`

Метод `querySelectorAll()` позволяет получить доступ к узлам DOM на основе селектора стилей CSS. Синтаксис аргументов в следующих примерах должен быть вам знаком. Это может быть так же просто, как получение доступа к дочерним элементам для определенного элемента:

```
var sidebarPara = document.querySelectorAll(".sidebar p");
```

или так же сложно, как выбрать элемент на основе атрибута:

```
var textInput = document.querySelectorAll("input[type='text']);
```

Метод `querySelectorAll()` возвращает список узлов, аналогичный спискам узлов, возвращаемых методами `getElementsByTagName()` и `getElementsByClassName()`, даже если селектор соответствует только одному элементу.

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ...

Это относительно новый метод доступа к узлам DOM. И хотя метод `getElementsByClassName()` доступен в текущих версиях современных браузеров, он не будет работать в IE8 или более ранних его версиях.

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ...

Метод `querySelectorAll()` не поддерживается браузерами IE7 или более ранних его версий.

Получение доступа к значению атрибута

`getAttribute()`

Как упоминалось ранее, элементы — не единственные объекты, доступ к которым можно получить с помощью модели DOM. Для получения значения атрибута, привязанного к узлу элемента, мы вызываем метод `getAttribute()` с одним аргументом — именем атрибута. Предположим, имеется изображение `stratocaster.jpg`, размеченное следующим образом:

```

```

В следующем примере мы получим доступ к этому конкретному изображению (методом `getElementById()`) и сохраним ссылку на него в переменной (`bigImage`). В этот момент мы можем получить доступ к любому из атрибутов элемента (`alt`, `src` или `id`), указав его в качестве аргумента в методе `getAttribute()`. Поэтому в приводимом примере мы также получим значение атрибута `src` и воспользуемся им как контентом для вывода информационного сообщения. (Непонятно, зачем это нам нужно, но так удобно продемонстрировать этот метод.):

```
var bigImage = document.getElementById("lead-image");
alert( bigImage.getAttribute("src") ); // Сообщение:
"stratocaster.jpg".
```

Работа с узлами

Как только мы получаем доступ к узлу с помощью какого-либо из только что рассмотренных методов, DOM предоставляет нам несколько встроенных методов для управления элементами этого узла, их атрибутами и контентом.

Метод `setAttribute()`

В предыдущем примере мы увидели, как можно получить значение атрибута, но как быть, если мы хотим задать новое значение этого атрибута `src`? Примените метод `setAttribute()`! Этот метод нуждается в двух аргументах: в атрибуте, который необходимо изменить, и в новом значении для этого атрибута.

В следующем примере мы воспользуемся возможностями JavaScript, чтобы поменять изображение, изменяя значение атрибута `src`:

```
var bigImage = document.getElementById("lead-image");
bigImage.setAttribute("src", "lespaul.jpg");
```

Просто представьте себе все, что можно сделать с документом, изменяя значения его атрибутов. В приведенном примере мы поменяли изображение, но можно применить этот же метод для добавления в документ целого ряда изменений:

- обновить атрибуты `checked` флагков и переключателей, используемых для взаимодействия пользователя с контентом где-либо на странице;
- найти элемент `link` для нашего файла таблицы стилей (`*.css`) и указать значение `href` для другой таблицы стилей, тем самым изменения все таблицы стилей страницы;

- обновить атрибут `title` с помощью информации, относящейся к состоянию элемента (например: «этот элемент в настоящее время выбран»).

Метод `innerHTML`

`innerHTML` представляет собой простой метод доступа и внесения изменений в текст и макет внутри элемента. Его поведение отличается от поведения рассмотренных до сих пор методов. Допустим, нам необходим быстрый способ добавления абзаца текста к первому элементу на нашей странице с классом `intro`:

```
var introDiv = document.getElementsByClassName("intro");
introDiv[0].innerHTML = "<p>This is our intro text</p>";
```

Здесь вторая инструкция добавляет контент строки к `introDiv` (элементу со значением `class`, равным «`intro`») как *настоящий «живой» элемент*, поскольку `innerHTML` указывает JavaScript на необходимость воспринять строку между тегами «`<p>`» и «`</p>`» как разметку.

Свойство `style`

Модель DOM также позволяет добавлять, изменять или удалять CSS-стиль из элемента с помощью свойства `style`. Оно работает аналогично назначению стиля с помощью строчного атрибута `style`. При этом отдельные CSS-свойства становятся доступны как атрибуты свойства `style`. Могу поспорить, что вы, опираясь на вновь полученные знания о CSS и DOM, уже можете понять, что делают следующие инструкции:

```
document.getElementById("intro").style.color = "#fff";
document.getElementById("intro").style.backgroundColor = "#f58220";
//orange
```

В JavaScript и DOM названия свойств, которые в CSS пишутся через дефис (такие, как `background-color` и `border-top-width`), записываются в «верблюжьем» стиле (`backgroundColor` и `borderTopWidth`, соответственно), поэтому отсутствие в них символа «-» не принимается за ошибку.

В приведенных примерах свойство `style` задает стили для элементов узла. Оно также может быть использовано для получения значения стиля для применения его в другом месте скрипта.. Следующая инструкция получает цвет фона элемента `#intro` и присваивает его переменной `brandColor`:

```
var brandColor = document.getElementById("intro").style.
backgroundColor;
```

Добавление и удаление элементов

До сих пор мы видели примеры получения и настройки узлов в существующем документе. DOM также позволяет разработчикам самим изменять структуру документа, добавляя и удаляя узлы «на лету». Мы начнем с создания новых узлов, что весьма просто, а затем добавим созданные нами узлы на страницу. Методы, показанные здесь, более реалистичны и точны, чем добавление контента с помощью `innerHTML`. Выполнив все это, мы удалим добавленные узлы.

Метод `createElement()`

Для создания нового элемента примените метод, удачно названный `createElement()`. Он принимает единственный аргумент — элемент, который в результате будет создан:

```
var newDiv = document.createElement("div");
```

Применение этого метода поначалу немного смущает, поскольку новый элемент сразу на странице не отображается. Будучи созданным, этот элемент остается в «пла-вающем» состоянии где-то в глубинах JavaScript, пока мы не добавим его в документ. Можно представить себе такой подход как создание *ссылки* на новый элемент, кото-рый находится лишь в памяти и добавляется на страницу только тогда, когда нам это станет необходимо.

Метод `createTextNode()`

Если мы хотим добавить текст либо в созданный нами элемент, либо в элемент, уже существующий на странице, можно вызвать метод `createTextNode()`. Чтобы им воспользоваться, предоставьте ему в качестве аргумента строку текста, а метод сформирует дружественную DOM версию этого текста, готовую для добавления на страницу:

```
var ourText = document.createTextNode("This is our text.");
```

Как и метод `createElement()`, этот подход создает ссылку на новый текстовый узел, который можно сохранить в переменной и добавить на страницу, когда воз-никнет необходимость.

Метод `appendChild()`

Итак, мы создали новый элемент и новую строку текста, но как сделать их частью документа? Воспользуйтесь методом `appendChild()`. Он принимает единственный аргумент — узел, который необходимо добавить в DOM. Вы вызываете его для существующего элемента, который станет его родителем в структуре документа. Рассмотрим это на примере.

Пусть на странице имеется простой элемент `div` с идентификатором (`id`) «`our-div`»:

```
<div id="our-div"></div>
```

Давайте добавим к элементу `#our-div` абзац с текстом «Hello, world!». И нач-нем мы с создания элемента `p` (`document.createElement()`) и текстового узла для контента, который будет в нем содержаться (`createTextNode()`):

```
var ourDiv = document.getElementById("our-div");
var newParagraph = document.createElement("p");
var copy = document.createTextNode("Hello, world!");
```

Теперь у нас есть наш элемент и некоторый текст, и для объединения этих частей вместе мы можем применить метод `appendChild()`:

```
newParagraph.appendChild( copy );
ourDiv.appendChild( newParagraph );
```

Первая инструкция добавляет содержимое переменной `copy` (это наш текстовый узел «Hello, world!») к новому созданному абзацу (`newParagraph`), после чего этот элемент получает наш контент. А вторая инструкция добавляет созданный абзац `newParagraph` к исходному элементу `div` (`ourDiv`). Теперь элемент `ourDiv` больше в DOM не пустой, и отображается на странице с контентом «Hello, world!».

Вы разобрались, как это работает? Может, привести еще пару примеров?

Метод `insertBefore()`

Метод `insertBefore()`, как можно догадаться, вставляет элемент перед другим элементом. Он принимает два аргумента: первый — это добавляемый узел, а второй — элемент, перед которым он добавляется. Вам также необходимо знать предка элемента, к которому элемент добавляется.

Для примера вставим в следующей разметке новый заголовок перед абзацем:

```
<div id="our-div">
    <p id="our-paragraph">Our paragraph text</p>
</div>
```

Мы начнем с присвоения имен переменных элементу `div` и элементу `p`, который в нем содержится, а затем создадим элемент `h1` и его текстовый узел, после чего соединим их вместе, как было показано в предыдущем примере:

```
var ourDiv = document.getElementById("our-div");
var para = document.getElementById("our-paragraph");

var newHeading = document.createElement("h1");
var headingText = document.createTextNode("A new heading");
newHeading.appendChild(headingText);
// Добавление нашего нового текстового узла в новый заголовок
```

И, наконец, в следующей инструкции метод `insertBefore()` размещает элемент `newHeading` (заголовок `h1`) перед элементом `para` внутри элемента `ourDiv`:

```
ourDiv.insertBefore( newHeading, para );
```

Метод `replaceChild()`

Метод `replaceChild()` заменяет один узел другим и принимает два аргумента: первый аргумент — это новый дочерний элемент (то есть узел, который должен появиться в результате действия), а второй — это заменяемый узел. Как и в случае применения метода `insertBefore()`, нам также необходимо определить родительский элемент, в котором происходит замена. Возьмем для примера следующую простую разметку:

```
<div id="our-div">
    <div id="swap-me"></div>
</div>
```

и заменим в ней изображением элемент `div`, имеющий идентификатор (`id`) «`swapMe`». Начнем мы с создания нового элемента `img` и установки атрибута `src`, равным значению пути к файлу изображения. В завершающей инструкции мы воспользуемся методом `replaceChild()` для размещения элемента `newImg` вместо элемента `swapMe`:

```
var ourDiv = document.getElementById("our-div");
var swapMe = document.getElementById("swap-me");
var newImg = document.createElement("img");
// Создание нового элемента изображения

newImg.setAttribute("src", "path/to/image.jpg");
// Присвоение новому изображению атрибута "src"
ourDiv.replaceChild(newImg, swapMe);
```

Метод `removeChild()`

Перефразирую слова моей матери: «Мы принесли эти элементы в наш мир, и мы сможем их уничтожить». Удаление узла или целой ветви из дерева документов можно реализовать с помощью метода `removeChild()`. Этот метод принимает единственный аргумент — удаляемый узел. Помните, что DOM работает с узлами, а не просто с элементами, поэтому дочерним элементом для элемента может быть не другой элемент, а содержащийся в нем текст (узел).

Как и в случае применения метода `appendChild()`, метод `removeChild()` всегда вызывается для родительского элемента удаляемого элемента, отсюда и название — «*remove child*» (удалить дочерний элемент). Поэтому нам понадобится ссылка как на родительский узел, так и на узел, который необходимо удалить. Возьмем для примера следующий шаблон разметки:

```
<div id="parent">
  <div id="remove-me">
    <p>Pssh, I never liked it here anyway.</p>
  </div>
</div>
```

Наш сценарий может принять следующий вид:

```
var parentDiv = document.getElementById("parent");
var removeMe = document.getElementById("remove-me");
parentDiv.removeChild(removeMe);
// Удаляет элемент div с идентификатором "remove-me" со страницы.
```

Для дополнительного изучения...

Надеюсь, вы получили хорошее представление о том, что такое сценарии DOM. Конечно, здесь вы лишь только начали знакомиться с возможностями DOM, но если пожелаете узнать больше, обязательно обратитесь к книге «DOM Scripting: Web Design with JavaScript and the Document Object Model», 2-е издание), написанной Джереми Китом (Jeremy Keith) и Джеффри Сэмбеллом (Jeffrey Sambells).

ПОЛИФИЛЫ

В этой книге вы уже познакомились со множеством современных технологий: новыми элементами HTML5, «продвинутыми» способами стайлинга, предоставляемыми CSS3, применением JavaScript для управления DOM и многими другими. В идеальном мире все браузеры шли бы с передовыми технологиями нога в ногу, не отставая от них и внедряя их по мере появления (см. врезку «Войны браузеров»). В таком идеальном мире браузеры, которые не поспеваю за технологиями (вспомним IE8), просто вымерли бы как мамонты. К сожалению, это не тот мир, в котором мы живем, и недостатки браузеров остаются головной болью всех веб-разработчиков.

Мне когда-то казалось, что, для того чтобы чему-то научиться, нужно «изобрести колесо». С одной стороны, это действительно помогает в учебе. С другой — вследствие такого подхода наши машины не могут преодолевать каменистые склоны и бурелом... Так что, когда дело доходит до того, чтобы справляться с любыми странностями браузеров, не нужно начинать «с нуля». Многие пользователи уже нашли разумные способы обойти недостатки браузеров и исправить те части кода JavaScript и DOM, которые у этих браузеров вызывают затруднения. При этом для исправления кода JavaScript можно применить сам JavaScript.

ВОЙНЫ БРАУЗЕРОВ

JavaScript пришел в мир в старое и темное время беззакония, когда веб-стандартов не было и в помине, а все основные игроки в мире браузеров бились друг с другом не на жизнь, а на смерть. Вероятно, никого не удивит тот факт, что Netscape и Microsoft внедрили радикально разные версии DOM, и при этом преобладало мнение, что «лучший браузер победит».

Я избавлю вас от мрачных подробностей битвы за звание «хозяина горы» JavaScript, но обе эти конкурирующие реализации настолько отличались друг от друга, что были практически бесполезными, если только вы не захотели бы поддерживать две отдельные базы кода или добавлять на свой сайт предупреждающее сообщение: «лучше прошматривать эту страницу в Internet Explorer/Netscape».

А теперь добавьте эволюцию веб-стандартов! В это непростое время консорциум W3C заложил основы современного стандартизированного DOM, который мы все знаем и любим. К счастью, браузеры Netscape и Microsoft присоединились к движению стандартов, и стандартизованный DOM поддерживался вплоть до Internet Explorer 5 и Netscape Navigator 6. К сожалению, достижения Internet Explorer в этой области после появления версии IE6 застопорились. В результате устаревшие версии IE имеют несколько существенных отличий от современной версии DOM. Впрочем, в Internet Explorer 9 и более поздних версиях эти изменения уже учитываются.

Проблема состоит в том, что ваш проект, вероятно, будет нуждаться в поддержке пользователей более старых версий IE. Это — сильная головная боль, и к ней надо быть готовыми. У нас есть замечательный набор инструментов, таких, как полифили и библиотеки JavaScript, насыщенные вспомогательными функциями, нормализующими странные выкрутасы, с которыми мы обычно сталкиваемся при переходе от браузера к браузеру.

Полифил — это термин, придуманный Реми Шарпом (Remy Sharp) для обозначения «прокладок» JavaScript, упорядочивающих различия в поведении браузеров (remysharp.com/2010/10/08/what-is-a-polyfill). Или, как сказал Пол Айриш (Paul Irish), полифил это:

«“прокладка”, имитирующая будущий API и обеспечивающая резервную функциональность для старых браузеров».

Возможно, эта цитата предполагает путешествия во времени, но основной ее смысл сводится к тому, что мы придааем браузерам некоторые расширенные возможности, которые они изначально не поддерживали, — будь то совершенно новая технология (например, определение физического местоположения пользователя) или исправление имеющихся в работе этих браузеров ошибок.

Как бы там ни было, но к настоящему моменту созданы сотни полифилов, предназначенных для решения конкретных задач и способных, например, заставить старые браузеры распознавать новые элементы HTML5 или селекторы CSS3, причем новые полифили появляются по мере возникновения новых проблем. Я собираюсь рассказать вам о наиболее часто используемых на момент подготовки этой книги полифилах из набора инструментов современного веб-разработчика. Занявшись профессиональным веб-дизайном, вы быстро поймете, насколько они окажутся вам полезными. Впрочем, вы также можете обнаружить, что для браузеров, которые вам приходится поддерживать, некоторые из этих методов не понадобятся.

«Прокладки» (или «заточки») HTML5

Возможно, вы вспомните, что мы слегка коснулись подобной проблематики в главе 5 (см. там врезку «Поддержка HTML5 в Internet Explorer»), но именно теперь, когда у вас уже есть некоторый багаж сведений о JavaScript, ей следует уделить немного больше внимания.

«Прокладки/заточки» (от англ. *shim/shiv*) HTML5 применяются с тем, чтобы браузер Internet Explorer 8 и более ранние его версии могли распознавать новые элементы HTML5 (такие, как `article`, `section` и `nav`) и выполнять их стейлинг.

Существует несколько вариантов HTML5 *shim/shiv*, но все они работают примерно одинаково: сканировать DOM в поисках элементов, которые IE не распознает, и немедленно заменить их тем же элементом, но таким, который IE в DOM способен увидеть. Тогда все написанные для этих элементов стили работают, как положено. Эту технику предложил Сьюрд Виссер (Sjoerd Visscher), и сейчас создано множество вариантов подобных сценариев. Наиболее широко применяемой сегодня является версия HTML5 *shim* Реми Шарпа (Remy Sharp).

Ссылка на сценарий «прокладки» должна размещаться в заголовке документа, чтобы сообщить браузеру Internet Explorer о новых элементах HTML5, до того как он закончит отображение страницы. Эта ссылка вставляется внутрь условного комментария, относящегося к IE, и сценарий «прокладки» запускается только в том случае, если используется версия браузера, более ранняя (`lt`), чем IE9, — другими словами, версия 8 и более ранние:

```
<!--[if lt IE 9]>
<script src="html5shim.js"></script>
<![endif]-->
```

Главная опасность здесь заключается в том, что более старые версии Internet Explorer, в которых JavaScript отключен или недоступен, будут по-прежнему получать элементы без стилей.

Для того чтобы узнать побольше о HTML5 shim/shiv, обратитесь к следующим ресурсам:

- посвященная HTML Shiv статья в Википедии (https://ru.wikipedia.org/wiki/HTML5_Shiv);
- оригинальный пост Реми Шарпа (remysharp.com/2009/01/07/html5-enabling-script).

Selectivizr

Инструмент Selectivizr, созданный Кейт Кларк (Keith Clark), позволяет версиям Internet Explorer 6–8 воспринимать сложные селекторы CSS3, такие, как `:nth-child` и `::first-letter`. Selectivizr использует JavaScript для извлечения и анализа содержимого таблицы стилей и «латания дыр» в тех случаях, когда собственный синтаксический анализатор браузера не справляется.

Инструмент Selectivizr должен применяться совместно с библиотекой JavaScript (речь о ней пойдет в следующем разделе). Ссылка на сценарий инструмента включается в условный комментарий IE после ссылки на файл библиотеки (с расширением `js`), например, таким образом:

```
<script type="text/javascript" src="[JS library]"></script>
<!--[if (gte IE 6)&(lte IE 8)]>
<script type="text/javascript" src="selectivizr.js"></script>
<noscript><link rel="stylesheet" href="[fallback css]" />
</noscript>
<![endif]-->
```

Поскольку мы здесь отказываемся от встроенного в браузер синтаксического CSS-анализатора, вы можете в соответствующих браузерах заметить небольшое снижение производительности.

Для получения более подробной информации об инструменте Selectivizr обратитесь к сайту Selectivizr (selectivizr.com).

НЕ ЗАДЕЙСТВУЙТЕ HTML5 SHIM ПОНАПРАСНУ

Если вам не нужна поддержка браузера IE8 и более ранних его версий, тогда вам не нужен и HTML5 shim.

НЕ ЗАДЕЙСТВУЙТЕ SELECTIVIZR ПОНАПРАСНУ

Если вам не нужна поддержка браузера IE8 и более ранних его версий, тогда вам не нужен и Selectivizr.

Picturefill (полифил для адаптивных изображений)

Полифил Picturefill обеспечивает поддержку для элемента `picture`, атрибутов `srcset` и `sizes`, а также для функций, связанных с добавлением изображений с учетом размера и величины разрешения области просмотра (их также называют

адаптивными изображениями, как отмечается в главе 7). Этот полифил создан Скоттом Джелом (Scott Jehl) из Filament Group и поддерживается группой Picturefill. Загрузить полифил Picturefill и получить информацию о его применении можно на сайте по адресу: scottjehl.github.io/picturefill/.

Для применения полифила Picturefill загрузите его сценарий и добавьте ссылку на него в область head документа:

```
<head>
  <script>
    // Элемент picture HTML5 shiv
    document.createElement( "picture" );
  </script>
  <script src="picturefill.js" async></script>
</head>
```

Первый сценарий здесь создает элемент `picture` для браузеров, которые его не распознают. Второй сценарий вызывает полифил Picturefill, а атрибут `async` сообщает браузеру, что он может загружать полифил Picturefill *асинхронно*, то есть не дожидаясь завершения сценария до загрузки остальной части документа.

Здесь надо отметить, что браузеры без JavaScript, которые также не поддерживают элемент `picture`, будут воспринимать только альтернативный текст для изображения.

БИБЛИОТЕКИ JAVASCRIPT

Продолжая тему «Вам не нужно писать все “с нуля”», зайдемся библиотеками JavaScript. Библиотека JavaScript — это набор заранее написанных функций и методов, которые вы можете использовать в своих сценариях для выполнения общих задач или упрощения сложных.

Существует множество JS-библиотек. Некоторые из них представляют собой большие фреймворки, включающие все наиболее распространенные полифилы, ярлыки и виджеты, которые вам когда-либо понадобятся при создании полноценных веб-приложений Ajax (см. врезку «Что такое Ajax?»). Другие предназначены для решения конкретных задач, таких, как обработка форм, анимация, графики или математические функции. Для опытных профессионалов, пишущих на JavaScript, воспользоваться библиотеками — это потрясающая экономия времени. А начинающие с помощью библиотек смогут решить задачи, которые пока могут быть им и недоступны при имеющихся навыках.

Недостаток библиотек заключается в том, что они обычно содержат все свои функциональные возможности в одном большом файле `*.js`. Из-за этого вы, в конечном итоге, принуждаете своих пользователей загружать большие объемы кода, который может никогда не быть применен. Но разработчики библиотек знают об этом и сделали многие из своих библиотек модульными, а также продолжают прилагать усилия для оптимизации их кода. В некоторых случаях также можно соответствующим образом настроить сценарий и задействовать только необходимые вам его разделы.

ЧТО ТАКОЕ AJAX?

Ajax (иногда пишут AJAX) расшифровывается как Asynchronous JavaScript And XML (Асинхронный JavaScript и XML). При этом фрагмент названия «XML» здесь не столь важен — при использовании Ajax вам не придется применять XML (речь об этом пойдет чуть позже). «Асинхронная» часть — вот что для нас важно.

Традиционно, когда пользователь взаимодействует с веб-страницей таким образом, что необходима доставка данных с сервера, выполнение кода страницы должно приостановиться в ожидании данных, а после получения необходимых данных всю страницу необходимо перезагрузить, чтобы она стала доступной. Это было сделано в интересах пользователей, не имеющих значительного опыта работы.

А вот с помощью Ajax, поскольку страница может получать данные с сервера в фоновом режиме, можно в процессе взаимодействия с пользователем обновлять страницу плавно и в режиме реального времени без необходимости ее перезагрузки. Веб-приложения при этом становятся более похожими на «настоящие» приложения.

Подобный подход применяется при разработке многих современных веб-сайтов, хотя это не всегда заметно. Например, в Твиттере прокрутка страницы вниз приводит к загрузке новых твитов. Они не прописаны в разметке страницы и загружаются динамически, по мере необходимости. Аналогичный подход действует и при поиске изображений в Google — как только вы достигаете нижней части текущей страницы, появляется кнопка, которая позволяет загружать следующие изображения, не покидая текущей страницы.

Термин «Ajax» впервые введен Джесси Джеймсом Гарреттом (Jesse James Garrett) в статье «Ajax: A New Approach to Web Applications». Ajax — не отдельная технология, а, скорее, комбинация HTML, CSS, DOM и JavaScript, включающая объект XMLHttpRequest, позволяющий асинхронно передавать данные. Ajax может для обработки данных использовать XML, но для обмена данными более распространено применение технологии JSON (JavaScript Object Notation, Текстовый формат обмена данными, основанными на JavaScript), основанной на JavaScript и удобочитаемом формате.

Написание веб-приложений с использованием Ajax — это не то, что можно делать прямо сходу, но многие из библиотек JavaScript, рассматриваемых в этой главе, имеют встроенные помощники и методы Ajax, которые позволят вам начать работу со значительно меньшими усилиями.

Библиотека jQuery и другие библиотеки

На момент подготовки этой книги среди библиотек JavaScript доминирующей является jQuery (jquery.com). Скорее всего, если вы используете библиотеку JavaScript, речь идет именно о ней (или, по крайней мере, она будет одной из первых). Созданная в 2005-м году Джоном Резигом (John Resig), jQuery проложила себе дорогу на две трети веб-сайтов. Более того, если сайт вообще использует библиотеку JavaScript, вероятность того, что это jQuery, составляет 97%.

jQuery — это бесплатная библиотека с открытым исходным кодом, использующая синтаксис, облегчающий ее применение, если вы уже освоили CSS, JavaScript и DOM. jQuery можно дополнить библиотекой пользовательского интерфейса jQuery UI, которая добавляет интересные элементы интерфейса, такие, как виджеты календаря,

функциональность перетаскивания, раскрывающиеся «аккордеонные» списки и простые анимационные эффекты. jQuery Mobile — еще одна созданная на основе jQuery библиотека, которая предоставляет элементы пользовательского интерфейса (UI) и полифили, разработанные для учета широкого разнообразия мобильных браузеров и их печально известных особенностей.

Конечно, jQuery — не единственная библиотека JavaScript. Есть и другие библиотеки — например, MooTools (mootools.net), Dojo (dojotoolkit.org) и Prototype (prototypejs.org). Что же касается небольших JS-библиотек, предназначенных для обработки специализированных функций, то они постоянно создаются и устаревают, в связи с чем я рекомендую вам выполнить веб-поиск: *JavaScript libraries for _____* (Библиотеки JavaScript для _____) и посмотреть, что именно доступно. Некоторые категории библиотек включают следующие разделы:

- Формы.
- Анимация.
- Карусели изображений.
- Игры.
- Информационная графика.
- Изображение и трехмерные эффекты для элемента canvas.
- Строковые и математические функции.
- Обработка базы данных.
- Сенсорные жесты.

Как работать с jQuery?

Любую из упомянутых мной библиотек реализовать несложно — вам нужно лишь загрузить соответствующий файл JavaScript (*.js), опубликовать его на своем сервере, указать на него в теге `script`, и готово! Этот файл выполнит всю тяжелую работу, предоставляя в ваше распоряжение заранее написанные функции и сокращенные команды синтаксиса. Включив файл библиотеки, вы можете писать собственные сценарии, действующие встроенные в его структуру функции. Мне, конечно, интересно, как вы на самом деле воспользуетесь всеми этими возможностями (описание которых, к сожалению, в значительной степени выходит за рамки этой главы).

Как член команды jQuery Mobile, я испытываю к этой библиотеке весьма очевидное пристрастие, поэтому в следующих далее примерах буду придерживаться именно jQuery. И дело не только в том, что это самая популярная библиотека, — они сказали, что будут давать мне доллар каждый раз, когда я говорю: «jQuery» ;-).

Загрузите файл jQuery

Для начала работы с jQuery перейдите на сайт **jQuery.com** и щелкните на большой кнопке **Download** (Загрузить), чтобы загрузить копию файла **jquery.js**. У вас там будет выбор между рабочей версией, из которой удалены все лишние пробелы для уменьшения размера файла, и версией для разработки, которую значительно легче читать,

но размер ее файла почти в восемь раз больше файла рабочей версии. Скачать рабочую версию имеет смысл, если вы не собираетесь редактировать ее самостоятельно.

Скопируйте код, вставьте его в новый текстовый документ и сохраните документ с тем же именем файла, которое отображалось в адресной строке окна браузера. На момент подготовки книги последней версией является версия jQuery-3.2.1, а имя рабочей версии — **jquery-3.2.1.min.js** (**min** здесь означает «минимизированный»). Поместите сохраненный файл в каталог с другими файлами вашего сайта. Некоторые разработчики хранят сценарии в каталоге **js** своей организации или просто сохраняют их в корневом каталоге сайта. Куда бы вы ни решили поместить этот файл, обязательно запишите путь к нему, поскольку этот путь понадобится вам при разметке.

Добавьте файл jQuery к своему документу

Подключите к своему документу сценарий jQuery, так же, как вы включаете в документ любой другой сценарий, — с помощью элемента `script`:

```
<script src="path to your js / jquery-3.2.1.min.js"></script>
```

Чаще всего это делается именно так. Однако стоит упомянуть и об имеющейся альтернативе. Если вы не желаете сами публиковать свой файл библиотеки, можно указать одну из его общедоступных версий и использовать уже ее. Одним из преимуществ этого метода является то, что файл кэшируется браузером, и вполне вероятно, что некоторые браузеры ваших пользователей уже имеют его копию. На странице загрузки jQuery приведено несколько параметров, включая следующую ссылку на код на сервере Google:

```
<script src="https://ajax.googleapis.com/ajax/libs/
jquery/3.2.1/ →
jquery.min.js"></script>
```

Просто скопируйте этот код в точности так, как он здесь показан, вставьте его в раздел `head` документа или перед тегом `</body>` и получите jQuery!

Будьте готовы!

Вы не хотите запускать сценарии, до того как документ и DOM будут к ним готовы? Отлично, у jQuery имеется оператор, известный как *ready event* (событие готовности), который проверяет документ и ожидает, когда он будет подготовлен для работы с ним. Не все сценарии требуют этого (например, если вы только отправляете браузеру сообщение), но, если вы работаете с DOM, неплохо было бы начать с установки окружения для ваших сценариев, включив эту функцию в свой пользовательский элемент `script` или файл `*.js`:

```
<script src="path to your js / jquery-3.2.1.min.js"></script>
<script>
$(document).ready(function(){
    // Ваш код
});
</script>
```

Написание сценария с помощью jQuery

Выполнив указанную настройку библиотеки, можно приступать к написанию собственных сценариев с помощью jQuery. Возможности, которые предлагает jQuery, делятся на две основные категории:

- огромный набор встроенных сценариев по обнаружению функций и полифилов;
- более короткий и интуитивно понятный синтаксис для указания элементов (jQuery's *selector engine*, селекторный движок jQuery).

Проработав предыдущий раздел, вы должны были получить приемлемое представление о том, что делают полифили, поэтому давайте сейчас посмотрим, что нам дает этот селекторный движок.

Одна из задач, выполняемая jQuery, — перемещение по DOM, поскольку вы можете применять для этого синтаксис селекторов, который вы изучали применительно к CSS. Далее приводится пример получения элемента по его значению *id* без участия библиотеки:

```
var paragraph = document.getElementById( "status" );
```

Эта инструкция находит элемент с ID "status" и сохраняет ссылку на него в переменной (paragraph). Не слишком ли длинная запись для решения такой простой задачи?! Можете себе представить, насколько все становится избыточным, если вы получаете доступ ко множеству элементов на странице. Теперь же, когда в игру вступила библиотека jQuery, можно применить следующее сокращенное выражение:

```
var paragraph = $("#status");
```

Действительно, это селектор *id*, о котором вы узнали и который полюбили при изучении CSS. Однако не будем останавливаться на достигнутом. Любой селектор, который применяется в CSS, станет работать и в рамках этой специальной вспомогательной функции:

- вы хотите найти все с помощью класса *header*? Примените `$(".header")`;
- с помощью имени элемента? Уверенно: `$("div")`;
- каждый подзаголовок во врезке? Проще простого: `$("#sidebar .sub")`;
- вы можете настроить выборку элементов на основе значения атрибутов: `$("[href='http://google.com']")`;

Но не будем зацикливаться на селекторах. Для сканирования DOM вы можете применять огромное количество вспомогательных функций, встроенных в jQuery и подобные ей библиотеки, — пауки типа Spider-men, Spider-persons и Web-slingers.

jQuery также позволяет связывать вместе объекты, чтобы они указывали на такие объекты, указать на которые не способен даже CSS (например, на родительский

элемент элемента). Допустим, имеется абзац и необходимо добавить `class` к родительскому элементу этого абзаца. Мы не всегда знаем, что это был за родительский элемент, поэтому не можем напрямую указать на этот родительский элемент. Однако, чтобы добраться до этого родительского элемента в jQuery, можно применить объект `parent()`:

```
$( "p.error" ).parent().addClass( "error-dialog" );
```

Еще одним важным преимуществом такой записи является легкость ее прочтения с первого взгляда: «найдите любой абзац(ы) с классом `error` и добавьте класс «`error-dialog`» к их родителю (ям)».

Как быть, если я не знаю, как писать сценарии?

Для изучения JavaScript требуется время, и, вполне вероятно, что вы не сразу же сможете самостоятельно писать сценарии. Не озабочивайтесь этим. Выполните поиск в Интернете того, что вам нужно (например, карусель изображений jQuery или аккордеонный список jQuery), и велика вероятность, что вы найдете множество созданных другими пользователями сценариев, которыми они поделились со всеми, кого это интересует, приложив также комплект документации по их применению. То что jQuery использует селекторный синтаксис, очень похожий на CSS, упрощает настройку чужих сценариев jQuery для применения с вашей собственной разметкой.

ПРОМЕЖУТОЧНЫЙ ФИНИШ

В этих двух главах мы прошли большой путь от знакомства с переменными до манипулирования DOM и использования библиотек JavaScript. Но, даже учитывая все, нами здесь рассмотренное, мы только лишь приступили к изучению того, что может сделать JavaScript.

Когда вам при просмотре какого-нибудь веб-сайта покажется, что он выполняет что-либо хорошо, откройте его исходный код в окне своего браузера и просмотрите код JavaScript. Можно многому научиться, читая или разбирая чужой код. И помните, что с помощью JavaScript ничего нельзя сломать, чего невозможно было бы починить несколькими нажатиями клавиши `<Delete>`.

Более того, JavaScript поддерживает целое сообщество увлеченных разработчиков, которые стремятся как учиться, так и учить. Ищите единомышленников и делитесь тем, что вы узнали на этом пути. Если вы затрудняетесь при разрешении сложной задачи, не стесняйтесь обращаться за помощью и задавать вопросы. Редко встречаются проблемы, с которыми никто не сталкивался, а сообщество разработчиков открытого исходного кода всегда с радостью делится своими знаниями. Об этом и шла речь в этих двух главах.

КОНТРОЛЬНЫЕ ВОПРОСЫ

А теперь ответьте на следующие вопросы, чтобы проверить свои знания. Если нужна помощь, обратитесь к *приложению 1* за ответами.

1. Комбинацией каких технологий является Ajax?

2. Что тут выполняется?

```
document.getElementById("main")
```

3. Что тут выполняется?

```
document.getElementById("main").getElementsByTagName("section");
```

4. Что тут выполняется?

```
document.body.style.backgroundColor = "papayawhip"
```

5. Что делается тут (Это немного сложнее, поскольку имеются вложенные функции, но вы постарайтесь собрать все вместе.)?

```
document
    .getElementById("main")
    .appendChild(
        document.createElement("p")
        .appendChild(
            document.createTextNode("Hey, I'm walking
here!"))
    )
);
```

6. В чем преимущество использования библиотеки JavaScript — такой, как jQuery?

- a. Доступ к упакованной коллекции полифилов
- b. Возможно, более короткий синтаксис
- c. Упрощенная поддержка Ajax
- d. Все указанные здесь варианты вместе

ЧАСТЬ V

ИЗОБРАЖЕНИЯ

ДЛЯ ВСЕМИРНОЙ

ПАУТИНЫ

ГЛАВА 23

ВЕБ-ГРАФИКА: ОСНОВНЫЕ ПОНЯТИЯ

В этой главе...

- ▶ Способы получения веб-изображений
- ▶ Форматы PNG, JPEG, GIF и WebP
- ▶ Разрешение изображения и экрана
- ▶ Стратегия создания веб-изображений
- ▶ Способы создания фавиконов

Если вы не собираетесь создавать исключительно текстовые сайты, вам понадобятся изображения. Для многих из вас это может означать, что вы впервые обратитесь к программе редактирования изображений и приобретете некоторые базовые навыки по созданию графики. Если вы опытный дизайнер, возможно, вам придется приспособить свой стиль и методы работы, чтобы создавать графику, пригодную для сайтов.

В главе 7 мы познакомились с основами встраивания изображений в HTML-документ, в том числе разобрались в том, чем растровые графические форматы отличаются от векторного формата SVG. В этой главе рассматриваются основы собственно изображений. Мы начнем с обзора источников изображений и познакомимся с форматами файлов, пригодными для веб-графики, — это поможет вам решить, какой из них использовать. Мы рассмотрим разрешение изображений и узнаем, как оно связано с разрешением экранов, на которых они выводятся, включая дисплеи высокой плотности. Я также отвечу на ряд вопросов, которые помогут вам сформировать стратегию создания изображений для вашего сайта. А в завершение главы я приведу краткое руководство по созданию и работе с пиктограммами сайтов (фавиконами).

ИСТОЧНИКИ ИЗОБРАЖЕНИЙ

Чтобы воспользоваться изображением, необходимо его *иметь*, поэтому, прежде чем перейти к рассмотрению особенностей форматов файлов, давайте сначала познакомимся с некоторыми способами получения изображений. Для этого есть много вариантов: от сканирования, фотосъемки или рисования их самостоятельно

до использования доступных в Интернете фотографий и рисунков, а можно также просто нанять специалиста, который создаст нужные вам изображения.

Создание собственных изображений

В большинстве случаев наиболее экономически эффективный способ получить изображения для своего сайта — это создать собственное изображение «с нуля». Дополнительное преимущество этого способа состоит в том, что вы будете обладать исключительными правами на использование такого изображения (мы обязательно вскоре рассмотрим вопросы, связанные с авторскими правами).

Итак, дизайнеры имеют возможность создавать изображения с помощью сканеров, фотокамер или графических программ:

- *фотографии* — вы можете запечатлеть на фото все краски окружающего мира и передать его в программу редактирования изображений. В зависимости от требуемого типа изображения вы можете получить фотографии достаточно хорошего качества даже с помощью камеры смартфона. Имейте при этом в виду, что всегда полезно создавать версии изображений с высоким разрешением, при необходимости сохраняя их копии с меньшим разрешением;
- *электронная иллюстрация* — если у вас есть соответствующие навыки, вы можете самостоятельно создавать изображения с помощью приложений, предназначенных для рисования иллюстраций или редактирования фотографий. У каждого дизайнера имеются любимые инструменты и приемы. Для создания логотипов и штриховых рисунков я рекомендую начать с графических векторных программ, таких, как Adobe Illustrator, CorelDRAW или Sketch, сохранив полученные изображения в формате, пригодном для размещения в Сети;
- *сканирование* — это отличный способ получения изображений. Сканировать можно практически все — от плоских рисунков до небольших трехмерных объектов. Однако не поддавайтесь соблазну сканировать и использовать удачные, с вашей точки зрения, чужие изображения. Скорее всего, большинство найденных вами исходных изображений защищены авторским правом и не могут использоваться без разрешения автора, даже после того как вы их измените. Дополнительные советы по технике сканирования можно найти во врезке «*Сканирование изображений*».

Доступные коллекции фотографий и иллюстраций

Если вы не уверены в своих дизайнерских навыках или же просто не хотите тратить на создание изображений много времени и сил, имейте в виду, что имеется множество доступных для приобретения или распространяемых бесплатно коллекций готовых фотографий, иллюстраций, кнопок, анимации и текстур. Такие фотографии и иллюстрации, как правило, делятся на две большие категории: изображения с управляемыми правами и изображения без лицензионных платежей.

Изображения с управляемыми правами

Изображения с управляемыми правами — это те изображения, правообладатель которых (или представляющая его компания) определяет, кто может их воспроизвести. Чтобы использовать изображение с управляемыми правами, вам необходимо приобрести лицензию на его воспроизведение выбранным способом и на определенный период времени. Одним из преимуществ лицензирования изображений является получение эксклюзивных прав на изображение, распространяемое с помощью определенного носителя (например, в Интернете) или в определенном секторе бизнеса (например, в сфере здравоохранения или банковской сферы). Вы также точно знаете, что источник изображения проверен (то есть оно не краденое).

С другой стороны, изображения с управляемыми правами весьма дорого, и в зависимости от приобретаемых по лицензии прав использования цена за одно изображение может составлять несколько тысяч долларов. Если вам не нужны эксклюзивные права и вы собираетесь публиковать изображение только в Интернете, стоимость лицензии, скорее всего, составит — в зависимости от источника — не более нескольких сотен долларов.

Крупнейшим фондом изображений с управляемыми правами, большинство конкурентов которого за последние годы были им поглощены, является сайт Getty Images (gettyimages.com).

Далее мы рассмотрим возможности использования изображений, свободных от лицензионных платежей.

СКАНИРОВАНИЕ ИЗОБРАЖЕНИЙ

При сканировании источников изображений, предназначенных для использования в Интернете, воспользуйтесь следующими советами, которые помогут вам создавать изображения лучшего качества:

- поскольку при уменьшении размера легче поддерживать качество изображения, чем при его увеличении, сканируйте изображение с большими размерами, чем действительно нужно. Это даст вам дополнительные возможности для изменения размеров изображения в дальнейшем. Вопросы, связанные с размерами изображений, подробно рассматриваются в разд. «Размер и разрешение изображения» далее в этой главе;
- сканируйте черно-белые изображения в режиме градаций серого (8 битов), а не в режиме черно-белого (1-битовые или растровые изображения). Это позволит вам вносить корректизы в серые области среднего тона изображения после его подгонки в окончательные размеры и разрешение. Если вам действительно нужно только черно-белое изображение, выполните соответствующее преобразование на последнем этапе обработки;
- если вы сканируете изображение из печатного источника, необходимо исключить полученный в процессе печати точечный шаблон. Для этого лучше всего применить к изображению небольшое размытие (в Photoshop используйте фильтр размытия по Гауссу), изменить размер изображения, несколько его уменьшив, а затем применить фильтр повышения резкости. Это позволит вам избавиться от досадных точек. Ну и, конечно, убедитесь в том, что у вас есть права на использование печатного изображения.

Изображения без лицензионных платежей

Если вы не располагаете четырехзначным или хотя бы трехзначным бюджетом на приобретение изображений, рассмотрите возможность использования художественных изображений, за которые не нужно платить лицензионный сбор. Художественные произведения, не требующие лицензионных платежей, доступны



Рис. 23.1. Фотография блаженствующего кенгуру, полученная без лицензионных платежей на сайте [Gettyimages.com](http://gettyimages.com)

на условиях единовременной оплаты, после чего их можно публиковать без ограничений, но вы не сможете проконтролировать тех, кто еще их также использует. Изображения, свободные от лицензионных платежей, предлагаются первоклассными профессиональными фондами изображений — такими, как Getty Images (Gettyimages.com). У них, например, всего за 50 долларов можно купить изображение блаженствующего кенгуру, представленное на рис. 23.1. Подобные изображения можно приобрести и за меньшую сумму, а иногда и получить бесплатно.

Один из моих любимых источников — iStockPhoto (istockphoto.com), где хранится огромная коллекция изображений, цена которых начинается от 10 долларов США за штуку. Вы можете покупать изображения по одному или же приобрести план подписки.

Лицензия Creative Commons

Другой способ получения бесплатных изображений — обращение к фотографиям и рисункам, выпущенным художниками под лицензией Creative Commons.

ЛИЦЕНЗИИ CREATIVE COMMONS

Дополнительные сведения о лицензиях Creative Commons можно получить на сайте: creativecommons.org/licenses/.

Существует несколько типов лицензий Creative Commons, поэтому обязательно ознакомьтесь с их условиями. Одни художники предлагают свои работы бесплатно для использования на ваше усмотрение, другие — просят выразить

им уважение, разместив указание на их авторство, а трети — ограничивают использование изображения некоммерческими целями.

Существует много ресурсов для изображений, выпущенных по лицензии Creative Commons, но я рекомендую для начала обратить внимание на следующие три:

- Flickr Creative Commons (www.flickr.comcreativecommons) — при поиске фотографий, выпущенных на условиях лицензии Creative Commons, я первым делом обращаюсь к сервису обмена фотографиями Flickr. Качество фотографий, доступных на этом ресурсе, варьируется, но обычно я нахожу то, что мне нужно (например, фотографию красной панды, показанную на рис. 16.28), на условиях указания на авторство;

- Unsplash (unsplash.com) — сервис Unsplash предоставляет бесплатные фотографии высокого качества, которые «дарятся самым щедрым в мире обществом фотографов». Я использую в этой книге взятые отсюда фотографии для иллюстрации кулинарных продуктов;
- Wikimedia Commons (commons.wikimedia.org/wiki/Main_Page) — этот родственный Википедии сайт представляет собой обширный ресурс, содержащий миллионы фотографий, а также других медиафайлов, распространяемых на условиях лицензии Creative Commons. Эти фотографии предоставлены сообществом и доступны для использования на бесплатной основе.

Коллекции клипов и пиктограмм

Коллекция клипов — эта коллекция распространяемых без лицензионных платежей иллюстраций, анимации, кнопок и других элементов дизайна, которые можно копировать и вставлять на свои страницы с самыми разными целями. В Интернете имеется ряд ресурсов, и некоторые из них предоставляют такие графические объекты на бесплатной основе, хотя вам, возможно, будет досаждать множество всплывающих окон с рекламой. Ряд ресурсов взимают членский взнос от 10 до 200 долларов в год. Недостаток этих источников в том, что многие их изображения имеют низкое качество или грешат недостатками, связанными с особенностями восприятия фотографов. Впрочем, вот пара сайтов, с которых можно начать:

- Clipart.com (www.clipart.com) — этот сервис взимает членский взнос, при этом хорошо организован и, как правило, предоставляет более качественные изображения по сравнению с бесплатными сайтами;
- #1 Free Clip Art (www.1clipart.com) — еще один бесплатный сайт, содержащий коллекции вполне приемлемых клипов.

В Интернете также несложно найти пиктограммы (значки) для веб-страниц и приложений как бесплатно, так и за небольшую цену — достаточно лишь провести поиск по ключевым *free icons* (бесплатные пиктограммы). Далее приведены два ресурса, которыми можно для начала воспользоваться:

- The Noun Project (thenounproject.com) — этот ресурс коллекционирует и организует собранные отовсюду классические одноцветные пиктограммы. На бесплатной основе здесь доступны десятки коллекций, а годовая подписка предоставит доступ ко всему богатству ресурса;
- Icon Finder (www.iconfinder.com) — хороший ресурс, содержащий относящиеся ко всем стилям полноцветные пиктограммы. Некоторые из них бесплатны, но большинство доступны на условиях ежемесячного тарифного плана. Обязательно ознакомьтесь с условиями лицензии Creative Commons, которые существенно зависят от выбранной коллекции пиктограмм.

Найдите дизайнера

Поиск и создание пользовательских изображений требуют времени и определенных способностей. Если у вас больше денег, чем этих ресурсов, рассмотрите возможность найма графического дизайнера, фотографа или иллюстратора, который бы

создал нужные для вашего сайта изображения. Преимущество найма профессио-нала заключается в том, что вы получаете изображения, адаптированные к вашему контенту или бренду, а не просто какие-либо стандартные. Так что, получив в свое распоряжение высококачественные оригинальные изображения, вы потом сможете воспользоваться навыками, выработанными при изучении этой книги, для создания их веб-версий.

ЗНАКОМСТВО С ФОРМАТАМИ ИЗОБРАЖЕНИЙ

Подобрав нужные изображения, необходимо перевести их в формат, который будет работать на веб-странице. В мире существует множество форматов графических файлов. Например, работая под Windows, вы можете использовать графику BMP, а если занимаетесь дизайном для полиграфии, можно воспользоваться изображениями в форматах TIFF и EPS. Растворные изображения (на основе пикселов), способ-

ФОРМАТ WebP

На момент подготовки этой книги формат WebP считался достаточно новым, и его поддерживали лишь немногие браузеры и графические программы. Тем не менее я включила его в свое описание, поскольку именно этому формату предстоит стать одним из самых распространенных по мере улучшения его поддержки.

графика), рассмотренный нами с точки зрения его добавления в разметку в главе 7. Формат SVG превосходит в том смысле, что генерируется текстовым XML-файлом. И он действительно настолько уникален, что ему посвящена отдельная глава (см.

чтобы можно было отображаться в Интернете, можно сохранять в следующих форматах: JPEG («джи-пег»), PNG («пинг» или «пи-эн-жи»), GIF («гиф») и WebP (иногда его произносят «виппи», но «веб-пи» звучит, по моему мнению, лучше).

Для веб-страниц также применяется векторный формат SVG (Scalable Vector Graphics, масштабируемая векторная

Фотогеничный JPEG

Формат *JPEG*, названный так по имени создавшей его организации по стандартизации Joint Photographic Experts Group (Объединенная группа экспертов по фотографии), является одним из самых популярных графических форматов, применяемых в Интернете. Его лучше всего использовать, если изображение представляет собой фотографию или содержит плавные переходы цветов (рис. ЦВ-23.2). Схема сжатия JPEG хорошо передает градиент и смешанные цвета, но не особенно успешно обрабатывает сплошные цвета или зубчатые края изображений.

24-битные изображения Truecolor

JPEG-файлы способны отображать миллионы цветов в цветовом пространстве RGB (также называемом *пространством Truecolor*). Это пространство также известно как *24-битный цвет*, потому что каждый из трех цветовых каналов: красный, зеленый и синий — определяется 8 битами информации.

Отображение 24-битного цвета — одна из причин, благодаря которой JPEG и стал идеальным форматом для выведения фотографий — в этом формате имеются все необходимые цвета, которые могут вам когда-либо понадобиться. Для сравнения: другие форматы — такие, как PNG-8 и GIF — основаны на палитре, ограничивающей количество цветов в изображении всего-то значением 256 (почему именно этим количеством, я поясню чуть позже).

Сжатие с потерями

Работа схемы JPEG-сжатия сопряжена с потерями, а это означает, что часть информации об изображении в процессе сжатия отбрасывается (см. врезку «Кумулятивная потеря качества изображения»). К счастью, эта потеря практически незаметна для большинства изображений при использовании приемлемых уровней сжатия. Но если изображение JPEG сжимается сильно, вы увидите цветные пятна и квадраты (называемые *артефактами*), которые возникают в результате работы принятого в этом формате способа сжатия участков изображения (рис. ЦВ-23.3).

Большинство программ работы с изображениями с помощью настройки

ФОРМАТ JPEG

JPEG — наилучший формат для использования в Сети, если ваше изображение является фотографией или содержит мягкие и плавные цветовые переходы.

RGB-ЦВЕТА

RGB-цвета были рассмотрены в главе 13.

КУМУЛЯТИВНАЯ ПОТЕРЯ КАЧЕСТВА ИЗОБРАЖЕНИЯ

Помните, что ухудшенное в результате сжатия *JPEG* качество изображения восстановить невозможно. Поэтому следует избегать повторных сохранений *JPEG*-изображений в формате *JPEG*. Потеря качества изображения будет все время накапливаться — иными словами, каждый раз ухудшение качества изображения, полученное при предыдущем сохранении, будет дополнительно ухудшаться.

Лучше всего придерживаться исходного изображения и, по мере необходимости, экспортировать из него *JPEG*-копии требуемого уровня сжатия. При этом, если появится необходимость изменить уровень сжатия *JPEG*-изображения, вы сможете вернуться к оригиналу и выполнить новое сохранение или операцию экспортации.

Quality (Качество) позволяют управлять тем, насколько сильно вы хотите сжать изображение при его сохранении или экспорте. Здесь вы должны найти компромисс между размером файла и качеством изображения: чем сильнее сжимается изображение (для получения файла меньшего размера), тем больше страдает качество изображения, и наоборот, если максимально сохранять качество, увеличивается размер файла. Наилучший уровень сжатия существенно зависит от конкретного изображения и тех целей, которые преследуются вами при работе над сайтом.

Прогрессивные JPEG-файлы

Прогрессивные JPEG-файлы отображаются в виде серии проходов, начинающейся с версии с низким разрешением, которая становится все четче с каждым последующим проходом, как показано на рис. ЦВ-23.4. В некоторых программах по редактированию изображений можно указывать количество проходов, необходимых для вывода окончательного изображения (3, 4 или 5).

ШИРОКИЙ МИР JPEG

Объединенная группа экспертов по фотографии (Joint Photographic Experts Group) работает, не покладая рук, с момента выпуска оригинального JPEG-формата, который любим всеми нами. За прошедшее время появилось несколько новых стандартов JPEG (JPEG 2000, JPEG XR, JPEG-LS, JPEG XS и др.), которые идут нога в ногу с изменяющимися требованиями к изображениям во всех областях их применения — от цифровых камер до медицинских изображений. Более новые форматы JPEG приобрели такие функции, как сжатие без потерь, возможность хранить 16-битную информацию в цветовых каналах RGB, цветовую модель CMYK и облегченные реализации, которые несложно кодировать и декодировать. Дополнительные сведения по этой теме приведены на сайте JPEG.org.

Преимущество применения прогрессивных JPEG-файлов заключается в том, что пользователи получают представление об изображении еще до его полной загрузки. Кроме того, создание прогрессивного JPEG-файла несколько уменьшает его размер. Однако прогрессивные JPEG-файлы задействуют дополнительные вычислительные мощности, что делает их применение проблематичным на недорогих мобильных устройствах. Тем не менее, несмотря на этот небольшой недостаток, рекомендуется делать все JPEG-файлы прогрессивными не только ради получения файлов меньшего размера, но и потому, что они быстро отображаются на странице, что улучшает общую производительность сайта.

Могучий PNG

Формат PNG (Portable Network Graphics, Портативная сетевая графика) разработан для замены форматов GIF в онлайн-режимах и TIFF — для хранения и печати изображений. В формате PNG можно сохранять изображения многих типов: 8-битный индексированный цвет, 24- и 48-битный цвет RGB и 16-битная шкала серого, но для целей создания веб-сайтов следует выбирать только между 8-битными (PNG-8) и 24-битными (PNG-24) изображениями.

Несмотря на достаточно медленный старт в плане браузерной поддержки, PNG стал одним из самых популярных форматов у разработчиков веб-графики, и это

вполне оправданно, поскольку он предлагает широкий набор функций:

- сжатие без потерь;
- многоуровневую (альфа) или простую вкл/выкл (двоичную) прозрачность;
- прогрессивное отображение в несколько проходов;
- встроенную информацию о гамма-настройке (яркости);
- встроенный текст для прикрепления информации об изображении.

PNG-ИЗОБРАЖЕНИЯ В ГРАДАЦИЯХ СЕРОГО

PNG поддерживает 16-битные изображения в градациях серого — до 65536 оттенков серого (2^{16}), что позволяет сохранять черно-белые фотографии и иллюстрации с огромной степенью детализации, хотя это и не годится для Интернета. В дополнение к большим размерам файлов, необходимым для хранения столь большого количества информации об изображении, такой уровень переходов в оттенках серого обычно не отображается на большинстве компьютерных мониторов.

Формат PNG-8

Формат PNG-8 удобно применять для сохранения изображений со сплошными цветами — таких, как логотипы, штриховые рисунки и пиктограммы (рис. ЦВ-23.5). Вы можете сохранять в нем также и фотографии или текстурированные изображения, но это не будет столь эффективно, как при использовании JPEG, поскольку приведет к увеличению размера файла. Тем не менее PNG-8 отлично подходит для иллюстраций, сочетающих небольшое число фотографических изображений и больших плоских цветных областей.

Две ключевые характеристики PNG-8: использование индексированной цветовой модели и поддержка прозрачности — заслуживают более глубокого изучения.

8-битный индексированный цвет

Ранее я упомянула, что файлы PNG-8 содержат максимум 256 цветов. Уточним, почему это так.

Файлы PNG-8 хранят индексированные цветные изображения, содержащие 8-битную информацию о цвете (их можно сохранять и при меньших битовых значениях). Что это значит? Прежде всего, 8-битный означает, что PNG-8 может содержать не более 256 цветов — это максимальное количество, которое могут определить 8 битов информации ($2^8 = 256$). Использование меньшего числа битов приводит к отображению меньшего числа цветов, что, в свою очередь, уменьшает размер файла. Например, 4-битные изображения содержат только 16 цветов ($2^4 = 16$).

Понятие *индексированный цвет* означает, что набор цветов изображения — его *палитра* — хранится в *таблице цветов* (также называемой *картой цветов*). Каждый пиксель изображения содержит числовую ссылку (или индекс) на позицию цвета

НЕ ЗАБЫВАЙТЕ О ВЕКТОРНОЙ ГРАФИКЕ

Простые иллюстрации, наподобие показанных на рис. ЦВ-23.5, можно нарисовать с помощью векторной графики и сохранить в формате SVG.

ЕЩЕ О ФОРМАТЕ GIF

Файлы в формате GIF также представляют собой 8-битные индексированные цветные изображения, так что понятия битности и палитры применимы и к ним.

в этой таблице. Суть явления проще пояснить на примере — на рис. ЦВ-23.6 показано, как с помощью ссылок на свою таблицу цветов отображается 2-битное (4-цветное) индексированное цветное изображение. Для 8-битных изображений в таблице цветов будет уже 256 ячеек.

Программы редактирования изображений обычно позволяют просматривать таблицу цветов изображения (рис. ЦВ-23.7). Так, в Photoshop вы можете просматривать (и редактировать) таблицу цветов, выбрав команды: **Image | Mode | Color Table** (Изображение | Режим | Таблица цветов). Работая в GIMP, выберите команды: **Windows | Dockable Dialogs | Color Map** (Окна | Закрепляемые диалоговые окна | Цветовая карта). Перед этим сначала надо будет преобразовать изображение в индексированный цветовой режим.

Большинство исходных изображений (сканы, иллюстрации, фотографии и пр.) представлены изначально в формате RGB, так что для сохранения в форматах PNG-8 или GIF их необходимо преобразовать в индексированный цвет. Когда изображение преобразуется из формата RGB в индексированный режим, количество цветов в нем уменьшается до палитры, включающей 256 цветов или менее, — этот процесс известен как *квантование*. В большинстве программ, включая Photoshop, такое преобразование происходит при сохранении или экспорте изображения. Некоторые программы редактирования изображений (например, GIMP) могут потребовать сначала вручную преобразовать изображение в индексированный цвет, а затем, на втором шаге, выполнить экспорт изображения в формат PNG-8.

В любом случае вас могут попросить выбрать палитру для преобразования изображения в цветное индексированное. Во *врезке* «Общие цветовые палитры» представлены различные доступные в популярных графических инструментах параметры палитр. Для достижения наилучших результатов при работе с большинством типов изображений рекомендуется применять выборочную (Selective) и перцептивную (Perceptual) палитру в Photoshop или оптимизированный медианный срез (Optimized Median Cut) в PaintShop Pro. При работе в GIMP весьма полезной может оказаться опция **Generate optimum palette** (Генерирование оптимальной палитры), которая также обеспечивает доступ к длинному списку самых различных пользовательских палитр: Coldfire, Plasma, Paintjet и Bears — лишь некоторые из них.



Рис. 23.8. Прозрачность позволяет сформировать полосатый фон

Прозрачность

Части изображения в формате PNG-8 вы можете сделать полностью прозрачными, что позволяет отобразить из-под них фоновое изображение или цвет. Хотя все растревые изображения по своей природе прямоугольные, прозрачность создает иллюзию, что изображение имеет более интересную форму (рис. 23.8). В наиболее

ОБЩИЕ ЦВЕТОВЫЕ ПАЛИТРЫ

Все 8-битовые индексированные цветные изображения, такие, как PNG-8 и GIF, для определения цветов в изображении применяют палитры, причем на выбор имеется несколько стандартных палитр. Некоторые из них предоставляют возможности создания пользовательской палитры на основе цветов, имеющихся в изображении, другие же просто применяют к изображению имеющуюся палитру.

- **Exact** (точная) — создание пользовательской палитры на основе фактических цветов изображения, если изображение содержит менее 256 цветов.
- **Adaptive** (адаптивная) — создание пользовательской палитры на основе пикселов, наиболее часто используемых в отображении цветов. Это позволяет уменьшить глубину цвета при сохранении исходного характера изображения.
- **Perceptual** (перцептивная, только Photoshop) — создание пользовательской таблицы цветов, отдавая приоритет тем цветам, которые лучше воспринимает человек. В отличие от адаптивной палитры, она основана на алгоритмах, а не на количестве пикселов. Обычно применение этой палитры приводит к формированию изображений с улучшенной интегрированностью цветов, чем при использовании адаптивной палитры.
- **Selective** (селективная, только Photoshop) — подобна перцептивной палитре, но предпочтение отдается областям расширенного цвета.
- **Web Adaptive, Restrictive или Web216** (веб-адаптивная, ограничительная или Веб216) — создает палитру исключительно на основе 216 цветов, которые не размываются при отображении на 8-битных мониторах. Поскольку 8-битные мониторы остались в прошлом, эта палитра (известная как «веб-палитра») не актуальна и не рекомендуется к применению.
- **Custom** (пользовательская) — позволяет загружать ранее сохраненную палитру и применять ее к текущему изображению. В противном случае сохраняет в палитре текущие цвета.
- **System** (системная палитра Windows или Macintosh) — использует цвета палитры для указанной по умолчанию системы.
- **Optimized Median Cut** (оптимизированный медианный срез, только Paint Shop Pro Photo) — снижает количество цветов в изображении до нескольких цветов, используя подход, похожий на применение адаптивной палитры.
- **Optimized Octree** (оптимизированное октадерево, только Paint Shop Pro Photo) — рекомендуется, если в исходном изображении лишь несколько цветов и необходимо точно сохранить эти цвета.

распространенном типе прозрачности PNG-8 пиксели становятся либо полностью прозрачными, либо полностью непрозрачными, что также известно как *двоичная прозрачность*.

Файлы в формате PNG-8 также могут сохранять в индексированных цветовых картах несколько уровней прозрачности, позволяя мягким краям и теням сливаться с фоном. В прошлом, хотя браузеры поддерживали PNG-8 с несколькими уровнями прозрачности, было сложно найти инструмент для редактирования изображений, который мог бы их создавать. Теперь же вы можете создавать прозрачные

изображения PNG-8 непосредственно в Photoshop CC, причем имеется ряд инструментов для преобразования формата PNG-24 в PNG-8 при сохранении его уровней прозрачности.

Как двоичную, так и переменную прозрачность применительно к файлам формата PNG-8 мы рассмотрим в разд. «Работа с прозрачностью» главы 24.

Формат PNG-24

В формате PNG также можно сохранять изображения Truecolor, при этом каждый канал (красный, зеленый и синий) определяется 8- или 16-битной информацией, что приводит к 24- или 48-битным RGB-изображениям соответственно. Во многих графических программах 24-битные RGB-изображения в формате PNG идентифицируются как изображения PNG-24. Следует отметить, что 48-битные изображения, которые отлично подходят для сохранения высококачественных оригиналов, бесполезны для использования в Интернете из-за больших размеров своих файлов, да

и 24-битовые изображения в этом плане тоже не являются лучшим выбором. Как и формат JPEG, PNG-24 хорош для передачи фотографических изображений, где необходим максимальный цветовой диапазон.

КЛЮЧЕВЫЕ ХАРАКТЕРИСТИКИ PNG-24

Две ключевые характеристики формата PNG-24 заключаются в том, что они передают изображение «без потерь» и могут содержать несколько уровней прозрачности.

Две ключевые характеристики формата PNG-24 заключаются в том, что изображения в этом формате «лишены потерь» и могут отображать несколько уровней прозрачности. Рассмотрим эти моменты подробнее.

Сжатие без потерь

Известно, что в алгоритме сжатия с потерями JPEG часть информации об изображении отбрасывается для сокращения размера файла. В файлах PNG-24 такой потери данных не происходит, поэтому размер 24-битного файла PNG «без потерь» почти всегда существенно больше, чем размер файла JPEG «с потерями» этого же изображения. По этой причине JPEG-файлы и являются лучшим выбором для публикации фотографий в Интернете.

Тем не менее PNG-24 был первым форматом, обеспечившим функцию, которая сделала его одним из самых популярных форматов в Интернете, и это...

Альфа-прозрачность

Файлы PNG-24 могут содержать множество уровней прозрачности, обычно называемой альфа-прозрачностью. При этом 8-битная информация о прозрачности (256 уровней) сохраняется в четвертом канале, который называется альфа-каналом.

Иногда формат PNG-24 с альфа-прозрачностью называют 32-битным PNG, потому что для каждого из четырех каналов — красного, зеленого, синего и альфа — используются 8 битов.

На рис. ЦВ-23.9 показано одно и то же PNG-изображение, размещенное поверх двух разных изображений фона. Оранжевый круг полностью непрозра-

чен, но тень от него содержит несколько уровней прозрачности, которые варьируются от почти непрозрачных до полностью прозрачных. Множество хранящихся в PNG уровней прозрачности позволяет реализовать плавное перетекание падающей тени на любом фоне. Особенности свойства альфа-прозрачности мы рассмотрим в разд. «Работа с прозрачностью» главы 24.

Дополнительные возможности

Имеется еще несколько функций, которые превращают PNG в формат Pretty Nifty Graphic (Весьма изящная графика) 😊:

- *гамма-коррекция* — эта функция управляет настройкой яркости монитора. Изображения PNG могут нести информацию о гамма-настройке среды, в которой они создавались. Когда гамма-коррекция реализована в изображении и поддерживается браузером, формат PNG передает при воспроизведении заданную яркость и интенсивность цвета. К сожалению, эта функция поддерживается не всегда. Да и инструменты оптимизации изображений обычно удаляют фрагменты кода, управляющего гамма-коррекцией. Впрочем, как бы там ни было, при неудовлетворительной поддержке гамма-коррекции браузером ничего не теряется, кроме ненужных байтов;
- *информация о встроеннном цветовом профиле* — формат PNG также может хранить информацию о цветовом профиле ICC (International Color Consortium) системы, в которой он создавался. И если вы обнаружите, что трудно сопоставить значение RGB в PNG с тем же значением RGB в цвете фона, виноват в этом будет встроенный цветовой профиль. Блок кода, хранящий профили ICC, также обычно удаляется оптимизаторами изображений;
- *встроенный текст* — изображения в формате PNG способны хранить строки текста. Это полезно для постоянного прикрепления к изображению текста, содержащего, например, информацию об авторских правах или описание того, что демонстрирует изображение. В идеале метаинформация в PNG должна

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ

Браузер *Safari* сейчас поддерживает профили цветов ICC. В статье Криса Коуэра (*Chris Coyier*) «*CSS Tricks*» (css-tricks.com/color-rendering-difference-firefox-vs-safari/) приводится хороший обзор этой темы.

АНИМИРОВАННЫЕ PNG-ИЗОБРАЖЕНИЯ

Формат APNG (Animated PNG, анимированный PNG) представляет собой расширение PNG, добавляющее ему возможность анимации кадров. В дополнение к 8-битной анимации он включает также поддержку анимированных 24-битных изображений с альфа-прозрачностью. Формат APNG поддерживается текущими версиями Chrome, а также настольными и мобильными версиями Firefox, Safari и Opera, однако его не поддерживает ни одна из версий браузеров Internet Explorer или MS Edge. Как только поддержка этого формата расширится, а устаревшие версии браузеров выйдут из употребления, APNG, безусловно, заменит анимированные GIF-файлы (на платной основе).

быть доступна по щелчку правой кнопкой мыши на графическом изображении в браузере, но эта функция никогда не была реализована;

- *прогрессивное отображение (с чередованием)* — формат PNG может быть запrogramмирован на чересстрочное отображение (interlacing), когда изображение как бы проявляется в серии из семи проходов, открываясь как по горизонтали, так и по вертикали. Чересстрочная развертка увеличивает размер файла, и, как правило, в ней нет особой необходимости, поэтому, чтобы файлы были как можно меньше, отключите отображение чересстрочной развертки. Я считаю, что на современном этапе большинство инструментов все равно не позволяют использовать эту возможность.

Заключение

Прежде чем продолжить, подведем краткий итог рассказу о PNG-изображениях:

- если имеется растровое изображение с областями сплошного цвета, с прозрачными областями или без них, PNG-8 является наиболее подходящим форматом для его хранения, поскольку обеспечивает получение файла наименьшего размера;
- если вам нужны переменные уровни прозрачности, то, независимо от типа изображения, формат PNG-24 может оказаться единственным вариантом, который вам обеспечит ваши инструменты работы с изображениями. Однако размеры файлов в этом формате больше, чем файлы PNG-8;
- если у вас имеется фотографическое изображение без прозрачности, вы также можете применить формат PNG-24, но формат JPEG в этом случае практически всегда обеспечит файл меньшего размера.

«Дедушка» форматов GIF

Формат GIF (Graphic Interchange Format, Формат обмена графической информацией) был первым форматом изображений, который поддерживался веб-браузерами, и некоторое время был также единственным, который вообще отображался в окне браузера. (Да-да, я это знаю точно.) И хотя он и не был разработан специально для Интернета, однако его универсальность, небольшой размер файла и кросс-платформенная совместимость способствовали его широкому распространению.

СПИСОК «COMPUSERVE GIF»

Формат GIF можно найти в списке «Compuserve GIF», потому что именно Compuserve создал этот формат. Срок действия патента на GIF, принадлежащий Unisys, истек в 2006-м году.

тв для веб-дизайна (по крайней мере, до тех пор пока формат APNG и анимированный WebP не получат более основательной поддержки). Однако для изображений, не содержащих анимацию, GIF уступает формату PNG, который может делать все, что умеет GIF, и обычно лучше. Более того, в новых графических инструментах просто отсутствует опция сохранения файлов в формате GIF. Так что наш старый друг GIF может отправляться на пенсию. Но его возможности мы вкратце все же рассмотрим.

В наше время GIF является синонимом «анимированного вирусного мема» и, будучи единственным хорошо поддерживаемым форматом веб-изображений, способных к анимации, продолжает занимать достойное место в ряду форматов

8-битный индексный цвет

Подобно PNG-8, GIF является 8-битным индексированным цветовым форматом. Вы можете сохранить файл GIF с еще меньшой битовой глубиной, что приведет к сокращению числа цветов и уменьшению размера файла.

GIF-сжатие

GIF-сжатие является сжатием *без потерь*, хотя некоторая информация об изображении и теряется при преобразовании цветов RGB в индексированный цвет. При сжатии GIF используется схема сжатия LZW (аббревиатура имен Lempel-Ziv-Welch (Лемпел-Зив-Велш) — авторов схемы сжатия), основанная на поиске повторяющихся данных. То есть если в коде встречается строка пикселов одинакового цвета, она сжимается до описания одного преобладающего цвета (рис. ЦВ-23.10). Поэтому изображения с большими областями однородного цвета сжимаются лучше, чем изображения с текстурами. Формат PNG использует аналогичную схему сжатия цвета.

Прозрачность

Изображения GIF используют двоичную прозрачность, при которой пиксели либо полностью прозрачны, либо совсем непрозрачны.

Чересстрочная развертка

Чересстрочная развертка позволяет отображать GIF в серии проходов, как и в случае с применением прогрессивных JPEG-изображений. Каждый проход приводит к более четкому отображению по сравнению с предыдущим, пока все изображение окончательно не отобразится в окне браузера (рис. ЦВ-23.11). При быстром соединении эти эффекты (чересстрочная развертка или задержки при появлении изображения) могут быть и незаметными. Однако при медленных соединениях последовательность вывода крупных изображений дает пользователю понять, каким будет окончательное изображение.

Анимация

Еще одна особенность, присущая формату GIF-файла, — возможность отображать простые анимации (рис. 23.12). Многие из видимых вами вращающихся, мигающих, периодически исчезающих или иным образом движущихся рекламных баннеров представляют собой анимированные GIF-файлы, и они, безусловно, встречаются в ваших социальных сетях.

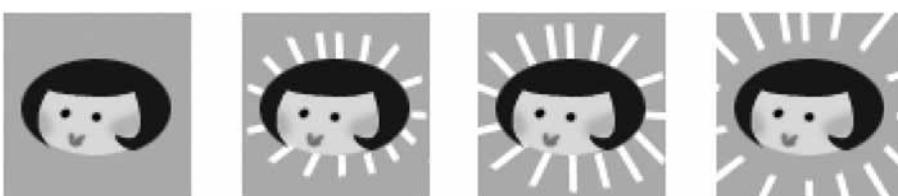


Рис. 23.12. Все кадры для этой простой анимации содержатся в одном GIF-файле.

Анимированный GIF-файл содержит несколько кадров анимации, которые представляют собой отдельные изображения, но при быстром последовательном просмотре создают иллюзию движения или изменения во времени — как при мультипликации. Все кадры анимации сохраняются в одном файле GIF вместе с настройками, описывающими порядок их воспроизведения. Эти настройки определяют, сколько раз повторяется последовательность, как долго каждый кадр остается видимым (*задержка кадра*), каким образом один кадр сменяет другой (*метод удаления*), является ли изображение прозрачным или «чересстрочным».

Существует большое число инструментов для создания анимированных GIF-файлов (просто выполните быстрый поиск). Многие из них являются веб-приложениями, которыми можно воспользоваться прямо из браузера или на мобильном устройстве, и многие из них распространяются бесплатно. Вы также можете создать анимированный GIF-файл в Photoshop, используя окно временной шкалы (Timeline) и выбрав опцию *Create Frame Animation* (Создать анимацию кадра).

Новый формат WebP

В наши дни появился новый формат изображений, который превосходит практически все иные форматы. Google называет этот открытый формат WebP «швейцарским армейским ножом графических форматов».

РОДСТВЕННЫЕ ФОРМАТЫ

Формат WebP является «родственным» для формата WebM — видеоформата с открытым исходным кодом.

Формат WebP обладает практически всеми функциями, присущими форматам JPEG, PNG и GIF, причем размеры файлов формата WebP меньше размеров файлов других форматов на 25–35%:

- *сжатие «без потерь» или «с потерями»* — формат WebP может сохраняться в формате «с потерями» (как, например, JPEG) или «без потерь» (как, например, PNG). Схема сжатия «с потерями» основана на той же кодировке, что и видео-кодек VP8;
- *альфа-прозрачность* — в WebP имеется альфа-канал для нескольких уровней прозрачности (как, например, у PNG-24). Альфа-прозрачность может применяться либо со сжатием изображения «без потерь» (в стиле PNG), либо — и это особенно важно — со сжатием «с потерями» (в стиле JPEG). Это единственный формат, объединяющий канал RGB «с потерями» и альфа-канал «без потерь», в результате чего результирующий файл оказывается на 60–70% меньшим, чем в случае файла PNG-24, используемого для хранения того же изображения;
- *анимация* — так же, как и GIF, вы можете анимировать изображения в формате WebP (извини, GIF, это было ваше преимущество);
- *метаданные* — подобно PNG, контейнер WebP может хранить метаданные прямо в своем коде;
- *цветовой профиль* — контейнер WebP также может нести информацию о цветовом профиле (ICC).

Еще о браузерной поддержке

Вот мы и добрались до «грустного тромбона» этой истории. Поскольку WebP формат новый, у него нет достаточной поддержки со стороны браузеров. На момент подготовки этой книги формат WebP поддерживается только новыми браузерами Chrome, Android, Opera, Vivaldi и Samsung. Но это не значит, что вы не можете его использовать! Современный веб-разработчик знает, что это хороший подход для обеспечения наилучшего (в данном случае — самого быстрого) метода обработки веб-страниц браузерами, которые могут с этим справиться, и для побуждения остальных браузеров к этому стремиться.

Вы можете воспользоваться инструментом Modernizr (см. главу 19), чтобы обнаружить формат WebP во всех его разновидностях: «с потерями», «без потерь», с альфа-каналами и с анимационными возможностями. Вы также можете применить элемент `picture`, чтобы предоставить файлы изображения в формате `*.webp` браузерам, которые способны его воспроизводить, и в формате JPEG — в качестве резервного варианта, как было показано в главе 7:

```
<picture>
  <source type="image/webp" srcset="pizza.webp">
    
</picture>
```

Кроме того, сервер обычно выполняет вызов и доставляет изображения WebP, когда обнаруживает, что браузер их поддерживает (на основе «заголовка подтверждения кодировки»). Впрочем, серверные решения выходят за рамки нашего обсуждения, однако об их наличии вам следует знать.

Создание файлов в формате WebP

Как и в случае с браузерами, формату WebP еще предстоит дождаться, пока не появятся инструменты создания изображений на его основе. Впрочем, уже сейчас мы можем создавать файлы WebP с помощью приложений Sketch, Pixelmator и некоторых других графических программ. Текущий список поддерживающих этот формат программ находится на странице Википедии, ему посвященной (ru.wikipedia.org/wiki/WebP). Имеется твердая уверенность в том, что полная поддержка WebP будет добавлена в GIMP для версии 2.10, которая еще не выпущена на момент подготовки этой книги.

В Adobe Photoshop доступны два плагина, которые позволяют сохранять изображения в формате WebP. Первый, созданный Тоби Тэйном (Toby Thain), доступен на сайте по адресу: telegraphics.com.au/sw/product/WebPFormat, а более новый, разработчиком которого является Брэндан Боллес (Brendan Bolles), доступен на сайте по адресу: [atgithub.com/fnordware/AdobeWebM](https://github.com/fnordware/AdobeWebM). После установки такого плагина WebP появится в списке форматов, в которых можно выполнять сохранение файлов.

Наконец, для преобразования изображений PNG и JPEG в формат WebP можно воспользоваться инструментом командной строки `cwebp` (обратите внимание, этот инструмент предназначен не только для программистов!). А команда `dwebp` преобразует формат WebP в PNG.

Дополнительные сведения

Формат WebP имеет свой официальный сайт, который доступен по адресу: developers.google.com/speed/webp. Здесь находится подробная документация, приводятся объяснения схем сжатия, содержатся обновленные списки вспомогательных инструментов и браузеров (включая ссылки для загрузки упомянутых ранее плагинов и инструментов командной строки), а также — коллекция примеров. Определенно стоит ознакомиться с этим материалом, если вы собираетесь работать с форматами изображений. А форматом WebP, безусловно, не стоит пренебрегать.

Выбор лучшего растрового формата

Первый шаг на пути к созданию качественной и быстро загружаемой веб-графики — это выбор правильного формата изображений. В табл. 23.1 вы найдете информацию, которая вам в этом поможет. Из-за недостаточной на момент подготовки этой книги поддержки формата WebP я включила в нее только форматы растровых изображений PNG, JPEG и GIF.

Учтите также, что для иллюстраций и значков с однородными цветами лучше всего подходит векторный формат SVG. Использование этого формата дает возможность уменьшить размер файлов для иллюстраций, содержащих комбинации однородных цветов и небольших областей растровых изображений, градиентов или таких эффектов, как тени. Об этом формате речь пойдет в главе 25, а сейчас содержание табл. 23.1 должно помочь вам разобраться с параметрами растровых файлов.

На этом мы завершаем рассмотрение форматов изображений. Полагаю, вы весьма часто слышали фразу: «Если это фотография, используйте JPEG, а если картинка содержит в основном однородные цвета, используйте PNG-8», но ведь важно понимать, *почему* это так.

Таблица 23.1. Выбор наилучшего растрового формата файла

Если ваше изображение...	Используйте...	Потому что...
Графическое, с однородными цветами	8-битный PNG или GIF	PNG и GIF превосходны при сжатии однородного цвета
Представляет собой фотографию или включает градиентный цвет	JPEG	JPEG-сжатие лучше всего ведет себя при обработке изображений со смешанными цветами. Поскольку эта обработка выполняется «с потерями», это обычно приводит к меньшим размерам файлов, чем в случае 24-битного PNG
Является комбинацией однородных и фотографических изображений	8-битный PNG или GIF	Индексированные цветные форматы лучше всего сохраняют и выполняют сжатие областей однородного цвета. Пикселизация (dithering), которая проявляется в фотографических областях в результате сокращения палитры изображения, обычно не вызывает проблем

Табл. 23.1 (окончание)

Если ваше изображение...	Используйте...	Потому что...
Требует прозрачности	GIF или PNG-8	И GIF, и PNG позволяют включать/выключать прозрачность изображений
Требует нескольких уровней прозрачности	PNG-24 или PNG-8	Только PNG поддерживает несколько уровней прозрачности. Файлы PNG-24 с альфа-прозрачностью имеют гораздо больший размер файла, но зато проще найти инструменты для их создания. Формат WebP также поддерживает альфа-прозрачность и может стать лучшим вариантом, если реализуется его поддержка
Требует анимации	GIF	GIF — единственный поддерживаемый формат, который может включать кадры анимации. Форматы APNG и WebP в будущем могут стать более приемлемыми вариантами

РАЗМЕР И РАЗРЕШЕНИЕ ИЗОБРАЖЕНИЯ

С недавнего времени специалистов, разрабатывающих веб-страницы и веб-приложения, стали называть *дизайнерами экрана*. И это правильно! По мере развития Интернета и средств, с помощью которых люди им пользуются, — мониторов и смартфонов, становится ясно, что требования к дизайну материалов, отображаемых на экранах, отличаются от требований к дизайну материалов, предназначенных для вывода на печать. Как веб-дизайнер, вы должны хорошо разбираться в том, как изображения отображаются на экранах, так что давайте вникнем в эту тему поглубже.

Изображения GIF, JPEG, PNG и WebP объединяет то, что все они являются *растровыми*. Увеличивая растровое изображение, вы увидите, что оно становится похожим на мозаику, состоящую из множества пикселов (крошечных одноцветных квадратиков). В этом и заключается существенное отличие растровой графики от векторной, состоящей из плавных линий и заполненных областей, форма которых определяется математическими выражениями (рис. ЦВ-23.13).

Разрешение изображения

Программы редактирования изображений отслеживают количество пикселов изображения, содержащихся в каждом его дюйме. Эта величина (количество пикселов на дюйм — *ppi*) и называется *разрешением* цифрового изображения. Для напечатанного на бумаге изображения более высокое значение его *ppi* означает, что оно будет более резким, более высокого качества, поскольку каждый квадратный его дюйм содержит большее количество элементов (см. врезку «*DPI и PPI*»). В мире печати чаще всего используются изображения с величинами разрешения, равными 300 и 600 пикселов на дюйм.

DPI И PPI

Разрешение цифровых изображений измеряется в *пикселях на дюйм* (ppi). Однако, когда дело доходит до вывода на печать, возможности принтеров и разрешение напечатанных страниц оцениваются в *точках на дюйм* (dpi) — количестве напечатанных точек, из которых состоит каждый дюйм изображения. Чем больше чернильных точек приходится на дюйм, тем четче изображение. Количество точек на дюйм напечатанного изображения (dpi) может соответствовать, а может и не соответствовать количеству пикселов на дюйм цифрового изображения (ppi).

Читая литературу, вы иногда можете встретить термины «dpi» и «ppi», используемые взаимозаменяющими (хотя это и неправильно), но теперь вам известна разница между ними.

Однако в Интернете понятие «дюйма» значения не имеет, поскольку окончательный размер изображения на экране зависит от разрешения экрана, на котором оно отображается (рис. 23.14).

А если мы отбрасываем понятие «дюйм», то должны также отбросить и понятие «пиксель на дюйм». И единственное, что мы знаем наверняка, так это то, что графическое изображение, показанное на рис. 23.14, имеет ширину 72 пикселя и оно выглядит в два раза шире, чем графическое изображение с шириной 36 пикселов. Отсюда основной вывод: *веб-изображения измеряются в количествах пикселов, и величина*

ГЛАВНАЯ ХАРАКТЕРИСТИКА ВЕБ-ИЗОБРАЖЕНИЙ

Веб-изображения характеризуются количеством пикселов. Разрешение (ppi), при котором они создаются, не имеет значения.

ppi, с учетом которой они создаются, значения не имеет.

Рекомендуется создавать изображения с разрешением, равным 72 ppi, если вы используете редактор растровых изоб-

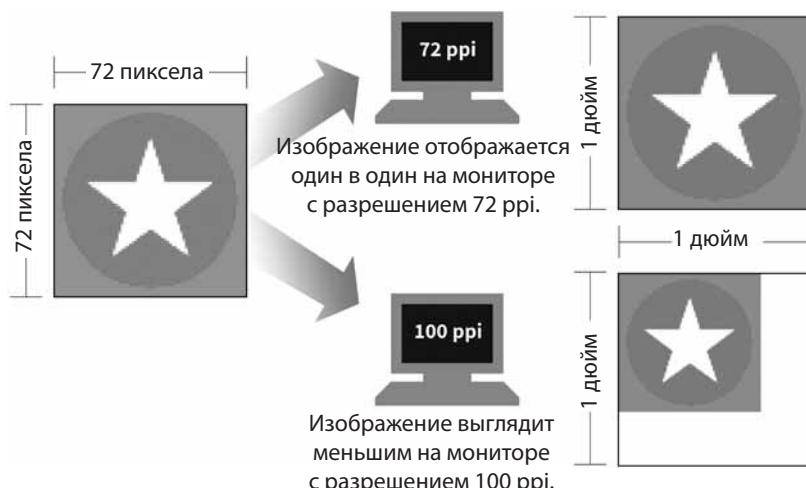


Рис. 23.14. Дюймы и, следовательно, «пиксели на дюйм» неприменимы к цифровым устройствам, для которых размер изображения зависит от разрешения экрана

ражений типа Photoshop или GIMP. Такое значение задано в них по умолчанию, и изображения будут сохраняться примерно в том размере, в каком они будут отображаться на мониторе настольного компьютера. Вы можете создавать изображения и с другим значением `ppi`, но сначала убедитесь в том, что это значение применяется во всех файлах проекта, поскольку размер изображений не изменяется при копировании и вставке их из одного файла в другой.

Разрешение экрана

Экранные дисплеи состоят из пикселов, поэтому их разрешение также можно измерять в пикселях на дюйм (`ppi`). При этом для экранов часто используется термин **пиксельная плотность**.

Первые компьютеры Macintosh имели экраны с разрешением 72 `ppi`, что сейчас считается весьма низким значением. Мониторы первых ПК имели разрешение 96 `ppi`. В наши дни стандартные мониторы для настольных компьютеров и ноутбуков имеют разрешение от 109 до 160 пикселов на дюйм. На протяжении многих лет производители постоянно увеличивают разрешение дисплеев, что привело к появлению мониторов с высокой плотностью.

РАЗМЕР ЭКРАНА И ЕГО РАЗРЕШЕНИЕ

Вы можете встретить рекламные описания мониторов, имеющих разрешение 2560×1440 пикселов, но в данном случае это не «разрешение», а размер экрана. Разрешением является мера плотности пикселов.

Мониторы высокой плотности

В период с 1980-х по 2010-й год мы могли в значительной степени рассчитывать на то, что пиксели наших изображений будут отображаться один в один с аппаратными пикселями настольных мониторов, как показано на рис. 23.14. Конечно, существовали исключения — браузеры, повинуясь нашим командам, могли увеличивать или уменьшать изображения, изображения также уменьшались, чтобы поместить их на экранах смартфонов, — но в целом правило соблюдалось.

В 2010-м году произошел серьезный прорыв, когда Apple представила iPhone 4 с дисплеем Retina. Дисплей Retina упаковал в два раза больше пикселов в одно и то же физическое пространство экрана, в результате чего изображения стали выглядеть намного четче (помните: чем больше пикселов на дюйм, тем лучше качество изображения). Оборотная сторона этого эффекта заключается в том, что стандартные растровые изображения стали растягиваться на вдвое большее число пикселов и выглядеть несколько размытыми (рис. ЦВ-23.15).

С дисплея Retina все только началось... В настоящее время уже имеются как 2-, так и 3-кратные устройства Apple (включая планшеты и ноутбуки), а устройства Android обрели 1,5-кратную пиксельную плотность, 2-, 3- и даже 4-кратную. В результате фактический пикセル устройства стал настолько мал, что изображения и текст оказались бы ужасно неразборчивыми, если бы отображались реальные (физические) пиксели. Как быть в подобной ситуации?! Ответ на этот вопрос вы найдете в следующих разделах.

Опорные пиксели: PT и DP

Если вы вспомните нашу дискуссию по поводу адаптивных изображений в главе 7, то поймете, что решение этой проблемы существует. Для устройств с высоким разрешением применяется единица измерения, называемая *опорным пикселом* (*reference pixel*), и ее можно использовать для целей макетирования. Опорные пиксели называются по-разному и вычисляются с небольшими различиями в зависимости от операционной системы, но они позволяют указывать размеры в пикселях, не привлекая понятия, связанные с пиксельной плотностью.

Apple называет свои опорные пиксели *пунктами* (*points*, PT). Один пункт на стандартном 1-кратном экране равен одному пикселу устройства. На 2-кратном экране пункт равен 2×2 пикселя устройства, а на 3-кратном — 3×3 пикселя устройства. Все они выглядят по размеру примерно одинаково, поскольку пиксели с высоким разрешением невероятно малы. Android называет свои опорные пиксели *независимыми от устройства пикселями* (*device-independent pixels*, DiP), или просто DP. Они всегда равны одному пикселу при разрешении 160 ppi, но работают одинаково.

Вероятно, к терминологии PT и DP целесообразно обращаться при разработке графики для использования в приложениях, предназначенных именно для смартфонов. Если же изображения предназначены для публикации в Интернете, достаточно выполнить дизайн макета с применением единиц измерения, выраженных в пикселях и в соответствующих CSS-единицах. Например, можно утверждать, что изображение для 2-кратных мониторов, показанное на рис. ЦВ-23.15 (внизу), имеет ширину 350 (опорных) пикселов (как задано в разметке), даже если файл изображения имеет ширину 700 пикселов.

Краткое резюме

Что бы мы ни говорили, но для своих макетов вы можете создавать изображения с размерами просто в пикселях. Однако иногда вам может понадобиться, чтобы некоторые важные изображения на дисплеях высокой плотности выглядели бы максимально четкими. В этом случае вам нужно будет или создать несколько версий этих изображений и выводить их на экран с помощью адаптивной разметки, или позволить заняться этим серверу. Так, если у вас есть фото продукта с разрешением 150×150 пикселов для 1-кратного экрана, вам понадобится как минимум версия

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ

Для получения более подробных сведений об изображениях и разрешениях экранов обратитесь к следующим материалам:

- Питер Новелл (Peter Nowell), «Pixel Density, Demystified» (medium.com/@nowelldesign/pixel-densitydemystified-a4db63ba2922);
- Себастьян Габриэль (Sebastien Gabriel), «Designer's guide to DPI» (sebastien-gabriel.com/designersguide-to-dpi/).

для 2-кратного (300×300) и, возможно, версия для 3-кратного (450×450), с учетом того что все они будут в макете занимать 150 опорных пикселов.

В главе 24 мы рассмотрим инструменты и методы для создания нескольких изображений разных размеров, предназначенных для мониторов с высокой плотностью. Разметка, применяемая для доставки изображения правильного размера на нужное устройство, описана в главе 7.

СТРАТЕГИЯ РАБОТЫ С ИЗОБРАЖЕНИЯМИ

К этому моменту вам известно, где можно получить изображения. Вы также познакомились с различными вариантами веб-форматов и понятиями, связанными с разрешением экрана. А из предыдущих глав книги узнали о важных принципах производительности и адаптивного веб-дизайна. Давайте объединим все знания в стратегию работы с изображений.

Как добросовестный веб-дизайнер, стремящийся предоставить наилучший вариант веб-страниц, отображаемых на широком спектре устройств, при создании графики для сайта вы должны учитывать следующие приоритеты:

- сохранять файлы изображений как можно в меньшем размере;
- минимизировать количество HTTP-запросов к серверу;
- не загружать изображений большего размера, чем это необходимо для устройств с маленькими экранами;
- доставлять для отображения на мониторах высокой плотности высококачественные изображения.

Весьма полезно системно оценивать требования к изображениям, исключая классы изображений и ненужные задачи, чтобы у вас остался четкий набор производственных параметров. На рис. 23.16 (стр. 790) представлена серия вопросов, ответы на которые позволят вам отбросить ненужные варианты создания изображений. В этом разделе мы рассмотрим каждый шаг процесса на концептуальном уровне, а в *главе 24* вы познакомитесь с конкретными методами создания изображений с учетом поставленных целей.

Прежде всего, определим, столь ли необходимо вам изображение.

Можно ли это сделать с помощью CSS?

Прежде чем начать работу с Photoshop, подумайте, сможете ли вы достичь желаемого эффекта только с помощью CSS. В понятие «эффекта» входит не только минимальный размер файла, но и возможность избежать лишнего обращения к серверу.

Такие эффекты, как закругленные углы и градиенты, которые вы захотите придать изображениям, теперь достижимы исключительно с помощью CSS-свойств (`border-radius` и `radial-gradient/lineargradient` соответственно).

С помощью CSS также можно создавать небольшие рисунки, которые могут быть удобнее пиктограмм (рис. 23.17). Основные фигуры, такие, как круги, прямоугольники, треугольники и т. п., можно создавать с помощью пустых элементов `div` и определенных преобразований границ и трансформации. Некоторые пользователи создали удивительно сложные иллюстрации, используя лишь HTML и CSS, но эта техника, расцвет которой пришелся на период с 2010-го по 2011-й год, в основном предназначена для демонстрационных целей, а не для выполнения серьезных задач по созданию сайтов.

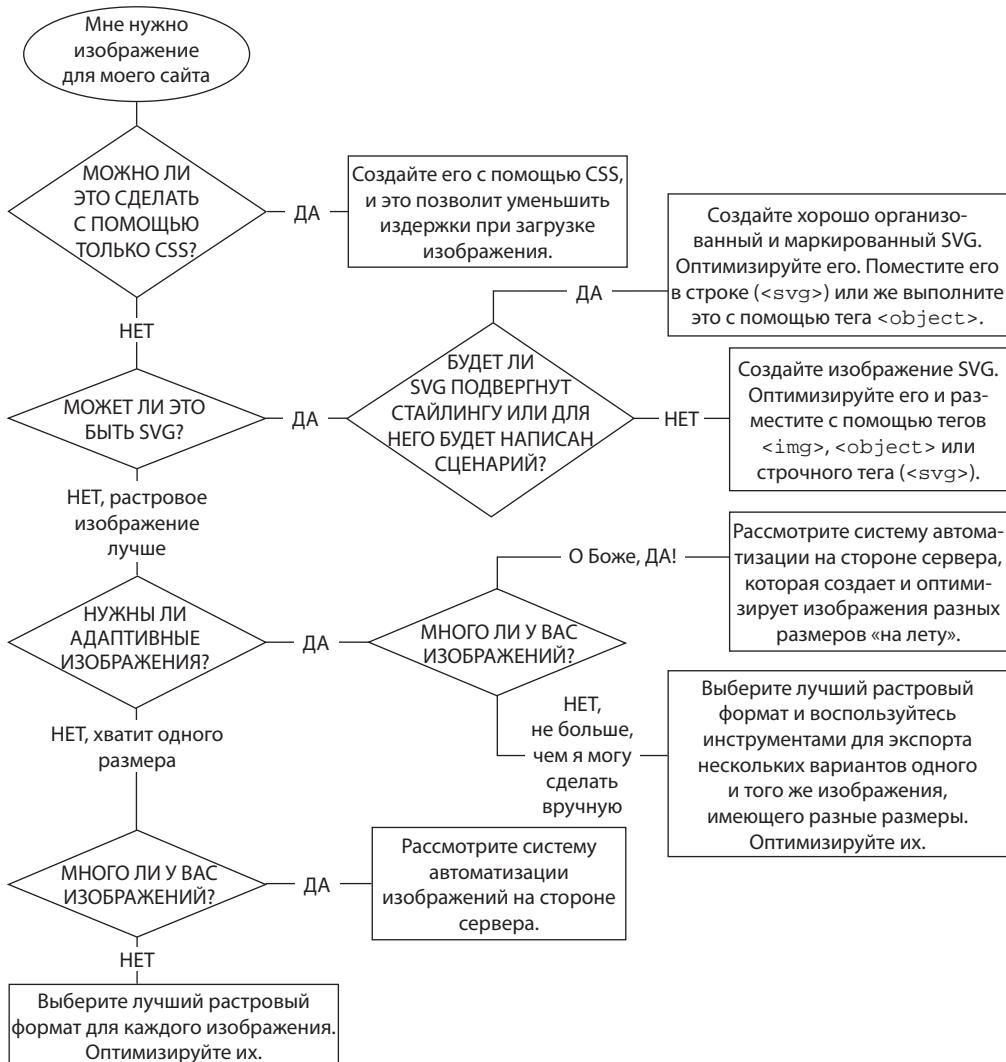
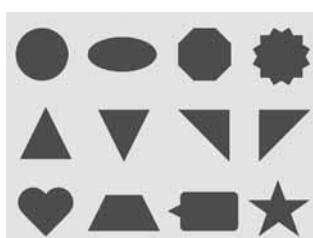


Рис. 23.16. Эта блок-схема может показаться вам просто жуткой, но она позволит уменьшить количество вариантов создания изображений. Затронутые в ней вопросы мы подробно рассмотрим далее



Формы, полученные Крисом Койером
css-tricks.com/examples/ShapesOfCSS/



Коала от Майкла Мангиларди
codepen.io/mikemang/pen/oYMePj



Чизкейк от Саши Тран
codepen.io/sashatran/pen/ggGeZr

Рис. 23.17. Эти небольшие рисунки созданы исключительно с помощью HTML-разметки и CSS

В этой главе я не хочу далеко отходить от методики создания изображений, поэтому ориентирую вас на ознакомление с работами, в которых можно больше узнать о CSS-формах и иллюстрациях:

- «The Shapes of CSS» — галерея одноэлементных форм CSS с кодом, примененным для их создания, составленная Крисом Койером (Chris Coyier): css-tricks.com/examples/ShapesOfCSS/;
- «Beginners Guide to Pure CSS Images» — пошаговое руководство Майкла Мангиаларди (Michael Mangialardi) по созданию изображения медведя коалы (см. рис. 23.17): medium.com/coding-artist/a-beginners-guide-to-pure-css-images-ef9a5d069dd2;
- Коллекция «not-so-semantic drawings made with CSS», собранная Уго Жирауделем (Hugo Giraudel) и доступная на сайте Codepen: codepen.io/collection/kFeDz/3/

Если вам необходимо получить более сложный эффект, чем могут дать CSS, обратитесь к форматам изображений.

Может ли это быть SVG?

Если изображение представляет собой логотип, пиктограмму или другую подобную иллюстрацию, создание его с помощью инструмента векторного рисования и сохранение в формате SVG обеспечивает преимущества, заключающиеся в получении файла небольшого размера и независимости от разрешения. Теперь, когда имеет место надежная поддержка SVG браузерами, это хорошее решение для применения на различных устройствах и экранах.

Если поместить SVG-код в строку с помощью элемента `svg`, вы избавитесь от еще одного HTTP-запроса и получите возможность задать ему стили и написать для него сценарий. Или, если вам нужна лишь статическая иллюстрация, идеальным вариантом представляется встраивание SVG в документ с помощью элемента `img`.

В главе 25 мы подробно рассмотрим работу с форматом SVG, поэтому я не хочу сейчас много о нем говорить, напомню только, что SVG должен быть вашим первым выбором, если вы создаете изображение или иллюстрацию в векторном формате.

Если формат SVG не подходит для вашего типа изображения или известно, что ваша целевая аудитория использует браузеры, не поддерживающие его, вам может потребоваться растровый формат.

Что такое лучший растровый формат?

От формата изображения сильно зависит размер его файла, поэтому выбор для изображения наиболее подходящего формата — это важный шаг на пути к оптимизации этого изображения. Как уже отмечалось, PNG-8 представляет собой лучший вариант для изображений с областями однородного цвета, а JPEG — лучший формат для фотографических изображений. В главе 24 я покажу, как выполняется сохранение изображения в различных форматах, и вы увидите, каким образом формат влияет на размер полученного файла.

Мы рассмотрим также сохранение изображения в гораздо более компактном WebP-формате и применение элемента `picture` для вывода таких изображений в браузерах, которые его поддерживают (для получения подробной информации см. «*Раздел третий: разметка адаптивных изображений*» главы 7). Все это ускоряет работу поддерживающих формат WebP браузеров и обеспечивает надежные резервные JPEG-или PNG-варианты изображений для прочих.

Определившись с форматом, самое время задуматься, как много версий для каждого изображения необходимо создать.

Нужны ли для вашего макета адаптивные изображения?

Следующий аспект, на который следует обратить внимание, — требует ли ваш макет применения адаптивных изображений.

Нет, хватит одного размера

Некоторые страницы — например, текстовые с небольшими иллюстрациями, вполне могут нормально работать только с одной версией каждого изображения, которая подходит для всех размеров экрана. Если это так, сохраните изображение в наиболее подходящем формате или экспортируйте его в этот формат, и вы почти закончили работу. Последний шаг — оптимизация изображения в целях его максимально возможного уменьшения. Методы оптимизации мы подробно рассмотрим в *главе 24*.

Да, мне нужно, чтобы каждое изображение сохранялось в нескольких размерах

Ваша адаптивный макет может потребовать от вас воспользоваться методикой создания адаптивных изображений, представленной в *главе 7*. Напомню, что термин «адаптивные изображения» относится к процессу предоставления изображений, адаптированных к пользовательской среде просмотра. Это включает предотвращение загрузки браузерами на небольших экранах большего количества изображений, чем им необходимо, а также предоставление изображений, достаточно больших, чтобы выглядеть четкими на дисплеях высокой плотности. Вы также можете представить альтернативные версии изображения на основе контента (принципов «художественного оформления») или воспользоваться преимуществами более новых форматов, таких, как WebP.

Как бы там ни было, но, пойдя по пути включения адаптивных изображений, вы сделаете свои страницы более интересными.

На вашем сайте много изображений?

Да, на моем сайте много изображений (подсказка: автоматизируйте их!)

Несмотря на то что было бы прекрасно иметь HTML-решение для вывода нужных изображений в правильно их отображающих браузерах, существующая система с ее стеками кода и необходимостью создавать несколько изображений представляется

нам несколько громоздкой. Если вы работаете с большим имиджевым сайтом, он может оказаться неуправляемым. Обработка изображений — это задача, которая требует автоматизации. И тут напрашивается иное решение — пусть этим займется сервер!

К счастью, существует множество инструментов и сервисов, как с открытым исходным кодом, так и на платной основе, которые позволяют серверу создавать «на лету» нужные версии изображений. Вы загружаете на сервер изображение с самым высоким качеством и наибольшим требуемым размером и позволяете ему сделать все остальное — при этом у вас пропадает необходимость создавать и хранить несколько версий каждого изображения. Некоторые сервисы выходят за рамки простого изменения размера и предоставляют возможности интеллектуального кадрирования изображений, добавления специальных эффектов (таких, как сепия) или иного преобразования изображений «на лету».

Некоторые системы управления контентом также имеют встроенные функции по изменению размера изображения. Есть и другой вариант — установить на вашем сервере программу или сценарий с открытым исходным кодом (например, Glide, glide.thephpleague.com). Имейте, впрочем, в виду, что сценарии, нуждающиеся в применении JavaScript, не являются идеальными решениями. Существуют также сторонние сервисы, которые на платной основе обеспечивают возможность изменения размеров изображения (например, Cloudinary.com, Akamai.com или Kraken.io). При работе с большими сайтами, отягощенными значительным количеством изображений стоит обратить внимание и на эти варианты.

Дополнительную информацию по этой теме вы найдете в статье «Image Resizing Services» Джейсона Григсби (Jason Grigsby) из Cloud Four (cloudfour.com/thinks/image-resizing-services/). Он поддерживает список текущих сервисов изображений, которые можете найти по ссылке в этой статье.

Нет, я могу создать изображения вручную

Если на вашем сайте имеется разумное количество изображений, которые обновляются по приемлемому графику, вы можете создавать их вручную на своем компьютере и загружать на сервер. Существует ряд новых инструментов, разработанных специально для создания веб-изображений, предоставляющих, в том числе, возможности одновременного создания нескольких версий изображения и оптимизации их в пакетном режиме. Даже наш старый добрый Adobe Photoshop развивается в направлении удовлетворения возрастающих потребностей производителей веб-изображений. Мы рассмотрим эти инструменты в следующей главе.

ФАВИКОНЫ

Раз речь у нас идет об изображениях, обратимся к еще одному связанному с сайтами изображению — фавикону. Фавикон — это маленький значок (пиктограмма), который отображается слева от заголовка страницы на вкладке браузера и в списках

закладок (рис. 23.18). Впервые такие пиктограммы появились как функции браузера Internet Explorer 5 еще в 1999-м году, после чего их быстро восприняли другие браузеры. Применение фавиконов необязательно, но они помогут пользователям идентифицировать и найти ваш сайт в длинном ряду вкладок или закладок. Фавиконы содержат мало деталей, однако имеющиеся определяют именно тот бренд, к которому относятся.



Рис. 23.18. Фавиконы для сайтов Adobe.com, W3C.org и Firefox.com на вкладках браузера Firefox

Некоторые веб-ориентированные устройства также задействуют для идентификации сайтов значки, напоминающие фавиконы. Например, Apple iPhone и iPad представляют сайты или веб-приложения, выводимые на их главном экране, соответствующими пиктограммами (называемыми *сенсорными значками*). Значки сайтов используются также плитками Microsoft, GoogleTV и другими системами.

В этом разделе рассказывается о том, что требуется для создания базового фавикона рабочего стола, а также полного набора всех нужных значков. Мы также рассмотрим инструмент, который выполнит за вас всю повторяющуюся работу.

Старомодные фавиконы для браузеров

Для компьютерных браузеров стандартная процедура создания фавикона достаточно проста:

- Сохраните ваш значок в формате ICO и назовите его файл **favicon.ico**.
- Поместите этот файл в корневой каталог сайта, чтобы браузерам было известно, где его искать.
- Третьего шага нет. Это все!

Это единственный метод создания фавиконов, который поддерживается браузером Internet Explorer 10 и более ранними его версиями, а также большинством других браузеров.

Существует несколько важных условий, которые необходимо учитывать при работе с файлами **favicon.ico**. Фавиконы следует создавать с разрешением 16×16 пикселов с дополнительной версией, имеющей разрешение 32×32 пикселя, — для четкого отображения их на устройствах с дисплеями Retina. Поскольку формат ICO позволяет сохранять в одном файле несколько изображений, вам потребуется для этого создать только один файл **favicon.ico**. К сожалению, большинство графических инструментов, включая Adobe Photoshop, не могут сохранять изображения в формате ICO, поэтому вам придется применять инструмент преобразования, который воспринимает PNG или JPEG и выдает ICO. В Сети имеется несколько бесплатных ICO-конвертеров (например, icoconverter.com и convertico.com), в окна которых для

преобразования в ICO файлы PNG или JPEG достаточно просто перетащить. Если вы работаете с Mac и желаете применять для конвертации более полнофункциональное средство, обратите внимание на инструмент Icon Slate от Kodlian (www.kodlian.com/apps/icon-slate), доступный в App Store за 5 долларов.

Как упоминалось ранее, если у вас есть файл **favicon.ico**, просто поместите его в корневой каталог сайта вместе с файлом **index.html**, и браузер автоматически его найдет. При этом нет никакой необходимости добавлять в файлы сайта какую-либо дополнительную разметку.

Полный набор фавиконов

Вы можете пойти обходным путем и создать полный набор фавиконов для представления вашего сайта на других устройствах. При этом можно сохранить все нужные значки в старом добром формате PNG и даже включить в них прозрачные области — полагаю, такой процесс вам будет более привычен.

Если ваши значки имеют формат PNG, необходимо связать их в разметке с вашими файлами с помощью элемента **link**, как показано в следующем примере, который добавляет сенсорный значок для iPhone с экраном Retina (см. врезку «Эффекты значков iOS»):

```
<link rel="apple-touch-icon-precomposed" sizes="120x120"
      href="apple-touch-icon-120x120.png">
```

В табл. 23.2 приведено большинство стандартных размеров значков, которые известны на момент подготовки книги.

Таблица 23.2. Наиболее популярные стандартные размеры фавиконов

Размер (в пикселях)	Назначение
32×32	Стандарт для большинства компьютерных браузеров
57×57	Стандартный экран iOS (iPod Touch, iPhone первого поколения)
76×76	Значок главного экрана iPad
96×96	Значок GoogleTV
120×120	iPhone Retina
128×128	Интернет-магазин Chrome
144×144	Плитка Metro IE10 для закрепленного сайта
152×152	Сенсорный значок iPad, значок Android (при необходимости автоматически уменьшается)

ЭФФЕКТЫ ЗНАЧКОВ IOS

По умолчанию iOS добавляет в файлы значков следующие визуальные эффекты, позволяющие привести их в соответствие с видом других значков главного экрана:

- скругленные углы;
- падающая тень;
- эффект «блестящего» отражения.

Если вам нравится значок в том виде, как он исходно создан, и вы хотите отключить спецэффекты, скажите об этом iOS, установив параметр **rel** в значение **apple-touch-icon-precomposed**, как и показано в приведенном примере. Если вам эти эффекты нравятся, присвойте параметру **rel** значение просто **apple-touch-icon**.

Табл. 23.2 (окончание)

Размер (в пикселях)	Назначение
167×167	Сенсорный значок iPad Retina
180×180	iPhone 6 плюс
196×196	Chrome для главного экрана Android
228×228	Значок Opera Coast

Создание фавикона

Чтобы иметь возможность полностью управлять качеством фавиконов, лучше всего создавать их вручную. У каждого к этому свой подход, но обычно рекомендуется начинать с векторного оригинала и экспорттировать его в требуемые размеры. Если же вы начинаете с растрового изображения, уменьшайте его постепенно и проверяйте качество на каждом шаге.

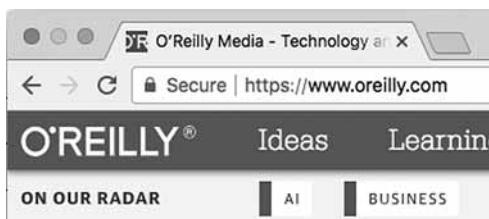


Рис. 23.19. O'Reilly Media использует в своем фавиконе фрагмент логотипа

Для очень маленьких значков (32- и, особенно, 16-пиксельных квадратиков) вам вероятно, потребуется выполнить точную попиксельную отладку, чтобы получить наилучший результат. Если ваш логотип имеет сложную структуру, рассмотрите возможность использования в фавиконе только определяющих его фрагментов, как это реализует O'Reilly Media (рис. 23.19).

Я настоятельно рекомендую вам книгу эксперта по фавиконам Джона Хикса (John Hicks) «The Icon Handbook (Five Simple Steps)», содержащую полезные практические советы по созданию фавиконов. Джон делится своими приемами эффективного дизайна фавиконов, а также вариантами поддержки наилучшего качества при небольших размерах.

Если создание всего набора фавиконов вручную кажется вам обременительным, проще применить генератор фавиконов, который создает необходимый значок на основе представленного оригинального изображения, а также генерирует весь необходимый код. Существует несколько подобных генераторов, но мне больше всех

нравится Favic-o-matic (www.favicomatic.com), показанный на рис. 23.20. Просто загрузите в него один файл PNG, размер которого превышает 300 пикселов, и инструмент выполнит все остальное.

ФАВИКОН ИЗ ОДНОЙ БУКВЫ

Если необходимо в качестве фавикона использовать только начальную букву названия вашего сайта, то **Favicon.io** — это удобный онлайн-инструмент, который генерирует значки на основе выбранного вами символа, шрифта и цвета.



Рис. 23.20. Генератор фавиконов и кода Favico-o-matic

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ

Описанные здесь конвертеры PNG и JPEG в ICO и инструменты генерации фавиконов представляют собой базовый набор инструментов для создания полных наборов фавиконов. Далее приводится несколько полезных ресурсов, с помощью которых вы при желании сможете ознакомиться с подробностями дизайна фавиконов, которые не вошли в эту книгу:

- «The 2017 Guide to FavIcons for Nearly Everyone and Everyone and Every Browser» от Emerge Interactive (www.emergeinteractive.com/insights/detail/The-Essentials-of-FavIcons-in-2017);
- «How to Make a Favicon» от Ника Петита (Nick Petit) из Treehouse (blog.teamtreehouse.com/how-to-make-a-favicon);
- Статья про фавиконы в Википедии (ru.wikipedia.org/wiki/Favicon).

ПОДВЕДЕМ ИТОГИ...

В этой главе мы рассмотрели много базовых вопросов. Если я хорошо выполнила свою задачу, у вас должна сформироваться серьезная основа по работе с веб-графикой и, в частности, понятие о том, где найти изображение и в каком формате его сохранить. Вы узнали о разрешении изображения, разрешении экрана и о работе с мониторами высокой плотности. Мы также сформировали стратегию формирования ваших требований к изображению, позволяющую вам сократить большое число вариантов обработки. И, конечно же, вы теперь знаете, как добавить на свой сайт фавикон.

В следующей главе вы получите практический опыт создания и оптимизации веб-изображений, поскольку мы приступим к изучению особенностей производственного процесса. Но сначала вам предстоит небольшая викторина.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Ответьте на следующие вопросы, чтобы узнать, получили ли вы общее представление о веб-графике. Ответы приведены в *приложении 1*.

1. Каково основное преимущество использования изображений с управляемыми правами?
2. Что означает «ppi»?
3. Что такое «индексированный цвет»? Какие форматы файлов его используют?
4. Сколько цветов в таблице цветов для 8-битного изображения? Если вам трудно ответить, определите максимальное количество цветов в 5-битном изображении.
5. Назовите два варианта действий, которые можно выполнить с GIF-изображениями, но нельзя сделать с JPEG-изображениями.
6. Назовите один вариант того, что вы можете сделать с GIF-изображением, но не можете сделать с PNG-изображением.
7. Назовите один вариант того, что вы можете сделать с PNG-изображением, но не можете сделать с GIF-изображением.
8. Сжатие «с потерями» JPEG является кумулятивным. Что это значит? Почему это важно знать?
9. В чем разница между двоичной и альфа-прозрачностью?
10. Выберите лучший формат файла растрового изображения для каждого из изображений, показанных на рис. ЦВ-23.21. Необходимо принять решение, просто просмотрев выведенные изображения и объяснить свой выбор. Для некоторых изображений может быть несколько возможных вариантов.

ГЛАВА 24

СОЗДАНИЕ ВЕБ-ИЗОБРАЖЕНИЙ

В этой главе...

- ▶ Выбор форматов веб-файлов при экспорте
- ▶ Двоичная и альфа-прозрачность
- ▶ Создание адаптивных изображений
- ▶ Инструменты и методы оптимизации изображений

В предыдущей главе шла речь собственно об изображениях, здесь же мы сосредоточимся на процессе их создания. Поскольку изображения обычно составляют 60–70% данных, представленных в Интернете, важно продуманно подходить к созданию изображений, ориентируясь на требования адаптивного дизайна и производительности сайтов. В этой главе мы снова встретимся с изображениями в растровых форматах: JPEG, PNG и GIF. Создание изображений в формате SVG требует иного подхода, и его мы рассмотрим в следующей главе. А эта глава посвящена пикселам!

Вы научитесь сохранять или экспортировать изображения в различных растровых форматах и создавать изображения с прозрачными областями. Вы узнаете несколько быстрых способов одновременного создания нескольких версий изображения для адаптивных макетов и дисплеев высокой плотности. Наконец, вы познакомитесь с инструментами и методами оптимизации, позволяющими сделать файлы изображений настолько маленькими, насколько это возможно.

А начнем мы с решения основных задач по созданию изображений, сохраняя их в соответствующих веб-форматах.

СОХРАНЕНИЕ ИЗОБРАЖЕНИЙ В ВЕБ-ФОРМАТАХ

Давайте посмотрим, как сохранять веб-изображения в Photoshop CC и GIMP. Вы можете спросить: «Почему именно эти два средства?». Обратитесь за ответом к небольшой врезке «*Почему именно Photoshop и GIMP?*», где приведены соответствующие пояснения. Если вы используете один из десятков других графических редакторов, процесс работы в них и терминология, скорее всего, будут аналогичны описанным здесь.

ПОЧЕМУ ИМЕННО PHOTOSHOP И GIMP?

Имеются десятки программ, предназначенных для создания изображений. Но при рассмотрении примеров в этой главе мы будем придерживаться программ Adobe Photoshop CC и GIMP, поскольку они представляют противоположные концы спектра графических программ. Очень важно, что обе эти программы доступны для macOS и Windows, и вы можете получить их для наших целей бесплатно.

Находящийся на одном конце спектра Photoshop — наиболее популярная среди профессиональных веб-дизайнеров программа для редактирования изображений, но ее использование обходится дорого, поскольку она доступна на условиях ежемесячной абонентской платы как часть пакета продуктов Adobe Creative Cloud. Однако для выполнения учебных упражнений вы можете загрузить ее бесплатную пробную версию на сайте по адресу: www.adobe.com/creativecloud/catalog/desktop.html.

Представляющий другой конец спектра GIMP (GNU Image Manipulation Program, программа обработки изображений на условиях лицензии GNU) — это графический редактор с открытым исходным кодом, обладающий многими функциями Photoshop, и при этом абсолютно бесплатный! То есть эта программа бесплатна на постоянной основе, а не просто в течение ограниченного испытательного периода. Загрузить эту программу можно с сайта gimp.org.

File Settings (Параметры файла) для выбора формата JPEG, PNG, GIF или SVG (рис. 24.1 ③). Выполняя команду **Export As**, Photoshop применяет агрессивные параметры сжатия для получения файла наименьшего размера.

Диалоговое окно **Export As** содержит специфичные для каждого формата параметры, и в окне предварительного просмотра изображение показывается в том виде, как будто эти параметры к нему применены:

В большинстве графических программ при сохранении или экспорте созданного изображения доступны форматы JPEG и PNG (если доступен только один формат PNG, то это будет формат PNG-24). Формат GIF доступен в более авторитетных программах, таких, как Photoshop, GIMP и PaintShop Pro, а формат WebP только начинает применяться.

В этом разделе подробно описывается процесс сохранения или экспорта изображений — для пользователей, незнакомых с работой в графических программах. Если вы в этом хорошо разбираетесь, можете просто быстро пролистать несколько страниц и перейти напрямую к выполнению *упражнения 24.1*.

Adobe Photoshop CC

В этой программе существует несколько способов сохранения графики в веб-форматах. Рассмотрим их далее по шагам.

Команда **Export As**

Популярным и наиболее рациональным способом сохранения файлов является использование команды **Export As** (Экспортировать как), доступной либо в меню **File | Export | Export As** (Файл | Экспорт | Экспортировать как), либо с помощью щелчка правой кнопкой мыши (Control-щелчок при работе с Mac) на слое для экспорта его содержимого (рис. 24.1 ④). В результате выполнения этих действий откроется диалоговое окно предварительного просмотра изображения и раскрывающееся меню

- для JPEG (рис. 24.1 **©**) можно выбрать уровень качества — чем выше качество, тем больше размер файла;
- для PNG (рис. 24.1 **©**) можно выбрать сохранение прозрачных областей изображения — тогда слгаженные края и тени будут сливаться с фоном. По умолчанию Photoshop экспортирует файлы в формат PNG-24, но если вы хотите экспортить изображение в формат PNG-8, сохраняя при этом несколько уровней прозрачности, можно выбрать параметр **Smaller File (8-bit)** (Меньший файл (8-разрядный));
- для GIF (рис. 24.1 **E**) никаких параметров не предусмотрено. Я полагаю, что этим Adobe показывает нам, что гораздо лучше работать с PNG.

ПРИМЕНЯТЬ ЛИ КОМАНДУ CONVERT TO sRGB? ДА!

В разделе **Color Space** (Цветовое пространство) диалогового окна Photoshop **Export As** (см. рис. 24.1 **B**, в самом низу справа) можно найти опцию **Convert to sRGB** (Преобразовать в sRGB). Вы наверняка захотите ее выбрать, потому что это цветовая коррекция, которая используется в Интернете. У Adobe есть собственное расширенное цветовое пространство RGB, поэтому вы получите непредсказуемые результаты, если не будете сначала конвертировать изображения в sRGB.

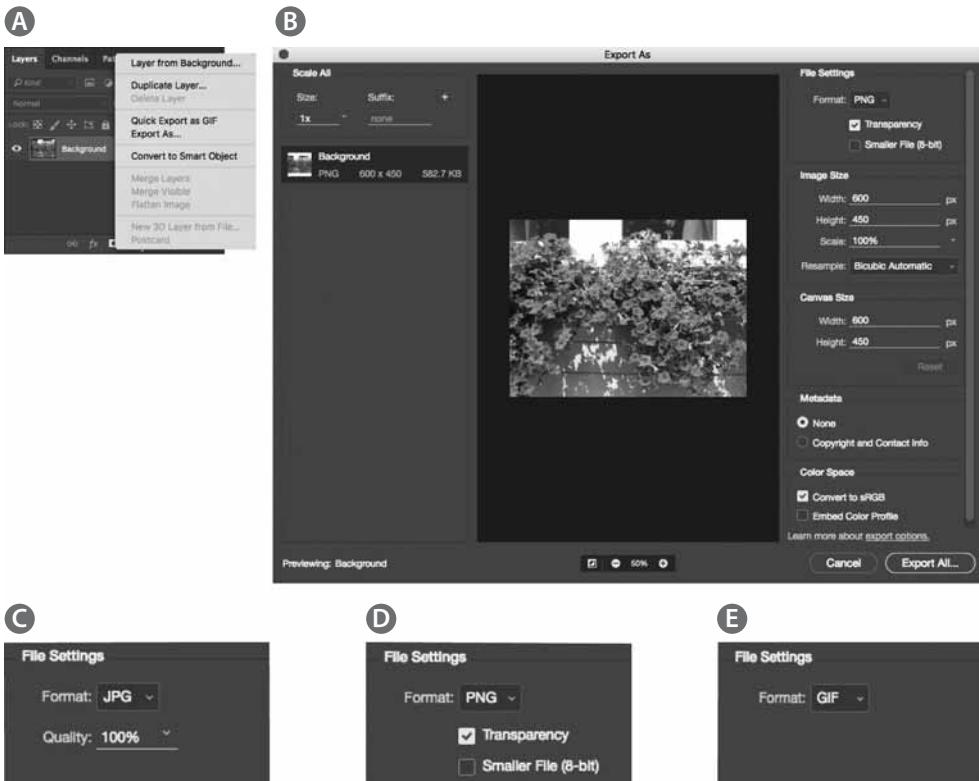


Рис. 24.1. Выбор типа файла в диалоговом окне Photoshop **Export As**

Диалоговое окно **Export As** (рис. 24.1 **B**) также содержит опции изменения размеров изображения. Это полезно при экспорте копий изображения с размерами, подходящими для различных вариантов макетов, не затрагивая при этом оригинала, имеющего натуральную величину.

Команда **Save As**

Вы также можете использовать команду **File | Save As** (Файл | Сохранить как) — для сохранения в новом формате находящегося в работе файла (вам будет предложен длинный список приемлемых веб-форматов). Как правило, выбрав команду **Save As**, вы получите дополнительные варианты сохранения — например, возможность включения чересстрочной развертки (interlacing), выбора палитры для GIF-файлов, преобразования изображения в прогрессивное изображение JPEG, но при этом утратите возможность сильного сжатия. В зависимости от характера изображения и типа файла размер файла изображения, полученного после выбора команды **Save As**, может быть в 10 раз больше, чем размер экспортированного аналога.

Команда **Save for Web** (устаревшая)

Команда **Save for Web** (Сохранить для Интернета) предоставляет настройки для ручной оптимизации размера файла, причем одновременно обеспечивается отслеживание полученного изображения в окне предварительного просмотра с учетом четырех параметров. Для изображений в формате GIF и PNG-8 можно уменьшить количество цветов (битовую глубину), частично устранить пикселизацию и подключить кроме всего прочего чересстрочную развертку. Для JPEG-изображения можно выбрать уровень качества, превратить его в прогрессивное либо оптимизированное или применить небольшое размытие, чтобы уменьшить его размер.

Начиная с 2014-го года, Adobe расценивает команду **Save for Web** как «устаревшую», и в будущих версиях Photoshop она исчезнет полностью без сохранения многих ее параметров. Дело в том, что уже сейчас доступны инструменты, позволяющие получить тот же уровень сжатия без дополнительной ручной работы. Если у вас имеется доступ к более ранней версии Photoshop, вы можете попробовать эту функцию, но не слишком ею увлекайтесь (как я!).

GIMP

В GIMP рабочие файлы по умолчанию сохраняются в «родном» для GIMP формате XCF. Чтобы выбрать нужный для сохранения файла формат, находясь в окне этой программы, выполните команду **File | Export As** (Файл | Экспортировать как). Самый быстрый способ получения нужного формата — сразу ввести в поле **Name** имя файла с соответствующим расширением: **jpg**, **png** или **gif**. К примеру, ввод `name.png` приведет к тому, что GIMP экспортит этот файл в PNG-формат. Впрочем, вы также можете для этого выделить тип файла в списке опций меню **Select File Type** (Выбрать тип файла), как показано на рис. 24.2 **A**.

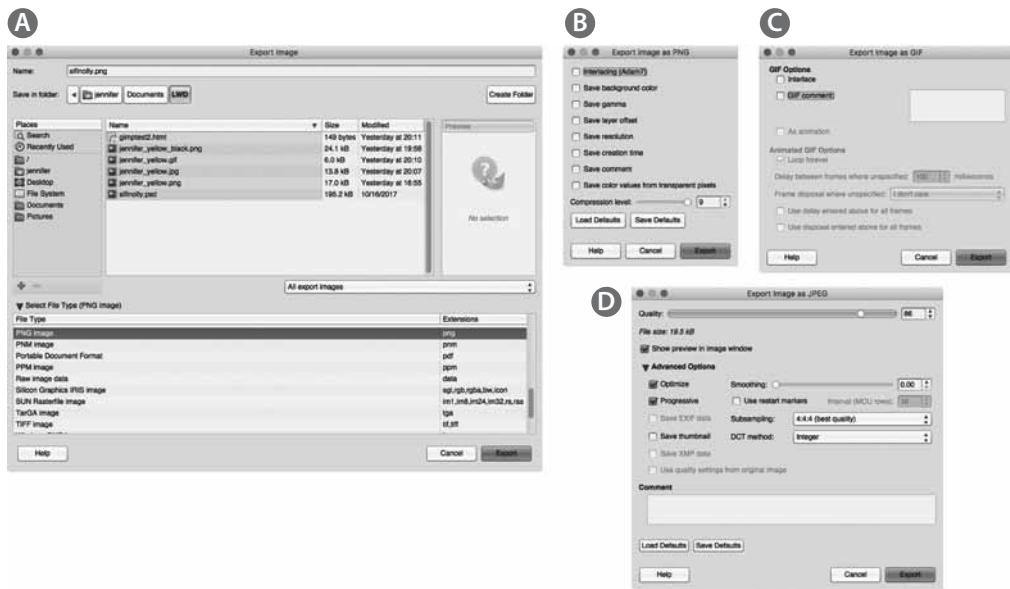


Рис. 24.2. Выбор формата файла в GIMP

После щелчка на кнопке **Export** (Экспорт) отобразится диалоговое окно с настройками, которые подходят для выбранного вами формата:

- для формата PNG (рис. 24.2 **Б**) отмените выбор всех параметров, поскольку многие из них хранят в файле ненужные метаданные, а некоторые имеют ограниченное использование при экспорте многослойного файла. Здесь же можно сделать изображение чересстрочным;
- сохранив файл в формате GIF (рис. 24.2 **С**), можно сделать изображение чересстрочным и добавить комментарии. Вы также можете сохранить его как анимацию, если слои настроены соответствующим образом;
- сохранив файл в формате JPEG (рис. 24.2 **Д**), вы можете поэкспериментировать с настройкой качества и возможностью просмотра результирующего уровня и размера файла в окне изображения (что рекомендуется). В разделе **Advanced Options** (Дополнительные параметры) можно оптимизировать JPEG-изображение (**Optimize**), сделать его прогрессивным (**Progressive**) и применить небольшое сглаживание (**Smoothing**) для уменьшения размера файла. В режиме **Subsampling** (Подвыборка) лучше всего выбрать настройку **4:4:4 (best quality)** — особенно если изображение имеет области, залитые однородным цветом, хотя в случае выбора настройки **4:2:2** создаются файлы меньшего размера. Результаты выбора этих настроек можно наблюдать в окне изображения.

Почему бы вам не опробовать способ создания веб-изображений, выполнив *упражнение 24.1?* Вы обнаружите, что выбранный формат серьезно влияет на размер файла. Если вы работаете в Photoshop или GIMP, можете следовать приведенным в упражнении инструкциям, если же вы используете какой-либо другой редактор

графики, есть большая вероятность того, что любой имеющийся у вас инструмент для создания изображений будет иметь аналогичные параметры для сохранения или экспорта изображений. Не забывайте о том, что можно всегда бесплатно загрузить пробную версию Photoshop и полнофункциональную GIMP.

До сих пор мы с вами рассматривали основы сохранения веб-изображений. А выполнив предложенное упражнение, мы обратим внимание на одно из основных свойств веб-изображений, с которым вам наверняка придется работать, — прозрачность.

УПРАЖНЕНИЕ 24.1. ФОРМАТЫ ФАЙЛОВ И ИХ РАЗМЕР

При выполнении этого упражнения мы будем экспортировать два изображения в различные форматы и сравнивать размеры файлов — так мы сможем увидеть, каким образом формат изображения влияет на размер файла. Использованные в упражнении файлы изображений (рис. 24.3) содержатся в материалах книги для этой главы, но вы также можете поэкспериментировать и со своими изображениями.

boats.png



asian.png



Рис. 24.3. Экспортируйте эти изображения в различные форматы, чтобы увидеть, как каждое из них влияет на размер файла

1. Откройте файл **boats.png** в выбранной вами программе и экспортируйте его в формат JPEG. Если ваш графический редактор не располагает функцией для экспорта, можно обратиться к опции **Save As**. Проверяйте, что выполнение этого упражнения всегда начинается с исходного изображения.

Используя Photoshop или GIMP, перемещайте ползунок **Quality** (Качество) вниз от значения, равного 100%, до значения, равного 0%, и обращайте внимание, каким образом изменяется качество изображения в окне предварительного просмотра или в окне изображения. Запишите размеры файлов, уровень качества которых соответствует значениям 100%, 60% и 10%. Сохраните результирующее JPEG-изображение с уровнем качества, равным 60%.

Если же в вашем графическом редакторе отсутствует возможность предварительного просмотра, вы можете экспортить три отдельных JPEG-файла с уровнями качества, установленными в значения 100%, 60% и 10%. Откройте полученные изображения в окне графической программы или в окне браузера для проверки качества и воспользуйтесь Finder или Проводником для проверки размеров полученных файлов.

2. Теперь экспортируйте исходное полноцветное изображение в формате PNG-24 (при работе с Photoshop не следует выбирать 8-битный вариант).
3. И в завершение преобразуйте изображение в режим индексированных цветов: **Image | Mode | Indexed** (Изображение | Режим | Индексированный), выбрав 256 цветов. Экспортируйте изображение в формат PNG-8. Работая с Photoshop, выберите параметр **Smaller File (8-bit)** (Меньший файл (8-битный)), а программа GIMP автоматически сохраняет индексированные цветные изображения в формате PNG-8. Пока оригинал еще находится в режиме индексированных цветов, снова выполните экспорт в формат GIF. По окончании можно вернуть изображение в режим RGB-цвета или закрыть его без сохранения.
4. Посмотрим, что получилось! В табл. 24.1 приведены итоговые размеры файлов изображений, полученных из файла *boats.png* с помощью Photoshop и GIMP. Обратите внимание, что размеры файлов различаются в зависимости от применяемого инструмента вследствие использования различных алгоритмов сжатия. Полученный вами файл, скорее всего, будет отличаться от моих вариантов, но не сильно.

Таблица 24.1. Итоговые размеры файлов для изображения из файла *boats.png* (в Кбайт)

Графическая программа	JPEG (100)	JPEG (60)	JPEG (10)	PNG-24	PNG-8	GIF
Photoshop CC	130,3	33,1	9,2	221	67,6	74,1
GIMP	179	20,9	7,4	225	73,7	80,3

Заключение: наилучшим форматом для изображения из файла *boats.png* служит JPEG, а качество порядка 60% обеспечивает наилучший баланс между качеством изображения и малым размером файла. Сохранение изображения в форматах PNG-8 и GIF дает файлы вдвое большего размера, которые притом выглядят неудовлетворительно. Конечно, «качество» оценивается нами субъективно. Иногда изображение столь важно, что качество 100% стоит дополнительного времени загрузки, но, как правило, можно удалить достаточно много байтов изображения при сохранении приличного уровня качества.

5. Отлично, теперь повторим все предыдущие шаги, но на этот раз с использованием изображения из файла *asian.png*. Экспортируйте исходное изображение в формат JPEG с различными настройками (или просто делайте заметки о размере файла на основе предварительного просмотра) и в формат PNG-24. После конвертации изображения в индексированные цвета поэкспериментируйте с количеством цветов, чтобы увидеть, сколько их можно убрать без ущерба для качества. Изображение хорошо выглядит при использовании 128 цветов? 64? 32?

Я действительно уменьшила палитру до 32 цветов, а затем экспортировала изображение в форматы PNG-8 и GIF. Вот полученные мною результаты (табл. 24.2).

Таблица 24.2. Итоговые размеры файлов для изображения из файла *asian.png* (в Кбайт)

Графическая программа	JPEG (100)	JPEG (60)	JPEG (10)	PNG-24	PNG-8 (32 цвета)	GIF (32 цвета)
Photoshop CC	22,7	8,7	3,3	14,8	4,2	4,3
GIMP	27,3	6,8	3,3	14,5	3,3	3,8

Заключение: для изображения из файла `asian.png` победителем оказался формат PNG-8 с уменьшенной цветовой палитрой. Конечно, размер JPEG-файла на 10% меньше, но качество просто ужасно! А вот формат PNG-8 дает наименьший размер файла, сохраняя однородные цвета без артефактов.

РАБОТА В РЕЖИМЕ RGB

Независимо от окончательного формата вашего файла работу по редактированию изображений всегда следует выполнять в режиме RGB (для нецветных изображений вполне подойдет режим работы с оттенками серого). Для проверки цветового режима изображения в Photoshop или GIMP выберите команду **Image | Mode** (Изображение | Режим) и убедитесь в том, что установлен флажок **RGB Color** (Цвет RGB).

Форматы JPEG и PNG-24 напрямую сжимают цветное изображение RGB. Если же файл сохраняется в формате GIF или PNG-8, изображение RGB необходимо преобразовать в индексированный цветовой режим либо вручную, либо как в составе экспортного процесса.

Индексированный цвет

Если нужно изменить существующий файл в формате GIF или PNG-8, прежде всего перед выполнением редактирования преобразуйте изображение в режим RGB. Это позволяет графической программе охватить весь спектр RGB. Если изменять размер исходного индексированного цветного изображения, результаты окажутся неутешительными, поскольку новое изображение будет ограничено цветами, взятыми из имеющейся таблицы цветов.

CMYK

Если у вас есть опыт создания графики для печати, вам более привычным может показаться режим CMYK (печатные цвета состоят из синих, пурпурных, желтых и черных чернил). Однако графика в режиме CMYK не используется для веб-графики и непригодна для нее, поэтому в начале процесса редактирования изображения перейдите в режим RGB.

РАБОТА С ПРОЗРАЧНОСТЬЮ

И формат GIF, и формат PNG позволяют делать части изображения прозрачными, в результате чего сквозь них могут просвечивать фоновые цвета или изображения станут прозрачными. В этом разделе мы подробно рассмотрим прозрачную графику, включая советы по ее созданию.

Учтите, что имеются два типа прозрачности. При работе с *двоичной (бинарной) прозрачностью* пиксели либо полностью прозрачны, либо полностью непрозрачны — как по команде включить/выключить. Файлы GIF и PNG-8 поддерживают двоичную прозрачность.

В случае *альфа-прозрачности* (или *прозрачности по альфа-каналу*) пикセル может быть полностью прозрачным, полностью непрозрачным или иметь в этом диапазоне до 254 уровней непрозрачности (всего имеется 256 уровней непрозрачности). Лишь форматы PNG, WebP и JPEG 2000 поддерживают истинную альфа-прозрачность.

Преимущество альфа-прозрачности формата PNG заключается в том, что при ее использовании происходит легкое смешивание основного цвета с любым фоновым цветом или изображением, как показано на рис. ЦВ-23.9. Формат PNG-8 также допускает несколько уровней прозрачности, но он делает это несколько иначе, и мы разберемся с этим чуть позже.

Итак, в этом разделе мы познакомимся с принципами, заложенными в основу каждого типа прозрачности, и узнаем, каким образом можно создавать прозрачные изображения с помощью GIMP и Photoshop.

Что такое двоичная прозрачность?

Как уже упоминалось, цвета пикселов для форматов PNG-8 и GIF сохраняются в индексированной таблице цветов. Прозрачность при этом просто рассматривается как отдельный цвет, занимая в таблице цветов одну позицию. На рис. ЦВ-24.4 показана таблица цветов в Photoshop для простого прозрачного формата GIF. Ячейка таблицы цветов, установленная прозрачной, обозначена в ней «шахматным» шаблоном. Когда такое изображение отображается в браузере, соответствующие этой позиции цветовой карты пиксели выводятся полностью прозрачными. Обратите внимание, что прозрачна только одна ячейка — остальные пиксели непрозрачны.

Как избежать ореола?

Если изображение имеет несколько уровней прозрачности, оно плавно сливается с фоном веб-страницы. Однако, когда мы имеем дело с двоичной прозрачностью, появляется риск того, что на зазубренных краях изображения появятся полосы пикселов, не соответствующих цвету, находящемуся позади них (рис. ЦВ-24.5). Подобная кромка известна как *ореол*, и это потенциальная опасность двоичной прозрачности.

Существует прием, позволяющий предотвратить появление ореолов при использовании двоичной прозрачности. Он заключается в смешивании зазубренных пикселов исходного изображения (например, *сглаженных* краев вокруг текста или фигур с заштрихованными краями) с цветом, максимально приближенным к цвету фона страницы. Многие инструменты для редактирования изображений, поддерживающие форматы веб-графики, предоставляют

РАБОТАЕМ С АЛЬФА-ПРОЗРАЧНОСТЬЮ В PNG

Вследствие неважной поддержки графическими инструментами и браузерами альфа-прозрачности форматов WebP и JPEG 2000, в этом разделе мы сосредоточимся на свойстве альфа-прозрачности в PNG.

ANTI-ALIASING (СГЛАЖИВАНИЕ)

Сглаживание — это небольшое размытие, применяемое к зазубренным краям растровой графики для формирования плавного перехода между цветами. Впрочем, сглаженные края все равно остаются несколько ступенчатыми. Однако, как бы там ни было, сглаживание текста и графики придает изображениям более профессиональный вид.

при сохранении или экспорте изображений способ выбрать смешиваемый цвет (также известного как *матовый*).

Некоторые программы используют для заливки зазубренных краев любой выбранный в качестве цвета фона цвет. Некоторые позволяют выбрать цвет смешивания вручную. Например, в устаревшей функции Photoshop **Save for Web** (Сохранить для Интернета) можно выбрать матовый цвет, если для изображения подключена прозрачность. Матовый цвет также применяется для заполнения любых прозрачных областей изображения при преобразовании изображения в формат JPEG. С другой

стороны, программа GIMP предотвращает появление ореолов, исключая вообще любые виды смешивания. Вы можете выбрать для отображения жесткие и ступенчатые края, либо для имитации слаженного края — размытый узор из цветных и прозрачных пикселов. Ни один из этих вариантов не дает безукоризненного результата, но зато устраняет ореолы.

ЭКСПОРТИРУЙТЕ PNG-8 С АЛЬФА-ПРОЗРАЧНОСТЬЮ

Интересно, что *предпочтительная* в Photoshop функция **Export As** автоматически заливает зазубренные края белым цветом и, как видно, не позволяет выбрать для GIF и JPEG матовый цвет. Однако, благодаря возможности экспорттировать PNG-8 с альфа-прозрачностью, вам не придется прибегать к Photoshop к матовому цвету.

Чтобы с помощью описанных здесь методов избежать ореолов, следует заранее знать значения RGB для цвета фона страницы, что позволит сопоставить его с матовым цветом. Если цвет страницы изменится, нужно вернуться к этому этапу и заново экспорттировать графику уже с новым цветом. Именно в этом случае проявляется преимущество альфа-прозрачности — можно изменять фон, и все будет отлично.

Что такое альфа-прозрачность?

Изображения в режиме RGB, такие, как JPEG и PNG-24, сохраняют цвета в отдельных каналах: для красного, для зеленого и для синего цветов. Для сохранения информации о прозрачности файлы PNG-24 добавляют еще один канал, называемый *альфа-каналом*. В этом канале каждый пиксель может отображать одно из 256 значений, которые при выводе изображения соответствуют 256 уровням прозрачности. Черные области маски альфа-канала прозрачны, белые области — непрозрачны, а серые имеют промежуточные значения прозрачности. Это можно представить себе как наложенное на изображение покрытие, которое указывает каждому находящемуся под ним пикселу, насколько он прозрачен (рис. ЦВ-24.6).

Альфа-прозрачность для изображений в формате PNG-8

Промежуточные уровни прозрачности присущи не только изображениям, сохраненным в 24-битном формате PNG. Прозрачность также поддерживают и файлы в формате PNG-8! И хотя они называются как *PNG8 + альфа* или *альфа-палитра* *PNG-изображений*, они не сохраняют информацию о прозрачности в отдельном альфа-канале, показанном на рис. ЦВ-24.6.

Эксперт по PNG Грег Рулофс (Greg Roelofs) хорошо объясняет «альфа-прозрачность» PNG-8 в следующей выдержке из его книги «PNG: The Definitive Guide», выпущенной издательством О’Рейли в 1999-м году:

Изображение альфа-палитры PNG — это просто изображение, палитра которого также связана с альфа-информацией, а не изображение палитры с полной альфа-маской. Другими словами, каждый пиксел соответствует записи в палитуре с красными, зелеными, синими и альфа-компонентами. Поэтому, если вы хотите иметь ярко-красные пиксели с четырьмя различными уровнями прозрачности, вам следует использовать четыре отдельные записи палитры для их размещения — все четыре записи будут иметь идентичные компоненты RGB, но значения их альфа будут отличаться. Если вы хотите, чтобы все ваши цвета имели четыре уровня прозрачности, вы фактически уменьшаете общее количество доступных цветов с 256 до 64.

Ни одна графическая программа (насколько мне известно) не отображает таблицы цветов PNG-8 с несколькими уровнями прозрачности, поэтому одну из них я для вас смоделировала (рис. ЦВ-24.7). Эта палитра основана на оранжевом круге с тенью, взятом из рис. ЦВ-23.9, с палитрой, уменьшенной до 16 цветов. Полученное изображение немного затемнено в области тени, но это не так заметно, когда оно отображается над фоновым рисунком. При этом экономия от уменьшения размера файла составляет 75%, и это того стоит.

Создание прозрачных файлов PNG и GIF

Самый простой способ получения изображения с прозрачными областями состоит в создании их таковыми с самого начала и сохранении при экспорте. Хотя и есть возможность «вылечить» имеющееся непрозрачное изображение, делая прозрачными области с однородной цветной заливкой, однако при этом трудно получить плавное смешение прозрачности с фоном без появления зазубренных краев.

ПРИЕМ С ИСПОЛЬЗОВАНИЕМ КОМАНДНОЙ СТРОКИ

Узнать, является ли PNG-изображение 8 или 24-битным, можно с помощью команды `file`. Используйте командную строку для перехода к содержащему файл изображения каталогу и введите команду: `file имя_файла`. Будет возвращено краткое описание файла, включающее его размеры, информацию о цвете и о том, является ли он че-реестрочным.

В следующем примере проверяется файл `super8bit.png`, представляющий собой экспортированный из Photoshop CC 8-битовый PNG-файл с альфа-прозрачностью. Слово `colormap` указывает, что это — индексированное цветное изображение:

```
$ file super8bit.png  
super8bit.png: PNG image  
data, 500 x 92, 8-bit  
colormap, non-interlaced
```

При проверке 24-битного PNG-файла `super24bitInt.png` с альфа-прозрачностью в описании вы увидите `RGBA`, что означает цветовую модель RGB:

```
$ file super24bitInt.png  
super24bitInt.png: PNG image  
data, 500 x 200, 8-bit/color  
RGBA, interlaced
```

Вместо чтения длинной и нудной инструкции я предлагаю создать изображение, состоящее из нескольких слоев, и сохранить прозрачные области в процессе выполнения **упражнения 24.2**. При создании многослойного файла в Photoshop или GIMP убедитесь, что фоновый слой отображается в виде серого «шахматного» шаблона и не заполнен цветом. Если цвет фона все равно останется, можно его выделить и удалить.

УПРАЖНЕНИЕ 24.2. СОЗДАНИЕ ПРОЗРАЧНЫХ ИЗОБРАЖЕНИЙ

Выполнив это упражнение «с нуля», вы приобретете опыт создания многослойного изображения с прозрачными областями. Я собираюсь сделать здесь все, пойдя самым простым путем, предложив вам методы, которыми вы сможете воспользоваться при выполнении более сложных и интересных проектов.

Поскольку Photoshop и GIMP исповедуют разные подходы, я собираюсь отдельно рассмотреть их применительно к каждой этой программе. Вы можете воспользоваться и другим каким-либо инструментом, лишь бы он предусматривал доступ к слоям с помощью интерфейса.

Photoshop CC (2018)

1. Начните с создания нового файла с размерами 250×250 пикселов и разрешением 72 ppi. В диалоговом окне **New Document** (Новый документ) найдите параметр **Background Contents** (Фоновое содержимое) и в раскрывающемся меню (рис. ЦВ-24.8 А) выберите параметр **Transparent** (Прозрачный). После щелчка на кнопке **Create** (Создать) вы должны увидеть квадрат, заполненный серым «шахматным» шаблоном, указывающим на прозрачный фон.
2. Выберите эллипсовидный инструмент **Marquee** (Выделение) и присвойте значение 10 параметру **Feather** (Растушевка). Нарисуйте в центре поля документа круг и залейте его цветом — вы получите форму с размытыми краями, сквозь которую видно изображение шахматной доски. Это все, что нужно для целей этого упражнения, но при желании вы можете добавить к рисунку и какие-нибудь дополнительные украшающие элементы.
3. Теперь вам следует выполнить команду **File | Export As** (Файл | Экспортировать как), выбрать **PNG** в разделе **File Settings** (Настройки файла) и удостовериться в том, что флажок **Transparency** (Прозрачность) установлен (рис. ЦВ-24.8 Б). Убедитесь также в том, что выбран параметр **Convert to sRGB** (Преобразовать в sRGB). Щелкните на кнопке **Export All** (Экспортировать все), присвойте файлу имя **circle24.png** и щелкните на кнопке **Export** (Экспортировать).
4. Сохраните файл также в формате PNG-8, выполнив команду **Export As | PNG | Transparency** (Экспортировать как | PNG | Прозрачность), но выберите на этот раз команду **Smaller File (8-bit)** (Меньший файл (8-битный)). Присвойте файлу имя **circle8.png** и щелкните на кнопке **Export**.
5. Исключительно для сравнения выберите функцию **Export As** снова, но на этот раз выберите в меню **File Settings** параметр **GIF**. В окне предварительного просмотра можно будет увидеть, что области, которые не являются непрозрачными на 100%, смешиваются с белым цветом (рис. ЦВ-24.8 Г), что не есть хорошо,

но в любом случае сохраните файл под именем `circle.gif`. Как можно видеть, функция **Export As** не предлагает способа изменения цвета заливки (матового) для GIF и JPEG.

Теперь, имея файлы с прозрачностью `circle24.png`, `circle8.png` и `circle.gif`, вы можете перейти к разд. «Как они выглядят?» этого упражнения.

GIMP

1. Выполнив команду **File | New** (Файл | Создать), создайте новый файл, установите размеры до 250 пикселов по ширине и высоте, задайте разрешение X и Y до 72,000 пикселов/дюйм. В раскрывающемся меню **Fill with:** (Заполнить) выберите вариант **Transparency** (Прозрачность). Комментарий **Created with GIMP** (Создано с помощью GIMP) можете удалить. Щелкните на кнопке **OK** — откроется новое окно изображения, заполненное серым «шахматным» фоном. Сохраните эту рабочую копию под именем `circle.xcf`.
2. А теперь перейдем к рисованию. На панели инструментов (**Toolbox**) выберите параметр **Ellipse Select Tool** (Эллипсоидный инструмент выделения), в разделе **Tool Options** (Параметры инструмента) выберите параметр **Feather Edges** (Растушевка краев) и установите значение радиуса, равным 10. Нарисуйте круг в окне изображения. Установите цвет переднего плана на удобный вам и перетащите цвет в круг для его заполнения (рис. ЦВ-24.9 **A**). Это все, что нужно сделать в рамках этого упражнения, но при желании вы можете добавить к рисунку и какие-нибудь дополнительные украшающие элементы.
3. Теперь займемся экспортом. Выполните команду **File | Export As** (Файл | Экспортировать как) и назовите этот файл `circle24.png` (рис. ЦВ-24.9 **B**). Расширение `png` указывает GIMP сохранить файл в формате PNG, а поскольку исходное изображение имеет формат RGB с прозрачными областями, GIMP создает 24-битный файл в формате PNG с альфа-прозрачностью. При работе с GIMP — это лучший вариант для применения прозрачности. В диалоговом окне **Export Image as PNG** (Экспортировать изображение как PNG) (рис. ЦВ-24.9 **C**) снимите все флагки.
4. Для сравнения посмотрим, как GIMP обрабатывает двоичную прозрачность. В GIMP для экспорта 8-битного изображения сначала необходимо преобразовать его в индексированный цвет, выполнив команду **Image | Color Mode | Indexed Color** (Изображение | Цветовой режим | Индексированный цвет). В нашем случае примените оптимальную палитру с 256 цветами. Оставьте флагок **Enable dithering of transparency** (Включить сглаживание прозрачности) неустановленным и щелкните на кнопке **Convert** (Преобразовать). Все мягкие края исчезли, а пиксели стали либо непрозрачными, либо прозрачными. Я рекомендую вам увеличить изображение до 200% (настройка масштабирования находится в нижней части окна), чтобы лучше увидеть зазубренные края (рис. 24.9 **D**, слева).
5. Ладно, верните файл в RGB (**File | Revert**) и преобразуйте его в индексированный цвет снова, только на этот раз щелкните на поле, находящемся рядом с параметром **Enable Dithering** (Включить сглаживание). Увеличив масштаб, вы сможете заметить, что GIMP создает рисунок, состоящий из сплошных и прозрачных пикселов, который имитирует сглаженные края круга (рис. ЦВ-24.9 **D**, справа).

Экспортируйте этот файл в формате PNG с именем `circle8.png`. Вы также можете сохранить его и в формате GIF.

Как это выглядит?

Теперь, когда у нас есть прозрачная графика, я предлагаю вам посмотреть на нее на следующей простой веб-странице с белым фоном:

```
<!DOCTYPE html>
<html>
<head>
    <title>Transparency test</title>
    <style>
        body {background-color: white;}
        p {text-align: center;}
        img {margin: 2em;}
    </style>
</head>
<body>
    <p>
         <!-- left -->
         <!-- center -->
         <!-- right -->
    </p>
</body>
</html>
```

АЛЬТЕРНАТИВА MATTE

Если у графического инструмента отсутствуют функция **Matte** (это, например, GIMP и Photoshop CC 2018), создайте новый слой под слоем «stack» (стек) и залейте его цветом фона вашей страницы. Когда изображение сглаживается при изменении на индексированный цвет, сглаженные края смешиваются с подходящим фоновым цветом. Просто выберите цвет фона, чтобы он был прозрачным при экспорте в GIF или PNG, и ваше изображение будет лишено ореола.

Если выбранный прозрачный цвет не подходит для используемого вами инструмента, можно скопировать важные части изображения, включая размытые края, а затем вставить их в новый файл для прозрачного изображения. Затем экспортируйте этот файл в формат GIF или PNG-8.

Все это достаточно большая работа, подразумевающая обработку каждого из изображений, поэтому в таком случае использование альфа-прозрачности является наилучшим выбором.

Файлы изображений и файл `transparent.html` расположены вместе с материалами в каталоге этой главы — если вы собираетесь смотреть на них вместе с нами. Впрочем, вы можете воспользоваться и созданной вами графикой. Если открыть файл этой страницы в окне браузера, на белом фоне графика выглядит почти одинаково (рис. ЦВ-24.10, *вверху*). Но если изменить цвет фона веб-страницы на бирюзовый (`background-color: teal;`), разница между альфа-прозрачностью и двоичной станет очевидной (рис. 24.10, *внизу*). Здесь также можно заметить ореол на GIF-изображении (рис. ЦВ-24.10, *внизу слева*). Экспортированные же в Photoshop версии PNG-8 и PNG-24 имеют плавную альфа-прозрачность.

Обтекание

В заключение этого упражнения: работая с Photoshop CC, экспортируйте прозрачные изображения в 8-битные PNG-изображения. Используя другие графические инструменты, возьмите для этого

файл в формате PNG-24 с альфа-прозрачностью, но следите за размером файла. Если файл неприемлемо велик, вы можете преобразовать его в PNG-8 + альфа-канал, применив один из инструментов, представленных в разд. «*Оптимизация изображения*» далее в этой главе. Вы можете также применить двоичную прозрачность и матовый цвет, соответствующий фону страницы. Если у вашего графического инструмента отсутствует функция **Matte** (Матовый цвет), обратитесь к врезке «*Альтернатива Matte*».

Закончив эксперименты с прозрачностью, вы можете перейти к знакомству с советами по созданию адаптивных изображений, приведенными в следующем разделе.

СОВЕТЫ ПО СОЗДАНИЮ АДАПТИВНЫХ ИЗОБРАЖЕНИЙ

Если вы создаете адаптивный сайт, вам не обойтись без адаптивных изображений. А когда речь идет о растровых изображениях, «адаптивный» фактически означает «имеющий несколько версий».

В главе 7 (см. «*Раздел третий: разметка адаптивных изображений*») рассматривались четыре сценария работы с адаптивными (отзывчивыми) изображениями, и теперь самое время освежить в памяти эту информацию (она ведь была приведена сотни страниц назад). Разрабатывая эти сценарии, я имела в виду, что если мы ранее представляли посетителю сайта лишь одно изображение, то в наших нынешних условиях поднялись на качественно новый уровень и теперь можем:

- предоставлять наборы изображений различных размеров для применения в адаптивных макетах на экранах с различными *размерами областей просмотра*;
- предоставлять версии изображения с различной степенью детализации в зависимости от размера и ориентации устройства (этот подход также известен как сценарий использования *принципов художественного оформления*);
- предоставлять крупномасштабные изображения, которые выглядят четкими на экранах с *высокой плотностью*;
- осуществлять поддержку *альтернативных форматов изображений*, которые сохраняют то же изображение при гораздо меньших размерах файлов.

В этом разделе представлены инструменты, советы и общие стратегии для создания (или автоматизации!) изображений, которые необходимы для первых трех сценариев. Альтернативные форматы изображений были рассмотрены нами в главе 23.

Изображения для адаптивных макетов

В первом сценарии определяется диапазон размеров изображения, которое выбирается браузером в зависимости от размера области просмотра. В HTML можно

ЕЩЕ О ФОРМАТЕ SVG

Адаптивные изображения в формате SVG будут рассмотрены в главе 25.

сделать это, используя свойство `srcset` с `w`-селектором, который поддерживает точную ширину изображения (в пикселях), и атрибут `sizes`, сообщающий браузеру, насколько большим будет выглядеть изображение в макете.

Следующий пример должен быть вам знаком:

```

```

Здесь для конкретного элемента `img` предоставлены JPEG-изображения клубники, ширина которых равна 240, 480 и 672 пикселя. Другие макеты могут нуждаться в меньшем или большем числе точек прерывания для каждого изображения. Первый вопрос, который можно задать при создании изображений для применения в адаптивных макетах: «Сколько изображений следует создать?». Это хороший вопрос, на который нет простого ответа.

Начните с определения самых маленьких и самых больших размеров, которые может иметь изображение. Затем решите, сколько изображений промежуточных размеров удобно применить, чтобы избежать лишних загрузок. Если диапазон размеров не слишком велик, можно сделать вывод, что создания маленькой, средней и большой версий изображения вполне достаточно, и это лучше, чем ничего. Если же между крайними значениями размеров имеются значительные различия, может возникнуть необходимость во введении дополнительных точек прерывания. А если разница мала, вполне достаточно и одного размера изображения.

Изменяйте их вручную

Если необходимо получить лишь несколько версий изображения, вполне приемлемо изменять его размеры способом экспорта. На рис. 24.11 показаны параметры изменения размера в диалоговом окне Photoshop CC **Export As**, но аналогичные настройки можно найти и в других программах. Кроме того, вы можете применять инструмент **Image Size** (Размер изображения) для изменения размера изображения в ручном режиме перед его сохранением или экспортом. При этом вы получите возможность предварительно вносить в изображение корректизы (например, повышать его резкость).

СОВЕТЫ ПО ИСПОЛЬЗОВАНИЮ PHOTOSHOP

Если вы являетесь пользователем Adobe Photoshop (или намерены им стать), обратите внимание на советы экспертов по использованию Photoshop в рабочем процессе, приведенные в книге Дэна Роуза (Dan Rose) «Responsive Web Design with Adobe Photoshop» (Adobe).

Учтите, что рекомендуется начинать обработку с изображения самого большого для вашего графического редактора размера, а затем уменьшать размер изображения до получения нужного размера. Увеличение размера изображения обычно приводит к потере резкости.



Рис. 24.11. Изменение размера изображений вручную (как показано здесь на примере Photoshop CC) — подходящий вариант, если имеется разумное количество изображений для обработки

ИЗМЕНЕНИЕ РАЗМЕРА ИЗОБРАЖЕНИЯ

Изменение размера чрезвычайно большого фотографического изображения до небольшого размера, подходящего для макета веб-страницы, может привести к потере его резкости, если выполнить это за один шаг, например, в процессе экспорта. Я полагаю, что лучшие результаты получаются при небольшом уменьшении размера за один прием при усилении резкости изображения после каждого прохода. Например, если исходное изображение имеет ширину 4000 пикселов, а уменьшить его нужно до 250 пикселов, я начинаю с изменения его размера вручную на 50%, а затем с помощью инструмента **Sharpen** (Повышение резкости) поднимаю его резкость. Затем повторяю эти шаги, пока не получу изображение нужного размера. И хотя при этом приходится выполнять довольно большой объем работ, но это стоит затраченных усилий, если вас удовлетворят полученные результаты.

Генерирование изображений на основе размера файла

Если ваше изображение представляется в широком диапазоне размеров, может потребоваться больше точек прерывания, чем для «маленькой, средней и большой» его версий. В этом случае лучше определяться с количеством создаваемых версий изображения на основе размеров их файлов, а не размеров собственно изображений, заданных в пикселях (см. врезку «Дополнительные источники»).

Имейте в виду, что основной целью применения адаптивных изображений, выводимых в зависимости от размера области просмотра, является ограничение загрузки данных. Помните также о том, что именно браузер делает окончательный выбор файла изображения для той или иной пользовательской среды просмотра — дизайнеры лишь предоставляют ему варианты изображения с помощью своей адаптивной разметки. Мы вполне можем доверять браузеру правильный выбор, увеличивая или уменьшая по мере необходимости предоставляемый ему диапазон вариантов изображений.

Для формирования точек прерывания на основе размера файлов изображений вы создаете набор изображений с размерами файлов, увеличивающимися с фиксированными значениями приращений, — например: 20 Кбайт, 40 Кбайт или 80 Кбайт.

ДОПОЛНИТЕЛЬНЫЕ ИСТОЧНИКИ

Я рекомендую две статьи, где рассматриваются подходы к созданию адаптивных точек прерывания на основе размера файла, причем изложение там более подробное, чем у меня здесь:

- статья Надава Софермана (Nadav Soferman) «Responsive Image Breakpoints Generator, A New Open Source Tool», Smashing Magazine (www.smashingmagazine.com/2016/01/responsive-image-breakpointsgeneration/). Статья знакомит с генератором изображений, речь о котором идет в этом разделе далее, и предоставляет большой объем справочной информации об этом подходе;
- статья Джейсона Григсби (Jason Grigsby) «Responsive Images 101, Part 9: Image Breakpoints», доступная на сайте cloudfour.com/thinks/responsive-images-101-part-9-image-breakpoints/. В этой статье среди других решений рассматривается подход к созданию точек прерывания, основанный на «бюджете производительности».

При этом охватываются все возможности адаптивного дизайна и более точно регулируется объем загружаемых данных. Конечно, такой подход требует большой дополнительной работы, и может оказаться, что для сайта с большим количеством изображений его невозможно выполнить вручную.

К счастью, имеется инструмент, генерирующий изображения. Программа Responsive Image Breakpoints Generator от Cloudinary (responsivebreakpoints.com) позволяет загрузить в нее большое изображение, установить максимальные/минимальные размеры генерируемых изображений, размер шага и максимальное количество изображений и автоматически генерировать все изображения. На рис. 24.12 показано, как использовался этот инструмент для создания изображений клубники с шагом 20 Кбайт. Ко времени, когда эта книга попадет к вам в руки, таких инструментов появится больше, поэтому я рекомендую выполнить соответствующий поиск в Интернете, чтобы найти появившиеся новинки.



Рис. 24.12. Программа Responsive Image Breakpoints Generator от Cloudinary (responsivebreakpoints.com) создает пользовательские файлы изображений

Использование принципов художественного оформления

Для некоторых изображений недостаточно простого изменения размера в соответствии с разметкой. Может возникнуть необходимость в подрезке или изменении самого изображения, чтобы оно успешно отображалось как на экране смартфона, так и на мониторе настольного компьютера. Подобный подход к организации вывода адаптивных изображений известен как «художественное оформление». В главе 7 было приведено полное объяснение этого подхода и примеры выбора изображений на его основе, но в качестве быстрого напоминания рассмотрим следующий сценарий для элемента `picture`:

```
<picture>
  <source media="(min-width: 1024px)" srcset="icecream-large.jpg">
  <source media="(min-width: 760px)" srcset="icecream-medium.jpg">
  
</picture>
```

Если вы хотите получить полный контроль над тем, что демонстрирует изображение каждого размера, вам необходимо вручную подготовить и экспорттировать каждое такое изображение в вашем любимом графическом редакторе. Вполне вероятно, что каждая художественно-ориентированная версия изображения должна будет формироваться в нескольких размерах в зависимости от выбранных вами точек прерывания. И это прекрасно, если только для этого не потребуется обработать слишком много таких изображений.

И вот что достойно удивления! Программа Cloudinary предлагает способ автоматизации также и при обработке изображений на основе принципов художественного оформления. Для этого используются инструменты, находящиеся в разделе **Art-direction - Image aspect-ratio and view-port ratio**, расположеннном в правом нижнем углу окна программы Responsive Image Breakpoints Generator (см. рис. 24.12, слева), — с их помощью можно задавать пропорции изображений для настольных компьютеров, ноутбуков, планшетов и смартфонов. Инструмент Cloudinary выполняет сложный анализ изображений, в том числе обнаружение краев, распознавание лиц и определение визуальной уникальности изображенного, что позволяет ему найти наиболее важные части изображения. Окончательное изображение обрезается с тем, чтобы включить в него все визуально «горячие» точки. Для получения дополнительной информации о том, как это реализуется, ознакомьтесь со статьей Эрика Портиса (Eric Portis) «Automating Art Direction with the Responsive Image Breakpoints Generator», которая доступна на сайте по адресу: www.smashingmagazine.com/2016/09/automating-art-direction-with-the-responsive-image-breakpointsgenerator/.

Некоторые сервисы хостинга и автоматизации изображений также предлагают функцию распознавания лиц и улучшают качество создаваемых ими изображений. Если вы приобретаете подобную услугу, проверьте, доступна ли там функция интеллектуальной обрезки.

Изображения для экранов высокой плотности

Если вы хотите, чтобы изображение выглядело максимально четким на экранах высокой плотности @1,5x, @2x и @3x, его необходимо создать достаточно большим, чтобы охватить пиксели устройств с самой высокой плотностью.

ОСОБЫЕ ТРЕБОВАНИЯ К ДИСПЛЕЯМ ВЫСОКОЙ ПЛОТНОСТИ

Для уточнения особых требований к дисплеям высокой плотности, обратитесь к главе 7, где я впервые представила соотношение устройств и пикселов, а также разметку для вывода изображений под конкретные значения плотности дисплеев. Обратите также внимание на обсуждение изображений и разрешений экрана в главе 23.

бражения на дисплеях высокой плотности, использует атрибут `srcset` в элементе `img` с `x`-дескриптором, который задает целевую плотность экрана для каждого изображения:

```

```

К счастью, разработчики программ генерации изображений учли этот подход и начали встраивать подобные функции в свои инструменты, облегчающие одновременный вывод нескольких версий изображения для экранов с высокой плотностью.

Экспорт нескольких версий изображений высокой плотности

Программы Photoshop CC 2018, Sketch, Illustrator и Affinity Designer — вот четыре предназначенные для дизайнеров экрана инструмента, которые позволяют выполнять одновременный экспорт изображений для разных масштабов. Они обеспечивают хорошую и точно рассчитанную экономию времени. Если вы используете какой-либо другую графическую программу, проверьте, имеется ли у нее подобная опция (обычно она доступна в разделе, где находятся параметры экспорта). Позднее в этом разделе главы я поделюсь некоторыми стратегиями, которые обеспечивают получение изображения высокого качества даже при использовании больших увеличений.

- *Adobe Photoshop CC 2018* — Photoshop (рис. 24.13, *вверху*) в верхнем левом углу диалогового окна **Export As** позволяет открывать шкалу масштабирования. Для экспорта всей области рисования выберите команду **File | Export | Export As** (Файл | Экспорт | Экспортировать как). Вы также можете экспортить определенный элемент, щелкнув правой кнопкой мыши (щелкнув мышью при нажатой клавише `<Control>` на Mac) на соответствующем названии имени слоя и выбирая из раскрывающегося меню параметр **Export As** (Экспортировать как). В разделе

Так, если нужно, чтобы изображение в вашем макете было шириной 300 пикселов, потребуется его версия шириной 300 пикселов для стандартных дисплеев, версия шириной 600 пикселов для дисплеев 2x и версия шириной 900 пикселов, ориентированная на дисплеи 3x.

Для примера следующий сценарий, предусматривающий вывод версий изо-

Photoshop CC 2018

Щелкните правой кнопкой мыши на слое для экспорта одного элемента слоя за один раз.

Выберите формат файла в разделе **File Settings** (Параметры файла), расположенным на правом скриншоте справа. Откройте шкалу масштабирования в разделе **Scale All** (Масштабировать все), расположенным на правом скриншоте слева.



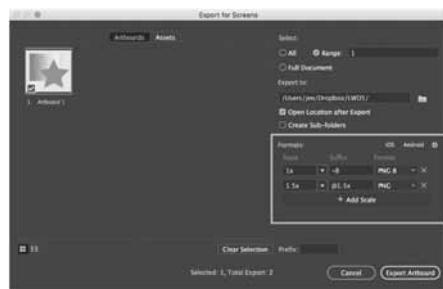
Illustrator CC

Выберите в меню **File** (Файл) команду

Export for Screens

(Экспорт для экранов) для экспорта всей области рисования.

Выберите формат файла и добавьте шкалы масштабирования в разделе **Formats** (Форматы).



Экспортируйте по одному элементу за раз из диалогового окна **Asset Export** (Экспорт ресурсов). Добавьте ресурсы, перетаскивая их на панель.

Sketch

Выберите элемент для экспорта и щелкните на кнопке **Make Exportable** (Сделать экспортируемым) в правом нижнем углу окна приложения.

В настройках **Export** (Экспорт) выберите форматы файлов и добавьте значения масштабов.



Affinity Designer

Выберите значок **Export Persona** (Экспорт персоны), щелкнув на соответствующем значке. Щелкните на значке стрелки рядом с фрагментом, который необходим для отображения настройки формата и масштаба.



Рис. 24.13. Новые инструменты дизайна позволяют экспортировать сразу заданное количество изображений высокой плотности, имеющих несколько размеров

Scale All (Масштабировать все) щелкните на кнопке + для экспорта большего числа значений масштаба. Небольшие стрелки «вниз» открывают меню стандартных шкал (1x, 5x, 3x и т. д.). Щелкните на символе корзины для удаления шкалы масштабирования. Если щелкнуть на кнопке **Export All**, все изображения сформируются сразу и будут иметь суффикс **@nx**.

СОГЛАШЕНИЕ «@NX»

Соглашение «@nx» — @1x, @2x и т. д. — установлено для библиотеки разработчика Apple iOS. Похоже, что оно перешло и в мир Интернета.

- **Adobe Illustrator CC** — при работе в Illustrator (рис. 24.13, *в центре*) для экспорта всей области рисования выберите команду **File | Export | Export for Screens** (Файл | Экспорт | Экспорт для экранов). В правом столбце можно найти опцию **Add Scale** (Добавить шкалу). Вы также можете экспортировать отдельные ресурсы (такие, как значки и прочие элементы) с помощью панели **Assets Export** (Экспорт ресурсов), которая имеет свои настройки экспорта. Для получения доступа к этой панели выберите команду **Window | Assets Export** (Окно | Экспорт ресурсов). Просто перетащите элементы на панель, и они будут готовы к работе. Диалоговое окно **Export As** также поддерживает доступ к отдельным ресурсам с помощью вкладки **Assets**, но они сначала должны быть добавлены на панель **Assets Export**. Один щелчок на кнопке **Export** (Экспорт) — и все масштабируемые ресурсы экспортируются за один раз!
- **Sketch** (только для Mac) — это инструмент для разработки интерфейсов веб-сайтов и приложений, популярность которого растет. При работе в Sketch (рис. 24.13, *внизу слева*) выберите область рисования или элемент страницы и щелкните в нижнем правом углу окна **Sketch** на значке + рядом с кнопкой **Make Exportable** (Сделать экспортируемым) — отобразится панель **Export**. Выберите формат файла и щелкните на значке + для добавления дополнительных значений масштабирования, которые будут учтены при экспорте.
- **Affinity Designer** — эта программа (рис. 24.13, *внизу справа*) имеет режим экспорта **Persona**, в котором вы получаете доступ ко всем настройкам экспорта. Создайте фрагменты для элементов, которые необходимо экспортировать. Выберите режим **Export Persona** (Экспорт Персона) с помощью меню **Affinity Designer | Export Persona** или щелчком на значке **Export Persona**. Выделите на панели **Slices** (Срезы) срез или срезы, которые необходимо экспортировать, затем щелкните на стрелке, находящейся слева от имени среза, чтобы отобразить параметры экспорта, включающие формат файла, а также возможность добавлять шкалы с помощью значка +. Когда все будет готово, щелкните на кнопке **Export Slices**.

Проблема, на которую следует обращать внимание при работе со всеми этими инструментами, заключается в том, что разработка ведется при стандартном (@1x) разрешении, а экспортанные версии @2x и @3x окажутся намного *больше*, чем они представлены в вашем рабочем документе. Это должно вас насторожить, поскольку удвоение или утройство размеров изображений обычно приводит к их размытию и потере резкости. Однако имеются способы устранения этой проблемы, которые мы сейчас и рассмотрим.

Работа с масштабом @1x

Даже если вы создаете версии изображений с высокими значениями разрешений, дизайн я настоятельно рекомендую выполнять в масштабе @1x. Другими словами, заданные в пикселях размеры в вашем рабочем документе (будь-то в Photoshop, Sketch или каком-либо другом инструменте) должны соответствовать пикселям разметки в вашем макете.

При работе в Photoshop и других подобных инструментах для редактирования изображений масштаб @1x эквивалентен разрешению 72 ppi. Преимущества работы с применением масштаба @1x следующие:

- проще указывать в ваших рабочих документах размеры шрифтов, величины длины и т. п. без необходимости делить все на два. Если вы работаете с масштабом @2x и желаете применить шрифт 16 пунктов, вам придется в вашем графическом документе увеличить его до 32 пунктов. А если вы захотите, чтобы в макете имелись 10-пиксельные отступы, надо будет присвоить им значения, равные 20 пикселам, и так далее;
- возможности, привязанные к пикселям, более надежны при использовании масштаба @1x. Привязка к пикселям — это способ сохранить резкость краев в мелких элементах, таких, как значки;
- размеры графических файлов, созданных под масштаб @1x, значительно меньше, что повышает производительность при работе на вашем компьютере. Сложные файлы с большим количеством областей рисования и слоев при масштабе @2x могут серьезно замедлить работу;
- создается более реалистичное представление о том, сколько пространства у вас имеется для размещения элементов. Пространство дизайна @2x может создавать впечатление, что такого места достаточно, но элементы окажутся слишком маленькими и тесно расположеными, когда уменьшатся на 50% для вывода на дисплеях @1x.

Начните с векторной графики, если это возможно

Один из способов сохранить качество при увеличении масштаба вашего дизайна — всегда, когда это возможно, использовать векторные изображения. Как мы говорили ранее, векторные изображения масштабируются без потери качества, поэтому они отлично подходят для веб-дизайна и дизайна приложений.

Многие новые инструменты дизайна пользовательского интерфейса для экранов и веб-интерфейсов — такие, как Sketch, Affinity Designer и Adobe XD — по умолчанию предназначены для работы с векторной графикой, поэтому у вас не возникнет проблем с выводом более крупных версий созданных с их помощью элементов (то же самое относится и к векторным изображениям, созданным в Adobe Illustrator). Если вы предпочитаете работать в Photoshop, всюду, где это возможно, используйте для создания общих элементов веб-страницы типа кнопок, значков и рисунков соответствующие векторные инструменты — такие, как фигуры, контуры и импортированные векторные смарт-объекты.

НЕКОТОРЫЕ ПРЕДПОЧИТАЮТ @2Х

С другой стороны, некоторые дизайнеры предпочитают работать именно с @2x и сокращать вдвое все для стандартных дисплеев, особенно, если работа ведется с оглядкой на дисплеи Retina. Дэн Родни (Dan Rodney) — один из этих дизайнеров, и вы можете ознакомиться с его аргументацией в пользу дизайна @2x по адресу:

www.danrodney.com/blog/designing-retinaweb-graphics-in-photoshop-shouldyou-work-at-1x-or-2x/.

Внедрение растровых изображений с большим масштабом

Для сохранения качества фотографий и других растровых элементов страницы при больших значениях масштаба начните с исходного изображения, которое должно быть, как минимум, столь же крупно, как ваш самый большой масштаб. Например, если известно, что ваша версия @3x имеет в ширину 2880 пикселов, исходное изображение должно быть таким же широким или даже шире.

При работе в Illustrator, Sketch и Affinity Designer размещение исходного изображения высокого разрешения в области рисования @1x и изменение его размера в соответствии с требованиями разметки предоставляет программе всю пиксельную информацию, необходимую для экспорта высококачественных крупномасштабных ресурсов.

Чтобы воспользоваться преимуществами полного разрешения изображения при работе в Photoshop CC, вы можете добавить изображение в макет в виде связанного

смарт-объекта. Смарт-объект подобен заполнителю для изображения в макете @1x, при этом оригинал с высоким разрешением представляется отдельно. Когда наступает время экспортировать изображение в различных масштабах, Photoshop ссылается на версию с высоким разрешением, и вы получаете экспортированный файл с полноразмерным изображением. Для размещения изображения в качестве смарт-объекта выполните команды **File | Place Linked** (Файл | Поместить связанный) и измените размер изображения в соответствии с вашим дизайном.

ОШИБКА PHOTOSHOP CC 2018

Когда готовилась эта книга, в Photoshop CC 2018 была замечена ошибка, которая не позволяет использовать в качестве смарт-объектов изображения в формате JPEG. При связывании крупномасштабного изображения JPEG Photoshop игнорирует его и увеличивает масштаб снимка изображения в текущем файле. Обходной путь состоит в преобразовании изображения JPEG с высоким разрешением в файл формата PSD еще до его добавления в качестве смарт-объекта. Adobe знает об этой ошибке, поэтому, надеюсь, она будет исправлена в следующем выпуске программного продукта.

Да здравствует автоматизация!

Я уже упоминала об этом в разд. «Стратегия работы с изображениями» предыдущей главы, но стоит повторить: если сайт перегружен изображениями, подумайте об использовании серверного программного обеспечения, которое автоматизирует процесс создания адаптивных изображений. Я не могу не согласиться с мыслями, выраженными Джейсоном Григсби (Jason Grigsby) в его статье: «Humans shouldn't be doing this» («Люди не должны делать это»), но вы можете думать иначе, если у вас есть склонность к выполнению повторяющихся задач.

Вы можете установить программное обеспечение на собственный сервер или, что удобнее, обратиться к стороннему сервису, который предоставляет услуги по управлению размещенными изображениями. Напомню, что самыми популярными в настоящее время сервисами такого рода являются Cloudinary (cloudinary.com), Akamai (akamai.com) и Kraken.io (kraken.io).

Я надеюсь, что вы определились с подходами к совершенствованию своего рабочего процесса по части создания нескольких версий изображений для адаптивных макетов. Или, возможно, вы просто решили позволить справиться с этим серверу? Как бы там ни было, давайте перейдем к последней теме нашего глубокого погружения в производство графических ресурсов — оптимизации.

ПОЛЕЗНЫЕ РЕСУРСЫ

Джейсон Григсби (Jason Grigsby) поддерживает доступную по адресу: tinyurl.com/pmpbvyzj таблицу, содержащую информацию о сервисах по изменению размера изображения. Обратите также внимание на связанную с этой тематикой статью «Услуги по изменению размера изображения» (cloudfour.com/thinks/imageresizing-services/).

ОПТИМИЗАЦИЯ ИЗОБРАЖЕНИЙ

Поскольку веб-страница публикуется в Сети, для передачи конечному пользователю ее необходимо пропустить через линии связи в виде небольших пакетов данных. Отсюда интуитивно понятно, что на передачу больших объемов данных потребуется больше времени. И угадайте, какая часть стандартной веб-страницы содержит целые «залежи» байтов — правильно, изображения.

Это формирует противоречивое отношение к изображениям в Интернете. С одной стороны, изображения оживляют веб-страницу по сравнению с просто текстовым ее вариантом, так что возможность выводить изображения становится одним из факторов, способствующих успеху вашего сайта. С другой стороны, изображения действительно испытывают терпение пользователей при низкой скорости интернет-соединений, а также вызывают повышенный расход оплачиваемого трафика у пользователей мобильных устройств.

Если обратиться к блок-схеме, представленной на рис. 23.16, можно заметить, что все пути заканчиваются на слове «Оптимизируйте». Наличие файлов изображений как можно меньших размеров очень важно для быстрой загрузки сайтов, поэтому все веб-дизайнеры и разработчики должны иметь под рукой несколько приемов оптимизации изображений.

Как показано в *упражнении 24.1*, выбор подходящего формата файла — первая линия защиты от «раздутых» файлов, первая, но не последняя. При экспорте изображений в вашем графическом редакторе вы можете выкинуть из них множество лишних данных.

Оптимизационные подходы делятся на две большие категории:

- усилия, предпринимаемые вручную и осознанно в процессе создания и экспорта изображения;
- сжатие после экспорта с помощью инструментов, которые просматривают код и еще сильнее сокращают его, выбрасывая, как правило, неиспользуемые данные.

ОВЛАДЕВАЙТЕ ПРИЕМАМИ ОПТИМИЗАЦИИ ИЗОБРАЖЕНИЙ

Все веб-дизайнеры должны иметь в своем распоряжении несколько приемов оптимизации изображений.

Этот раздел начинается с общих рекомендаций по ограничению размеров файлов. А затем, поскольку каждый формат изображения немного отличается от других, мы рассмотрим стратегии оптимизации для файлов JPEG, PNG-24, PNG-8 и GIF.

ЕЩЕ О ФОРМАТЕ SVG

Конечно, важно также оптимизировать и файлы в формате SVG, но об этом речь пойдет в главе 25.

Наконец, мы познакомимся с некоторыми инструментами оптимизации, которые работают с разными форматами и обеспечивают хороший заключительный этап в любом процессе создания изображений.

Общие рекомендации по оптимизации

Независимо от изображения или типа файла, необходимо учитывать несколько основных стратегий ограничения размера файла. В самом широком смысле они состоят в следующем:

- **начните с качественного оригинала** — начните с самого качественного исходного изображения, которое вам доступно. На его основе вы сможете делать копии с различными размерами и настройками сжатия, сохраняя этот оригинал в неприкосновенности;
- **ограничение размеров** — очевидно, самый простой способ для уменьшения размера файла, это ограничить размеры самого изображения. Нет никаких магических вычислений — просто не делайте изображения больше, чем нужно. Исключая лишнее пространство в изображении, представленном на рис. 24.14, я уменьшила размер файла на 3 Кбайт (23%);

600×200 пикселов (13 Кбайт)



500×136 пикселов (10 Кбайт)



Рис. 24.14. Можно уменьшить размер файла изображения, обрезав лишнее пространство

- **повторное использование** — если нужно неоднократно использовать на сайте одно и то же изображение, лучше создать только один файл изображения

- и ссылаться на него столько раз, сколько необходимо. Это позволит браузеру использовать кэшированное изображение, избегая дополнительных загрузок;
- *используйте соответствующие инструменты* — если вам предстоит выполнить значительную работу по созданию веб-изображений, стоит инвестировать средства в профессиональное программное обеспечение для редактирования изображений с веб-функциями. Выбирайте Photoshop, Sketch, PaintShop Pro или какую-либо другую упомянутую в этой книге программу в зависимости от ваших личных предпочтений и бюджетных ограничений;
 - *пропустите изображение через оптимизатор* — в вашем распоряжении должно быть несколько инструментов для оптимизации изображений. Я далее приведу перечень некоторых из них, в том числе и тех, которые можно использовать на бесплатной основе.

Оптимизация JPEG

Вот общие стратегии, применяемые для уменьшения размера файла JPEG:

- активнее применяйте сжатие;
- выбирайте опцию **Optimized** (Оптимизировать), если она доступна;
- смягчайте изображение, используя инструменты **Blur/Smoothing** (Размытие/Сглаживание);
- избегайте острых краев и резких деталей.

Активнее применяйте сжатие

Ваш инструмент номер один для оптимизации JPEG-файлов — это опция **Quality** (Качество), которую вы найдете практически в любом графическом инструменте. Настройка качества позволяет установить степень сжатия — пониженное качество предполагает использование более сильного сжатия и создание меньших по размеру файлов. Если графический редактор изображений имеет режим предварительного просмотра, вы сможете следить за качеством изображения по мере изменения степени сжатия. Разные изображения могут выдерживать различную степень сжатия, но, в целом, изображения достаточно хорошо сохраняют качество при умеренных (50–70) и даже низких (30–40) настройках качества. Качество, получаемое при тех или иных настройках, варьируется от программы к программе, поэтому для вашего конкретного изображения используйте ту настройку, которая обеспечит наилучший баланс качества и размера файла.

Выберите опцию **Optimized** в случае ее доступности

Оптимизированные JPEG-изображения имеют немного меньший размер файла и лучшую точность цветопередачи, чем стандартные JPEG-изображения (хотя лично я и не замечал особой разницы). Поэтому вам следует выбрать опцию **Optimized** (Оптимизировать), если ваша графическая программа ее предлагает.

Применяйте размытие изображений

Поскольку смягченные изображения сжимаются до меньшего размера, чем резкие, вы можете попробовать применить к изображению небольшое размытие по Гауссу, чтобы предоставить JPEG-сжатию большее поле для деятельности. При этом даже незаметное размытие всего изображения поможет уменьшить размер файла. В диалоговом окне **Export as JPEG** (Экспортировать как JPEG) программы GIMP имеется опция **Smoothing** (Сглаживание), которая выполняет эту задачу. Устаревшая команда Photoshop **Save for Web** (Сохранить для Интернета) также подключает использование ко всему изображению некоторой степени размытия.

Вы также можете применять к менее важным областям изображения более агрессивное размытие, не трогая при этом интересующие вас области. Например, на рис. ЦВ-24.15 показано, как я применила размытие ко всем областям изображения, кроме лица, которое осталось с исходным качеством отображения, что позволило уменьшить размер файла на 6 Кбайт или на 23%. Для такого изображения, как я полагаю, экономия стоит потери деталей по краям, но, конечно, вам решать, подходит ли размытие в каждом случае, основываясь на содержании и назначении ваших изображений.

Избегайте резких краев и деталей

JPEG-компрессия сжимает области мягко смешанных цветов более эффективно, чем области с высокой контрастностью, резкими краями и четкими деталями. Чтобы продемонстрировать вам эти различия, на рис. ЦВ-24.16 показаны два изображения со смешанными цветами. Размер файла изображения с большей контрастностью и детализацией (**detail.jpg**) более чем в четыре раза, превышает размер файла с мягкими переходами цветов (**gradient.jpg**) при той же настройке качества. Вам следует иметь этот момент в виду при создании своих изображений, и если на фотографии оказывается много деталей с резкими краями, подумайте, можно ли их как-то смягчить или подправить. Также посмотрите, не сможет ли формат PNG-8 предложить аналогичное качество изображения при меньшем размере файла.

«Оптимизация» PNG-24

Поскольку PNG-24 — это формат без потерь, и с изображениями в этом формате мало что можно сделать в плане оптимизации. Вам тогда лучше следовать таким советам:

- избегайте этого формата при работе с фотографиями — выбирайте JPEG;
- пропустите файл в этом формате через утилиту оптимизации;
- преобразуйте файл в этом формате в формат PNG-8 с несколькими уровнями прозрачности.

PNG-сжатие без потерь делает PNG-24 прекрасным форматом для сохранения качества изображений, но файл одного и того же изображения всегда будет иметь меньшие размеры при сохранении его в формате JPEG «с потерями». Следовательно, основная стратегия работы с фотографиями — избегать формата PNG-24, используя вместо него JPEG.

Возможно, вы используете формат PNG-24, поскольку вам необходимо иметь несколько уровней прозрачности, и это уважительная причина. Но и в этом случае у вас есть два следующих варианта. Обработка изображений одним из упомянутых в этом разделе далее оптимизаторов изображений — хороший способ удаления ненужных метаданных при сохранении изображения. Другой вариант — преобразовать файл изображения в формат PNG-8, сохраняя при этом альфа-прозрачность.

Преобразование в PNG-8

До недавнего времени инструментов для создания PNG-8 с альфа-прозрачностью не существовало. Теперь же Photoshop CC дает возможность создать файл PNG-8 с альфа-прозрачностью и меньшим размером файла с помощью диалогового окна **Export As** (Экспортировать как).

Вы также можете воспользоваться отдельной утилитой для преобразования PNG-24 в PNG-8 с альфа-прозрачностью. Среди возможных вариантов имеются следующие:

- **ImageAlpha** (pngmini.com) — эта программа для Mac, созданная Корнелом Лесинским (Kornel Lesiński), преобразует PNG-24 в PNG-8 (рис. 24.17). Размер файла с изображением оранжевого круга мне удалось уменьшить с 8,4 Кбайт до 2,6 Кбайт, сэкономив 69%. Поскольку круг окрашен в однородный цвет, я уменьшила его цветовую палитру до 64 цветов без какого-либо значительного изменения внешнего вида;
- **TinyPNG** (tinypng.com) — эта утилита позволяет вам для выполнения конвертации перетаскивать файл PNG прямо на ее веб-страницу.

АЛЬТЕРНАТИВЫ ФОРМАТУ PNG-24

Изображения в формате PNG-24 отличаются большими размерами файлов, поэтому разработчики ищут способы полностью избегать этого формата. Далее приводятся несколько вариантов, обеспечивающих достижение нескольких уровней прозрачности без использования PNG-24:

- преобразовать PNG-24 в PNG-8, как предлагается в этом разделе;
- поместить JPEG-версию изображения внутрь формата SVG, после чего создать прозрачные области с помощью SVG-функций обрезки или маскировки (см. главу 25);
- создать прозрачные области с помощью CSS Masks (www.w3.org/TR/css-masking-1/) — эта технология здесь не описана, но вполне заслуживает вашего внимания;
- использовать новые форматы изображения, такие, как WebP и JPEG 2000, которые поддерживают альфа-прозрачность. Они станут хорошей альтернативой PNG-24, как только улучшится поддержка этих форматов в графических редакторах и браузерах.

И ВСЕ-ТАКИ ОНА БЫЛА...

Adobe Fireworks имел мало кому известную возможность создания PNG-8 с альфа-прозрачностью, но эта возможность, начиная с 2013-го года, не поддерживается.

БИБЛИОТЕКА СЖАТИЯ PNGQUANT

Любопытно, что все эти инструменты используют созданную Корнелом Лесинским (Kornel Lesiński) библиотеку сжатия **pngquant** (pngquant.org), которая уменьшает количество цветов с 24-битных до 8-битных, назначая уровни прозрачности ячейкам индексированной цветовой карты.

Предлагаются также платная версия Pro и API-интерфейсы для разработчиков, которые позволяют применять инструмент «tinify» на большинстве платформ;

- **PunyPNG Pro** (punypng.com) — еще одна утилита сжатия с веб-интерфейсом, которая предлагает преобразование «с потерями» из PNG-24 в PNG-8, однако вы получаете эту функцию только с платной учетной записью Pro.

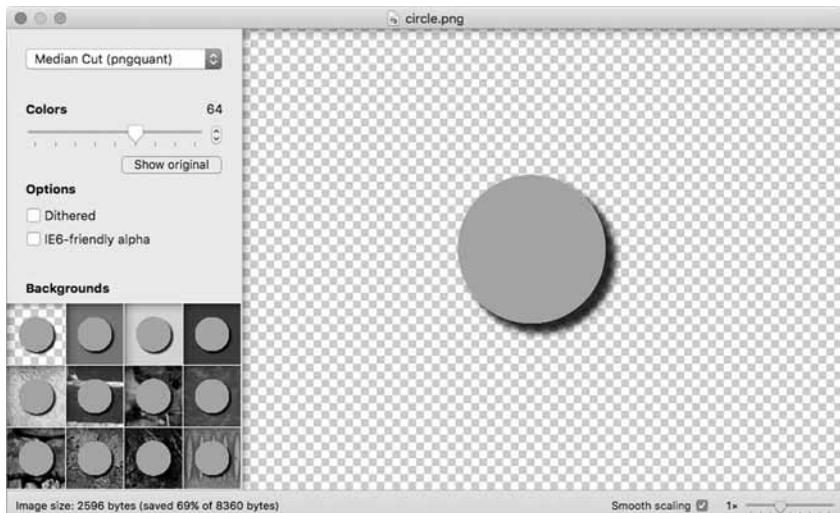


Рис. 24.17. Инструмент сжатия ImageAlpha (только для Mac) преобразует PNG-24 с альфа-прозрачностью в PNG-8, сохраняя при этом несколько уровней прозрачности

Оптимизация PNG-8 и GIF

Вот стратегии оптимизации, которым рекомендуется следовать в процессе создания и экспорта PNG-8 и GIF:

- уменьшайте количество цветов (битовую глубину);
- избегайте пятнистых участков (эффект дизеринга, dithering) или уменьшайте их;
- используйте в дизайне однородные цвета.

Уменьшайте количество цветов (битовую глубину)

Самый эффективный способ уменьшения размера индексированного цветного изображения и, следовательно, первый шаг на пути оптимизации — это уменьшение в нем количества цветов.

Хотя PNG-8 и GIF могут содержать до 256 цветов, не существует правила, которое утверждает, что это необходимо. Уменьшая количество цветов (битовую глубину), вы тем самым значительно уменьшаете размер файла изображения. Одна из причин этого заключается в том, что файлы с более низкой битовой глубиной содержат меньше данных. Другой побочный эффект от уменьшения количества цветов

заключается в том, что в результате комбинации сходных пиксельных цветов вы создаете большие области с однородными цветами. А однородные цветные области позволяют выполнить сжатие более эффективно.

Photoshop и GIMP в процессе преобразования изображения из RGB в индексированный цвет дают возможность уменьшать количество цветов. При работе в Photoshop выберите команды **Image | Mode | Indexed Color** (Изображение | Режим | Индексированный цвет) и введите в поле **Colors** (Цвета) количество цветов, которые буду использоваться в карте цветов. Если у вас имеется доступ к устаревшей функции Photoshop **Save for Web** (Сохранить для Интернета), найдите там настройку битовой глубины, с которой можно перед сохранением экспериментировать, наблюдая получаемое изображение в окне предварительного просмотра. При работе с программой GIMP перейдите в режим **Image | Mode | Indexed** (Изображение | Режим | Индексированный) и введите значение параметра **Maximum number of colors** (Максимальное количество цветов), которое вы решили использовать.

Разумеется, если вы чересчур сильно уменьшите количество цветов, изображение исказится или станет менее разборчивым. Например, как можно видеть на рис. ЦВ-24.18, уменьшив количество цветов в PNG до восьми, я утратила изображение радуги, которая была главным элементом композиции. Подобный эффект «развала» изображения, естественно, для каждого изображения будет своим. (Конечно, это изображение сафари и неба должно было бы быть в формате JPEG, но оно демонстрирует драматический эффект оптимизации, так что спасибо вам за понимание.)

Вы будете сильно удивлены, узнав, насколько хорошо выглядят изображения при использовании только 32-пиксельных цветов (5 битов) — таких, например, как логотип «азиатской кухни» (файл **asian.png**), с которым мы работали при выполнении *упражнения 24.1*. Это моя обычная отправная точка при уменьшении количества цветов, и дальше я иду только в случае крайней необходимости. Некоторые типы изображений при уменьшенной цветовой палитре получаются лучше, чем другие, но главное правило здесь: чем меньше цветов, тем меньше размер файла.

Эффект экономии на реальном размере файла выражается, когда в изображении присутствуют большие области однородного цвета. Имейте в виду, что даже если ваше изображение имеет 8-пиксельные цвета, однако в нем много пересечений, градиентов и деталей, вы не увидите той экономии размера файла, которую можно было бы ожидать при таком значительном снижении цвета.

32 ЦВЕТА — ЭТО ВПОЛНЕ ПРИЕМЛЕМО

Вы будете удивлены тем, как много изображений выглядят великолепно с палитрой только из 32-пиксельных цветов.

Уменьшение дизеринга

Если количество цветов в RGB-изображении уменьшено до определенной палитры, цвета, которые *отсутствуют* в этой палитре, аппроксимируются с помощью дизеринга (dithering). *Дизеринг* — это пятнистая структура, которая возникает при смешивании цветов палитры для имитации недоступного цвета. При преобразовании

БИТОВАЯ ГЛУБИНА

Битовая глубина представляет собой способ указания максимального количества цветов, которое может содержать изображение. Следующая табличка показывает количество цветов, которое представляет каждая битовая глубина:

1 бит —	2 цвета
2 бита —	4 цвета
3 бита —	8 цветов
4 бита —	16 цветов
5 битов —	32 цвета
6 битов —	64 цвета
7 битов —	128 цветов
8 битов —	256 цветов

ЭТО ВЫЗЫВАЕТ СОЖАЛЕНИЕ...

Не все инструменты для редактирования изображений позволяют контролировать степень дизеринга.

Если это неприемлемо, можно снова подключить дизеринг или опробовать большее количество цветов, если битовая глубина установлена в значение менее 8 битов.

Дизайн с использованием однородных цветов

При создании изображений следует учитывать, что форматы PNG и GIF хорошо сжимают области с однородным цветом.

Предпочтение однородных цветов градиентам и узорам сильно влияет на размер файла (рис. ЦВ-24.20). Уменьшение количества цветов с 256 (рис. ЦВ-24.20, слева) до 8 обеспечивает уменьшение размера файла, но смешиваемые цвета аппроксимируются с появлением эффекта дизеринга (рис. ЦВ-24.20, в центре), сглаживание которого плохо влияет на сжатие GIF и PNG. А вот если изначально создать изображение с однородными цветами, размер файла уменьшится вдвое по сравнению со сглаженной его версией, даже когда количество цветов в обоих изображениях сокращено до 8 (рис. ЦВ-24.20, справа).

Здесь я считаю необходимым уточнить, что подобные изображения следует создавать в программах векторной графики и сохранять в формате SVG, который дает меньшие по размерам файлы, чем растровые версии, и более универсален по сравнению с ними. Но если вы берете за исходное растровое изображение, вы должны отредактировать его таким образом, чтобы исключить ненужные сочетания цветов и узоров.

в индексированный цвет Photoshop и GIMP (а также большинство других графических редакторов) позволяют указывать, будет ли изображение сглаживаться с помощью дизеринга и каким образом станет выполняться это сглаживание.

На фотографических изображениях появление дизеринга не вызывает проблем и даже может приносить пользу, однако пятнистая структура в областях однородного цвета сильно портит впечатление и потому нежелательна. То есть с точки зрения оптимизации дизеринга следует избегать, поскольку наличие пятнистой структуры нарушает сглаженность областей цвета, мешает сжатию и приводит к увеличению размеров файлов.

Один из способов уменьшить размер файла формата PNG или GIF — полностью отключить дизеринг. При обработке некоторых изображений это может привести к появлению эффекта зазубренности, как показано на рис. ЦВ-24.19, *справа*.

Инструменты оптимизации

Даже если вы создаете изображения, в полной мере используя возможности конечного сжатия и средств оптимизации вашей программы редактирования изображений, есть большая вероятность, что вы сможете еще больше уменьшить размеры файлов своих изображений с помощью специальных инструментов оптимизации. Эти инструменты, как правило, выполняют преобразования изображений «без потерь», то есть никак не изменяют собственно картинки. Уменьшение размера файла достигается за счет удаления частей кода, предназначенных для метаданных, цветовых профилей и другой избыточной информации.

Я рекомендую завершать процесс создания изображения, пропустив его на последнем шаге через инструмент оптимизации. Таких инструментов имеется множество, так что вы обязательно найдете инструмент, который впишется в ваш рабочий процесс. Давайте рассмотрим несколько доступных вариантов.

Онлайновые оптимизаторы изображений

Одно из самых простых решений — это использование какого-либо свободно доступного в Интернете оптимизатора. Просто перетащите свои изображения на его веб-страницу и загрузите полученные сжатые файлы. К этому варианту имеет смысл прибегнуть, если обрабатывается не слишком много изображений. Он, кроме всего прочего, к тому же обладает преимуществом кросс-платформенности. В дополнение к бесплатным веб-инструментам большинство производителей также предлагают пакеты Pro, которые позволяют загружать больше данных и предоставляют дополнительные параметры сжатия, а некоторые также предлагают и серверные решения:

- **Optimizilla (optimizilla.com)** — доступен на бесплатной основе, может оптимизировать изображения как JPEG, так и PNG и выполнять сжатие до 20 изображений за раз;
- **Kraken.io (kraken.io/web-interface)** — в дополнение к коммерческим серверным сервисам предлагает бесплатный веб-интерфейс. Вы можете выбрать сжатие с потерями, без потерь, ручные «экспертные» настройки, а также возможность изменения размера изображений;
- **TinyPNG (tinypng.com)** — этот инструмент ранее упоминался как способ преобразования прозрачного PNG-24 в PNG-8, но вы можете использовать его и для сжатия любого PNG или JPEG;
- **PunyPng (punypng.com)** — создает файлы наименьшего размера на основе исходных файлов в форматах JPEG, PNG и GIF. Предлагается также пакет Pro, который дает больше возможностей сжатия, а также осуществляет преобразование PNG-24 в PNG-8.

ПОСЛЕДНИЙ ШАГ...

Всегда завершайте процесс создания изображений, пропуская их на последнем шаге через инструмент оптимизации.

Автономные приложения оптимизации

Возможно, вы предпочитаете иметь средства оптимизации на своем компьютере? Если это так, обратите внимание на следующие популярные загружаемые программы:

- **ImageOptim** (imageoptim.com) — этот инструмент, предназначенный только для Mac, обладает простым интерфейсом перетаскивания и служит для оптимизации PNG, JPEG, GIF (включая анимированный GIF) и даже SVG. Он был создан Корнелом Лесинским (Kornel Lesiński), который также подарил миру средство ImageAlpha;
- **PNGGauntlet** (pnggauntlet.com) — а этот инструмент представляет собой исключительно Windows-приложение, предназначенное для оптимизации PNG. Он также может конвертировать файлы JPEG, GIF, TIFF и BMP в формат PNG;
- **JPEGmini** (www.jpegmini.com) — программа для Mac и Windows, предназначенная для сжатия JPEG-изображений. Бесплатная пробная версия годится для обработки 200 изображений, но если вам необходимо обрабатывать большее количество изображений, придется заплатить за версию Pro. Пользователям предлагаются также бесплатный веб-интерфейс и серверный вариант;
- **Trimage** (trimage.org) — инструмент оптимизации, аналогичный по возможностям ImageOptim, но выполняется он на платформе Linux.

Плагины Grunt и Gulp

Если ваш рабочий процесс основан на обработчике задач, таком, как Grunt или Gulp, вы можете с помощью плагина `imagemin` реализовать оптимизацию изображений в форматах PNG и JPEG в виде автоматизированной задачи. Плагин `imagemin` поддерживается на [github.com/gruntjs/gruntcontrib-imagemin](https://github.com/gruntjs/grunt-contrib-imagemin), и там можно получить необходимые инструкции и ссылки для загрузки.

* * *

Теперь в вашем распоряжении имеется ряд методов оптимизации изображений, позволяющих уменьшить размеры их файлов при условии сохранения заданного их качества. Эти методы могут применяться как в процессе создания изображений, так и для последующей обработки и сжатия созданных файлов. Давайте проверим возможности некоторых из них при выполнении *упражнения 24.3*.

УПРАЖНЕНИЕ 24.3. ОПТИМИЗАЦИЯ НЕСКОЛЬКИХ ИЗОБРАЖЕНИЙ

В процессе выполнения этого упражнения мы возьмем хорошие изображения, полученные нами при экспорте в процессе выполнения *упражнения 24.1*, и посмотрим, можно ли их уменьшить еще сильнее, применяя инструмент оптимизации, доступный в Интернете. Я включила исходные изображения `boats-60.jpg` и `asian-32.png` в папку `materials` материалов для этой главы. Загрузите их оттуда, если хотите воспользоваться ими при выполнении этого упражнения.

Я собираюсь обратиться к сайту **Kraken.io**, поскольку он предлагает несколько вариантов оптимизации, включая выбор между сжатием с потерями и сжатием без потерь. Щелкните на кнопке **Try Free Web Interface** на домашней странице этого сайта для получения доступа к возможностям инструмента (рис. 24.21).

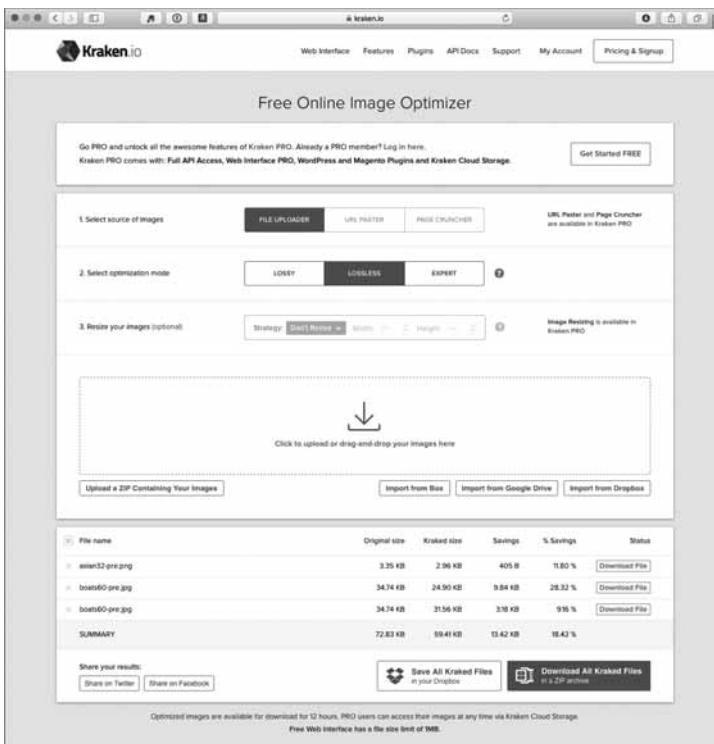


Рис. 24.21. Оптимизация изображений с помощью доступного в Интернете инструмента оптимизации изображений **Kraken.io**

- Начнем мы с файла **asian-32.png**, который, как вы помните, был уменьшен нами до 32-пиксельных цветов и сохранен в формате PNG-8. Воспользовавшись лучшим средством сжатия Photoshop (функцией **Export As**), мы получили размер файла для этого изображения, равный 3,35 Кбайт. Это совсем неплохо, а теперь посмотрим, сможем ли мы сделать этот файл еще меньше. Перетащите его в оптимизатор Kraken.io, выбрав режим **LOSSLESS** (Без потерь), — это означает, что данные собственно изображения сохранятся, а избыточная информация будет удалена.

Размер оптимизированного файла получился 2,96 Кбайт, то есть экономия с помощью оптимизатора Kraken.io составила 11,8% вовсе без изменения собственно изображения. Любопытства ради, я попыталась сжать это же изображение с помощью сервиса **TinyPNG.com** — файл в результате уменьшился на 15%. Это говорит о том, что разные инструменты дают разные результаты в зависимости от применяемых в них алгоритмов сжатия.

- Теперь посмотрим, что можно сделать с изображением **boats-60.jpg**. Сначала попробуйте использовать режим оптимизации «без потерь», не затрагивающий качество самого изображения. Размер файла изменился с 34,74 Кбайт до 31,56 Кбайт, что составляет чуть более 9%. Затем снова перетащите изображение в окно Kraken.io, используя на этот раз режим **LOSSY** (С потерями), позволяя инструменту удалить часть данных изображения, чтобы сжать его еще больше.

В результате размер файла получился всего 24,9 Кбайт, то есть достигается сжатие 28%! Я загрузила версии «с потерями» и «без потерь» и сравнила их в графическом редакторе, и, на мой взгляд, заметной разницы не обнаружила. Как видите, в результате применения оптимизации «с потерями» достигается существенное уменьшение размера файла без утраты качества изображения. Для сравнения — оптимизатору TinyPNG удалось уменьшить размер файла **boats-60.jpg** лишь на 3%.

Как можно видеть, обработка экспорттированных изображений с помощью оптимизатора стоит приложенных усилий. Тем не менее обработка каждого изображения по отдельности несколько затруднительна, поэтому, если надо оптимизировать большое количество изображений, рассмотрите возможность применения инструмента, позволяющего выполнить пакетную обработку, или автоматизируйте процесс с помощью обработчика задач, инструмента управления изображениями на стороне сервера или сервиса пакетной обработки.

На этом мы заканчиваем тур по технологиям создания изображений. Вы научились открывать изображения в приложениях для редактирования изображений и сохранять или экспорттировать их в различные форматы веб-изображений. Вы узнали о том, как форматы изображений хранят информацию о прозрачности и как выбрать наиболее подходящий формат для прозрачных изображений. Вы освоили приемы создания наборов изображений для адаптивных сайтов и, наконец, познакомились с вариантами финальной оптимизации изображений на последнем этапе их обработки.

Как обычно, глава завершается квестом, прохождение которого позволит вам применить полученные знания.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Освоили ли вы материал, относящийся к созданию изображений? Ответьте на эти вопросы, чтобы узнать, так ли это. Ответы на вопросы приведены в *приложении 1*.

1. Какими должны быть параметры формата файла, если необходимо иметь несколько уровней прозрачности в растревом изображении?
2. Какой инструмент лучше всего подходит для оптимизации изображений JPEG?
3. Какой инструмент лучше всего применить для оптимизации изображения с индексированными цветами, такого, как PNG-8 или GIF?
4. Каким образом дизеринг влияет на размер файла индексированного цвета PNG или GIF?
5. Как добавление размытия влияет на размер файла JPEG?
6. sRGB: Да или нет? Почему?
7. Зачем нужно создавать изображения масштабов @2x и @3x?
8. Зачем нужно обращаться к услугам таких компаний, как Cloudinary или Akamai?

ГЛАВА 25

SVG-ИЗОБРАЖЕНИЯ

В этой главе...

- ▶ *Фигуры SVG*
- ▶ *Обрезка и маскировка*
- ▶ *Эффекты фильтрации*
- ▶ *Стайлинг SVG-изображений*
- ▶ *Интерактивность и анимация*
- ▶ *Инструменты SVG*
- ▶ *Полезные советы*
- ▶ *Адаптивные SVG-изображения*

Графика SVG (Scalable Vector Graphics, скалярная векторная графика) несколько раз уже упоминалась мной ранее, но именно эта глава посвящена ей целиком.

При отображении в окне браузера SVG-изображения могут выглядеть как изображения любого иного формата, но как раз внутреннее устройство этих изображений делает их уникальными и универсальными. Уже из их названия следует, что формат SVG относится к векторным форматам — формы SVG-изображений определяются координатами и линиями, а не сетками пикселов. Это делает их масштабируемыми — можно до бесконечности изменять размеры этих изображений без потери качества.

На рис. ЦВ-25.1 приведено изображение тигра, сохраненное в форматах SVG и PNG. Формат SVG (*слева*) позволяет серьезно увеличить масштаб изображения без каких-либо потерь его качества. Линии и текст в нем отображаются четко вне зависимости от того, просматривается ли изображение с разрешением 100 или 10000 пикселов — вряд ли получится добиться такого же эффекта при обработке растрового изображения (*справа*)! В настоящее время наши веб-страницы и интерфейсы должны отображаться на любых устройствах с любыми масштабами — от смартфонов до мониторов высокой плотности и телевизоров с большим экраном, и возможность создания единого изображения, которое бы отлично смотрелось на любом устройстве, — это большое достижение.

ИСПОЛЬЗУЙТЕ SVG ДЛЯ СОЗДАНИЯ ЗНАЧКОВ И НЕСЛОЖНЫХ ИЛЛЮСТРАЦИЙ

Масштабируемость и уменьшенный размер файла делают SVG отличным форматом для создания значков и несложных иллюстраций.

Векторная природа SVG позволяет с успехом использовать этот формат для значков, логотипов, диаграмм и прочих линейных рисунков (рис. 25.2). Поскольку такие рисунки создаются на основе форм и контуров, размер их файлов часто значительно меньше, чем состоящие из сеток пикселов аналогичные растровые изображения.



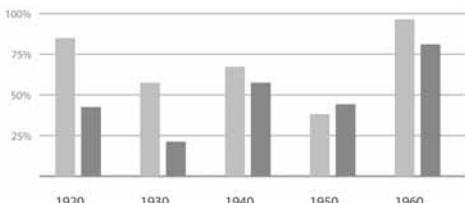
The Noun Project



"ben", Open Clip Art



Ozer Kavak, Open Clip Art



Ghostscript tiger

Рис. 25.2. Формат SVG подходит для создания штриховых иллюстраций

ДОСТУПНЫЕ ИСТОЧНИКИ SVG

Вы можете найти свободно доступные векторные изображения в формате SVG (или в форматах, легко конвертируемых в SVG) на следующих сайтах:

- The Open Clip Art library (openclipart.org);
- Freepik (freepik.com);
- IcoMoon (icomoon.io).

В SVG-файлы, кстати, можно включать и раственные изображения и, вообще, получать с помощью этого формата различные интересные эффекты, но экономия на размере таких файлов может оказаться уже не столь значительной. Формат SVG также позволяет добавлять в интерфейс анимацию и интерактивность. Все эти возможности мы здесь и рассмотрим.

ИСТОРИЯ SVG И БРАУЗЕРНАЯ ПОДДЕРЖКА

Формат SVG известен уже более 20 лет. Впервые он был представлен рабочей группой W3C SVG в 1998-м году, а его версия SVG 1.0 получила статус рекомендации в 2001-м году. Модуляризованная его версия SVG 1.1 была выпущена в 2003-м году, а затем изменена и переиздана в 2011-м году (www.w3.org/TR/SVG11/).

Браузерная поддержка формата SVG 1.1 более или менее приличная, но, к сожалению, не повсеместная. Браузеры начали широко поддерживать SVG (без плагинов) в период между 2004-м и 2006-м годами. Все современные браузеры поддерживают формат

▶ SVG в виде отдельных изображений и встроенного кода. Заметные «провалы в поддержке» — это Internet Explorer 8 (и более ранние его версии), а также Android 2.x, которые, к счастью, находятся на грани выхода из употребления, но все еще кое-где применяются на практике. Существуют и другие сложности с браузерной поддержкой, которые вы сможете почувствовать при переходе на уровень свойств и других тонкостей синтаксиса. Это распространенная проблема всех перспективных вебстандартов. Обзор поддержки функций SVG можно найти на странице «Comparison of Layout Engines (SVG)», доступной в Википедии по адресу: [en.wikipedia.org/wiki/Comparison_of_layout_engines_\(Scalable_Vector_Graphics\)](https://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Scalable_Vector_Graphics)).

Консорциум W3C также выпустил SVG Tiny 1.2 (www.w3.org/TR/SVGTiny12) как подмножество SVG 1.1, предназначенное для мобильных устройств, выпущенных до наступления эпохи смартфонов. Эта версия не поддерживается браузерами для смартфонов и ПК.

Формат SVG 2 (www.w3.org/TR/SVG2/), предусматривающий более тесную интеграцию с HTML5, CSS и WOFF (Web Open Font Format) находится сейчас в стадии разработки, но браузеры постепенно начинают внедрять поддержку отдельных модулей из спецификации SVG 2, так что и вы тоже можете начать ее тестирование и создание резервных вариантов на ее основе.

РИСОВАНИЕ С ПОМОЩЬЮ XML

Разберемся, каким образом в формате SVG поддерживаются «масштабируемость» и «векторность». Этот формат отличается от других форматов, благодаря тому что применяет язык XML (см. врезку «Краткое введение в XML») для описания двумерной графики, включая фигуры, контуры, текст и даже специальные эффекты фильтров. Растровые изображения, включенные в файлы SVG, сохраняются в них в значительной степени как непонятный код (если вы заглянете вовнутрь), однако собственно изображения SVG генерируются вполне понятными текстовыми файлами. То есть вы можете создавать SVG-графику, вводя код непосредственно в редакторе кода вместо применения графической программы.

КРАТКОЕ ВВЕДЕНИЕ В XML

XML (eXtensible Markup Language, расширяемый язык разметки) — это не самостоятельный язык, а, скорее, набор проверенных правил для создания других языков разметки. По сути — это метаязык.

Так, например, если вы хотите публиковать кулинарные рецепты, то можете использовать XML для создания собственного языка разметки рецептов (Recipe Markup Language, RML), который будет содержать элементы <ингредиент>, <инструкции> и <порции> для точного описания типов информации, включаемой в ваши рецепты. После правильно выполненной разметки такую информацию можно будет рассматривать как данные. XML реально оказался мощным инструментом для обмена данными между приложениями. Несмотря на

- ▶ то что XML разрабатывался с учетом потребностей Интернета, он, благодаря своим возможностям по обработке данных, оказал большее влияние и на внесетевые среды. Файлы XML используются «за кулисами» для все большего числа программных приложений, таких, как Microsoft Office и Apple iTunes. Вот лишь некоторые из языков XML, которые применяются в Интернете:
- **XHTML** — здесь HTML переписан в соответствии с более строгими правилами XML;
 - **RSS** (Really Simple Syndication или RDF Site Summary) — позволяет обмениваться контентом в виде данных и читать его с помощью RSS-ридеров;
 - **MathML** — используется для описания математических уравнений;
 - **SVG** — язык описания изображений, о котором пойдет речь в этой главе.

Требования к синтаксису XML

Поскольку в одном документе могут применяться несколько языков XML, важно, чтобы синтаксис их был строгим и чтобы все было понятно. Сокращения и упрощения, которые хороши в HTML (например, пропуск конечных тегов), в языках XML не сработают.

Формат SVG также придерживается более строгого синтаксиса XML, поэтому важно при записи кода SVG соблюдать следующие требования:

- имена элементов и атрибутов должны записываться в нижнем регистре;
- все элементы должны быть закрыты (завершены), а это означает, они должны иметь закрывающий тег. Для закрытия элементов без содержимого (например, пустых элементов) добавьте косую черту (слэш) перед закрывающей скобкой — например, так: `<rect/>`;
- значения атрибута должны заключаться в кавычки. Допускаются одинарные или двойные кавычки, если они применяются корректно. Кроме того, не должно быть никакого лишнего пространства (символьных пробелов или символов возврата строк) до и после значения атрибута внутри кавычек;
- все атрибуты должны располагать явными значениями атрибутов. XML не поддерживает минимизацию атрибутов — практику, при которой определенные атрибуты могут уменьшаться до значения собственно атрибута. Это лучше всего можно показать на примере из XHTML — более строгой версии HTML, переписанной на XML. В HTML вы можете записать `checked`, указывая тем самым на то, что кнопка формы должна проверяться при загрузке формы, но в XHTML нужно явно записать код: `checked= "checked"`;
- должно строго соблюдаться правильное вложение элементов;
- специальные символы всегда должны представляться символьными объектами (например, `&` для символа &). Обратите внимание, что большинство именованных объектов HTML не работают в XML. Вместо этого используйте числовую ссылку на кодовую точку Unicode;
- сценарии, поскольку они обрабатываются как простые текстовые символы, а не анализируются как разметка XML, должны содержаться в разделе `CDATA`. В разд. «Интерактивность, достигаемая с помощью JavaScript» этой главы приводится соответствующий пример.

Элементы SVG

SVG — это язык разметки, такой же, как и HTML, но он включает элементы для двумерной графики, среди которых:

- элементы для рисования линий и форм: `circle`, `rect`, `ellipse`, `path`, `line`, `polyline` и `polygon`;
- элемент `text`, предназначенный для добавления текстового содержимого;
- элементы структурирования, такие, как `g` — для группировки фигур, а также `use` и `symbol` — для повторного использования рисунков;
- элементы обрезки (`clipPath`) и маскировки (`mask`) областей изображения для формирования интересных форм;
- элементы `linearGradient` — для создания растровых эффектов и `filter` — для создания фильтров, подобных фильтрам Photoshop.

Конечно, это далеко не полный список элементов SVG, но он даст вам общее представление о том, что SVG собой представляет.

И лучше всего рассмотреть возможности SVG на простом примере. Так, на рис. ЦВ-25.3 показано изображение SVG (файл `simple.svg`), для создания которого использовались простые SVG-элементы. Я понимаю, что это далеко не шедевр, но все же познакомлю вас с некоторыми из них.

Далее приведено содержимое файла `simple.svg`, генерирующего изображение, показанное на рис. ЦВ-25.3. При внимательном просмотре этот пример будет вам интуитивно понятным, но все же небольшие пояснения тоже окажутся нeliшними:

```
<?xml version="1.0" encoding="utf-8"?> A
<svg version="1.1" B
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="150" height="200" viewBox="0 0 150 200"> C

<defs> D
  <radialGradient id="fade"> E
    <stop offset="0" stop-color="white"/>
    <stop offset="1" stop-color="orange"/>
  </radialGradient>
</defs>

<g id="greenbox"> F
  <rect x="25" y="25" width="100" height="100" fill="#c6de89"
        stroke-width="2" stroke="green"/> G
  <circle cx="75" cy="75" r="40" fill="url(#fade)"/>
  <path d="M 13 100 L 60 50 L 90 90 L 140 30"
```

ВСЕ ЭЛЕМЕНТЫ И АТРИБУТЫ SVG

Списки всех элементов и атрибутов SVG можно найти на сайте по адресу: www.w3.org/TR/SVG11/.

```

    stroke="black" strokewidth=
2" fill="none" /> ❷
</g>

<text x="25" y="150" fill="#000000" font-family="Helvetica"
fontsize="16">A Simple SVG</text> ❸

</svg>

```

Давайте познакомимся с различными частями файла **simple.svg** подробнее:

- Ⓐ Поскольку это XML-файл, то он начинается с некоторых моментов, связанных с XML. Первая строка — это объявление XML, идентифицирующее файл как XML. При публикации SVG в Интернете это объявление не требуется, если не использовать кодировку символов, отличную от заданной по умолчанию UTF-8, но вы, скорее всего, увидите его в коде, экспортируемом графическими программами.
- Ⓑ Весь документ содержится в корневом элементе `svg`. Графические программы обычно включают номер версии XML (1.1), хотя в этом нет необходимости.

ЕЩЕ ОБ АТРИБУТАХ `xmlns` И `xmlns:xlink`

Атрибуты `xmlns` и `xmlns:xlink` не нужны, если SVG-файл встроен в документ HTML5.

Два атрибута `xmlns` объявляют пространство имен XML, которое указывает браузеру на необходимость интерпретации этого документа с помощью словаря, определенного в SVG. Атрибут

`xmlns:xlink` позволяет размещать в документе SVG ссылки, а также отсылки к внешним файлам. Пространства имен помогают правильно использовать имена элементов, особенно когда в документе задействовано более одного языка XML.

- Ⓒ Атрибуты `width` и `height` корневого элемента `svg` устанавливают область рисования (*область просмотра*) шириной 150 пикселов и высотой 200 пикселов. Область просмотра выделена на рис. ЦВ-25.3 пунктирной линией, но здесь это сделано в демонстрационных целях и в браузере не отображается. Пиксели —

стандартная единица измерения в SVG, поэтому единицу «px» указывать не нужно. Атрибут `viewBox` управляет размерами собственно рисунка и позволяет графическому изображению аккуратно масштабироваться, если позже его ширина и высота будут изменяться.

ОБЛАСТЬ ПРОСМОТРА И АТРИБУТ `VIEWBOX`

Область просмотра и атрибут `viewBox` более подробно рассматриваются в разд. «Адаптивные изображения в формате SVG».

- Ⓓ Далее следует элемент `defs`, определяющий элементы и эффекты, на которые в документе будут позже ссылаться с помощью их значений `id`. Элементы в разделе `defs` создаются, но не отображаются сразу. Здесь мы задействуем элемент `def` для хранения шаблона радиального градиента, но также его можно использовать и для фигур (например, `circle`) или символов, отображаемых в документе с помощью элемента `use`. Определение фигуры, рисунка или эффекта выполняется один раз, а затем повторно используется — это отличный способ устранения избыточности в вашем SVG-коде.

- ❸ Элемент `radialGradient` состоит из двух элементов цвета `stop`: один — для белого, другой — для оранжевого и определяется идентификатором `id="fade"`.
- ❹ Здесь определяются элементы, описывающие собственно рисунок. Составляющие его прямоугольник (`rect`), круг (`circle`) и зигзагообразная линия (`path`) сгруппированы вместе с элементом `g` и названы именем "`greenbox`". Это в дальнейшем облегчает доступ к CSS или к сценарию.
- ❺ Квадрат создается с помощью элемента `rect` (от англ. `rectangle`, прямоугольник) с шириной и высотой, равными 100 пикселям. Обратите внимание, что этот и другие пустые элементы закрываются (заканчиваются) слэшем (/), расположенным перед закрывающей скобкой, как требуется в XML-языках.

Атрибуты `x` и `y` определяют позицию изображения на пиксельной сетке в области просмотра (см. рис. ЦВ-25.4 и врезку «Координаты SVG»). Вы можете увидеть также, как с помощью тех или иных атрибутов задаются различные размеры, цвета заливки, ширина и цвет обводки (`stroke`).

Центр элемента `circle` определяется с помощью атрибутов `cx` и `cy`, а его радиус — с помощью атрибута `r`. Круг `circle` с помощью записи `url()` заполняется определенным ранее радиальным градиентом, заданным идентификатором `id` «`fade`» (постепенно исчезающий).

- ❻ Зигзагообразная линия определяется элементом `path`. Атрибут `d` (data) определяет серию координат `x` и `y`, которые задают точки вдоль пути. Все пути начинаются с координаты `M` (от англ. `moveto`, перемещение), которая задает начальную позицию. Каждый элемент `L` (от англ. `lineto`, линия) рисует линию, определяемую его набором координат. Координаты могут разделяться пробелом (как показано здесь) или запятой.
- ❼ Здесь добавляется немного текста, определенного элементом `text`. Вы можете заметить, что ему присвоены стили с помощью атрибутов `font-size` и `font-family`, хорошо знакомых вам после изучения CSS. И, действительно, между атрибутами SVG и стилями CSS имеется много общего.

ШТРИХ

Термином `stroke` (штрих) в SVG называется обводка (линия или граница) вокруг фигуры.

КООРДИНАТЫ SVG

Координаты SVG начинаются с верхнего левого угла и увеличиваются вниз и вправо (рис. ЦВ-25.4). Созданный в этом примере с помощью элемента `rect` квадрат имеет координаты `x="25"` и `y="25"`, а это означает, что его верхний левый угол расположен на расстоянии 25 пикселов от левого края области просмотра/окна просмотра и на 25 пикселов ниже его верхнего края. Некоторые элементы, такие, как круги и эллипсы, могут позиционироваться с учетом их центральных точек (`cx` и `cy`).

Не только простые фигуры

Формат SVG, помимо рисования линий и фигур, предлагает еще несколько интересных возможностей.

Встроенные растровые изображения

Формат SVG не ограничивается векторными рисунками — вы также можете внедрять в него растровые изображения. Это обычно делается с целью добиться каких-либо специальных эффектов или чтобы добавить некое поведение или интерактивность, которое изображения PNG или JPEG не могут реализовать самостоятельно. Внедрение растровых изображений осуществляется с помощью элемента `image`:

```
<image xlink:href="kangaroo.jpg" x="45" y="0" width="100"
height="150" />
```

Обратите внимание, что поскольку SVG представляет собой XML-формат, он для указания на внешний файл изображения нуждается в атрибуте `xlink:href`.

Обрезка и маскировка

Формат SVG с помощью функций обрезки и маскировки позволяет выборочно показывать одни части изображения и скрывать другие.

При выполнении обрезки векторный путь используется для «вырезания» части изображения, при этом та часть изображения, которая выходит за пределы заданного контура обрезки, полностью скрывается. На рис. 25.5 для обрезки используется траектория в форме звезды, позволяющая вырезать часть изображения звездного неба. Контуры обрезки определяются с помощью элемента `clipPath`:

```
<defs>
  <clipPath id="star">
    <polygon points="390,12 440,154 590,157 470,250 513,393 390,307
266,393 310,248 189,157 340,154 390,12" style="fill: none"/>
  </clipPath>
</defs>

<image xlink:href="starrysky_600.jpg" width="600" height="400"
style="clip-path: url(#star)"/>
```



Звездообразный контур располагается над изображением.



Изображение обрезается по звездообразному контуру (пунктирная граница здесь показывает область просмотра SVG, но не является частью SVG-изображения).

Рис. 25.5. Звездообразный путь используется в качестве контура обрезки, который отбрасывает часть основного изображения

Маскировка функционирует аналогично обрезке, но это — пиксельный эффект. Изменяющиеся уровни затемнения маски — подобно альфа-каналу — определяют

различные уровни прозрачности в каждой точке изображения. В масках SVG чистые белые области соответствуют непрозрачности, равной 100%, а чистые черные области — непрозрачности, равной 0% (полностью прозрачные). Промежуточные уровни серого маски определяют уровни полупрозрачности изображения.

Вы можете поэкспериментировать со цветом заливки маски, чтобы выявить большую или меньшую степень маскировки объекта. Эффект получается более интересным, если маска включает градиенты, а не сплошные цвета заливки. В качестве маски можно применять даже изображение.

На рис. 25.6 показана та же фигура звезды, залитая градиентом и использованная в качестве маски для снимка звездного неба. Обратите внимание, что области изображения, которые выходят за пределы маскирующего объекта, полностью прозрачны, как и контур обрезки:

```
<defs>
  <linearParams id="blend">
    <stop offset="0%" stop-color="#ffffff" />
    <stop offset="100%" stop-color="#000000" />
  </linearParams>

  <mask id="star" x="0" y="0" width="400" height="381">
    <polygon points="390,12 440,154 590,157 470,250 513,393 390,307
266,393 310,248 189,157 340,154 390,12" style="fill: url(#blend)" />
  </mask>
</defs>

<image xlink:href="starrysky_600.jpg" width="600" height="400"
style="mask: url(#star);"/>
```



Звездообразный контур залит с градиентом.



Градиент работает как маска, в которой светлые области позволяют просвечивать большей части изображения. Чем темнее маска, тем светлее замаскированное изображение.

Рис. 25.6. Фигура звезды имеет градиентную заливку, которая влияет на прозрачность маскированного изображения

ОСОБЕННОСТЬ ОТОБРАЖЕНИЯ ВНЕШНИХ ИЗОБРАЖЕНИЙ

В файлах SVG, добавленных на страницу HTML с помощью элемента `img`, внешние изображения отображаться не будут. Дело в том, что все внешние файлы для таких SVG-файлов заблокированы по соображениям безопасности.

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ

Маскировка не поддерживается в версиях Android 4.3 и более ранних.

ЕСЛИ МАСКА ЦВЕТНАЯ...

Если маска представляет собой цветное изображение, она преобразуется в оттенки серого на основе уровня яркости с помощью формулы, интерпретирующющей желтый и зеленый цвета как более светлые, чем красный и синий.

ИМИТАЦИЯ ПРОЗРАЧНОСТИ JPEG

Можно имитировать прозрачность JPEG, внедряя файлы JPEG в файлы SVG и используя контур обрезки или маску, чтобы сделать прозрачными определенные области изображений. Это позволяет использовать полноцветные изображения JPEG небольшого размера для получения эффектов прозрачности, доступных только в PNG-24. К сожалению, здесь возможны проблемы с браузерной поддержкой, с которыми не приходится сталкиваться при работе с PNG-24.

Эффекты фильтрации

Вы будете удивлены, но векторный графический формат способен предоставить для работы с изображениями Photoshop-подобные фильтры. Формат SVG обеспечивает более дюжины эффектов фильтрации — таких, как простые размытия по Гауссу, смещение цвета, мозаичные узоры и внесение «добрых старых» теней — которые можно использовать как по отдельности, так и по слоям или комбинировать между собой.

Самое приятное в этом, что исходное изображение остается нетронутым — применение заданных фильтров происходит непосредственно при выводе изображения в браузере. На рис. ЦВ-25.7 показаны только несколько SVG-фильтров — это даст вам общее представление о возможностях SVG-фильтрации.

Чтобы разобраться в том, как это работает, рассмотрим пример наложения эффекта размытия на элемент эллипса (рис. ЦВ-25.8). Фильтр определяется с помощью элемента `filter`, который включает один или несколько *примитивов фильтра* (весьма

ПРИМЕНЕНИЕ АТРИБУТОВ СТИЛЯ

В этом и следующем примерах для добавления встроенных стилей к элементам применяется атрибут `style` (тот же, что мы используем в HTML). В следующем разделе речь пойдет о возможностях стилей SVG.

специфический эффект, который можно комбинировать с другими эффектами). Фильтру присваивается идентификатор `id`, по которому он потом вызывается как стиль для элемента, к которому применяется:

```
<defs>
  <filter id="blurry">
    <feGaussianBlur in="SourceGraphic" stdDeviation="4"/>
  </filter>
</defs>

<ellipse cx="200" cy="50" rx="150" ry="100" style="fill: orange; "/>

<ellipse cx="200" cy="300" rx="150" ry="100" style="fill: orange;
filter: url(#blurry);"/>
```

В следующем примере для формирования эффекта падающей тени определяется фильтр размытия, который затем объединяется со смещением, перемещающим ее вниз и вправо (рис. 25.9):

```

<defs>
<filter id="shadow">
    <feGaussianBlur in="SourceAlpha"
stdDeviation="4" result="blur"/>
    <feOffset in="blur" dx="7" dy="5"
result="offsetBlur" />
    <feMerge>
        <feMergeNode in="offsetBlur"/>
        <feMergeNode in="SourceGraphic"/>
    </feMerge>
</filter>
</defs>
<polygon points="390,12 440,154 590,157 470,250 513,393 390,307
266,393 310,248 189,157 340,154 390,12" style="fill: pink; filter:
url(#shadow)"/>

```

Разумеется, SVG-фильтры представляют собой гораздо более обширную тему, нежели здесь показано, но я надеюсь, что хорошее вступление к этой теме я все же вам предоставила.

Повторное использование

Значимая особенность SVG — это возможность определить фигуру или эффект один раз, а затем использовать их везде, где вам нужно, и столько раз, сколько вам нужно. Это уменьшает размер файла за счет удаления избыточного кода и является хорошим примером неповторяющегося (*сухого*) кодирования — от англ. DRY (Don't Repeat Yourself, не повторяйся).

Определение повторяющегося объекта (в следующем примере — значка) осуществляется с помощью элемента `symbol`. При этом сам элемент `symbol` не отображается — он только определяет схему будущего применения объекта:

```

<symbol id="iconA" viewBox="0 0 44 44">
    <!-- all the paths and shapes that make up the icon -->
</symbol>

```

Когда вы захотите показать на странице объект, помеченный элементом `symbol`, вызовите его с помощью элемента `use`, который запускает отображение такого объекта. Далее приведен пример минимального применения элемента `use`. Повторно используемый объект масштабируется до любых размеров, заданных при помощи параметра `svg.icon` в таблице стилей веб-страницы:

```

<svg class="icon">
    <use xlink:href="#iconA">
/>
</svg>

```



Рис. 25.9. Падающая тень, сформированная с помощью SVG-фильтров

SVG-РАСШИРЕНИЯ

Функции маскировки, фильтрации и преобразования CSS задействуются в SVG в качестве расширений. Органы стандартизации стремятся сделать так, чтобы они работали вместе как можно более плавно, а также стараются привнести некоторые из лучших аспектов SVG в CSS и стандартное поведение браузеров.

ХОРОШАЯ ПРАКТИКА КОДИРОВАНИЯ

Поскольку символы отображаться не будут, вам нет нужды помещать их в раздел `defs`, однако помещать их туда — хорошая практика, поскольку этот логический контейнер предназначен для элементов, определенных для дальнейшего использования.

Вы можете подключить к объекту и другие атрибуты с инструкциями — такими, как координаты x и y для позиционирования, размеры по ширине и высоте, а также со стилями, определяющими стили, унаследованные копией объекта.

Элемент `use` работает не только с объектом, помеченным элементом `symbol`. Аналогичным образом вы можете применять его для повторного использования любой основной SVG-фигуры, изображения или группы. Преимущество определения исходного SVG с помощью элемента `symbol` в том, что это дает возможность подключать атрибут `viewBox`, активизирующий пропорциональное масштабирование.

Элементы `symbol` и `use` являются инструментами *спрайтов* SVG. Спрайты — это методы, при которых несколько SVG-рисунков (например, набор значков — это, пожалуй наиболее популярный пример) определяются либо в одном SVG- или HTML-документе, либо в виде внешнего файла `*.svg`. В HTML-документе элемент `use` (внутри элемента `svg`) вытаскивает на страницу значок, определенный элементом `symbol`. То есть это — универсальный инструмент, предназначенный для управления значками SVG. В Интернете вы найдете множество учебных пособий по SVG-спрайтам, а Крис Коуэр (Chris Coyier) включил в свою книгу «Practical SVG» хорошее практическое руководство на эту тему.

К этому моменту вы разобрались, как SVG рисует основные фигуры, внедряет изображения, обрезает и маскирует выбранные области, а также добавляет замечательные спецэффекты. Возможности SVG-рисования являются основными для формата SVG, однако, если мы сосредоточимся только на том, что рисуется в окне браузера, мы упустим некоторые из лучших возможностей SVG. Давайте рассмотрим их сейчас.

ВОЗМОЖНОСТИ SVG, СВЯЗАННЫЕ С XML

Теперь вы знаете, что за каждым изображением SVG, отображаемым на экране, кроется структурированный текстовый документ. В этом отношении они почти такие же, как основанные на HTML. Кроме того, SVG, как структурированный язык документов, обладает DOM, которая включает объекты, свойства и методы, связанные с манипулированием графическими элементами. Это открывает некоторые действительно интересные возможности, которые делают SVG более гибким и полезным, чем это имеет место просто при выводе статических изображений.

Стайлинг

Вы можете назначить элемент в SVG (или же элемент `svg`, если он является встроенным) целевым объектом для изменения его представления с помощью CSS — например, задавая на странице одинаковый цвет или стиль границы как HTML-элементам, так и фигурам SVG.

Стили добавляются в SVG четырьмя способами:

- *атрибуты представления* — в представленном ранее примере «Простое SVG-изображение» (см. рис. ЦВ-25.3) для управления отображением фигуры используются определенные на языке SVG атрибуты представления, такие, как

`fill` `stroke-width`. Атрибуты представления переопределяются стилями, которые применяются с помощью CSS правил:

```
<rect x="25" y="25" width="100" height="100" fill="#c6de89"
      stroke-width="2" stroke="green" />
```

- *встроенные стили* — элементы SVG могут использовать встроенный атрибут `style`, который функционирует в них так же, как в HTML-элементах. Многие разработчики предпочитают именно такой подход. Тот же самый элемент `rect` можно записать следующим образом:

```
<rect x="25" y="25" width="100" height="100"
      style="fill:#c6de89;
              stroke-width:2; stroke:green;" />
```

- *внутренняя таблица стилей* — как и при работе в HTML, элемент `style` можно включать в верхнюю часть элемента `svg` (или в раздел `defs`, если он имеется), куда входят все стили, применяемые в SVG-документе:

```
<svg> <!-- Определение XML пропущено -->
<style>
    /* styles here */
</style>
<!-- Рисунок -->
</svg>
```

- *внешняя таблица стилей* — если SVG-документ встроен или размещен на странице с помощью элементов `object` или `iframe`, то, обращаясь к правилу `@import` в элементе `style`, можно импортировать внешнюю таблицу стилей. Учтите, внешние файлы не будут функционировать при работе с автономными SVG-документами, встроенными с помощью элемента `img`. Запись, импортирующая внешнюю страницу стилей, включает собственно таблицу стилей, а также ссылки на внешние ресурсы с использованием в правилах стилей нотации `url()`:

```
<svg>
<style type="text/css">
    @import "svg-style.css";
    /* дополнительные стили */
</style>
<!-- рисунок -->
</svg>
```

Для встроенных SVG-документов также можно стилизовать элементы (используя элемент `style`), обращаясь к таблице стилей, связанной с документом HTML с помощью элемента `link`:

```
<head>
    <!-- дополнительные элементы head -->
    <link href="svg-style.css" rel="stylesheet" type="text/css">
</head>
<body>
    <svg>
        <!-- рисунок -->
    </svg>
</body>
```

ВСПОМИНАЕМ: ДОБАВЛЕНИЕ SVG-ДОКУМЕНТА НА СТРАНИЦУ

В главе 7 мы уже вели речь о добавлении SVG-документа на страницу, но теперь, когда мы глубже погружаемся в SVG, было бы неплохо вспомнить, о чем мы говорили тогда. Впрочем, вы также можете вернуться к главе 7 и еще раз внимательно просмотреть ту ее часть, которая посвящена разметке. Итак, документы SVG могут добавляться в HTML-документ следующими способами:

В виде изображения

Отдельный файл *.svg вы можете добавить на страницу с помощью элемента `img`, как и любую иную графику:

```

```

Отдельный документ SVG также может быть использован в любом CSS-свойстве, принимающем изображения, — например: `background-image`.

Если отдельные SVG-файлы добавляются на страницы в виде простой графики, они и ведут себя как простая графика. Вы не сможете задавать им стили или создавать для их элементов сценарии — они не станут интерактивными (то есть не будут реагировать на пользовательские события, такие, как щелчки или наведение) и не будут выполнять загрузку внешних файлов, таких, как встроенные изображения, таблицы стилей или сценарии. Но, если вам необходимо лишь статическое изображение, этот вариант добавления является вполне приемлемым.

В виде встроенного объекта

Для встраивания SVG на страницу вы можете применять элемент `object`. Преимущество этого метода в том, что он позволяет запускать сценарии и загружать внешние файлы. А для браузеров, его не поддерживающих (хотя их и немного), можно также предоставить и резервное изображение:

```
<object data="star.svg" type="image/svg+xml">
  
</object>
```

В виде встроенного SVG-кода

Весь SVG-файл можно полностью вставить непосредственно в исходный HTML-код в качестве элемента `svg`. Такой подход предоставляет полный доступ к DOM SVG для стилей и сценариев, что является большим преимуществом. С другой стороны, код SVG бывает весьма объемным.

Интерактивность, достигаемая с помощью JavaScript

Документы SVG — это не просто красивые картинки, а изображения, которые можно *программировать!* Интерактивность к элементам SVG вы можете добавить с помощью JavaScript, поскольку все их элементы и атрибуты доступны в DOM. Следует отметить, что файлы SVG также могут содержать и простые ссылки, которые служат основным видом интерактивности.

Так, зная, что изображения SVG способны отслеживать указатели мыши, вы можете формировать связанные с наведением курсора забавные эффекты, вносящие инди-

видуальное поведение в элементы пользовательского интерфейса. Изменения в виде SVG также можно запускать щелчком или касанием. С помощью JavaScript вы получите множество возможностей — от небольшого сдвига значка до создания целых Flash-подобных игровых интерфейсов и мультимедийных презентаций (рис. 25.10).



Рис. 25.10. Пример игрового SVG-интерфейса, созданного с помощью библиотеки JavaScript [Snap.svg](#). При наведении указателя на каждую его точку появляется забавный маленький червячок. Много интерактивных демо-файлов SVG можно найти на сайте по адресу: snapsvg.io/demos

Если код SVG встроен в HTML-документ, сценарии в этом документе могут обращаться к элементам в SVG. Для самостоятельных SVG-документов вы можете использовать SVG-элемент `script`. Поскольку это все же XML-документ, код должен заключаться в XML-блок символьных данных (XML Character Data Block): (`<![CDATA[]>`) — тогда символы `<`, `>` и `&` будут анализироваться правильно:

```
<script><![CDATA[
    //сценарий
  ]]></script>
```

Анимация

SVG представляет собой популярный вариант средства для добавления на веб-страницу анимированных элементов.

SVG ИЛИ CANVAS?

В главе 10 мы рассматривали HTML5-элемент `canvas` и API, который формирует на веб-странице пространство для двумерного динамического рисования. Разница между SVG и Canvas заключается в том, что SVG-изображение формируется с помощью языка структурной разметки, а Canvas — с помощью команд JavaScript. В обоих случаях изображения могут включать другие изображения, видео, анимацию и динамически обновляться в режиме реального времени.

Использование элемента `canvas` удобнее, когда необходимо быстро перерисовывать изображение «на лету» (в конце концов, это всего лишь пиксели), что более подходит при разработке игр, редактировании изображений и сохранении изображений в растровых форматах. Преимущества SVG состоят в простоте написания сценариев, возможности анимации и доступности, однако сложные SVG-документы требуют большей вычислительной мощности, чем в случае применения элементов `canvas`.

На рис. 25.11 представлена моя попытка запечатлеть очаровательную анимацию в неподвижном изображении. Для получения более яркого впечатления я рекомендую перейти на сайт codepen.io и выполнить поиск: SVG animation.



Рис. 25.11. Пример анимированного SVG-изображения, предложенный Крисом Гэнноном (Chris Gannon). Посмотреть его в действии можно на сайте по адресу: codepen.io/chrisgannon/pen/empVgMg

Существует несколько способов представления SVG-анимации: элементы анимации в SVG, CSS-анимация и JavaScript:

- SVG/SMIL — спецификация SVG включает анимационные эффекты анимации на основе SMIL (Synchronized Multimedia Integration Language), языка XML, предназначенного для создания синхронизированных аудио, видео и анимированных элементов. Каждый анимационный эффект определяется соответствующим элементом с атрибутами для его тонкой настройки. И хотя встроенные элементы анимации SVG/SMIL предоставляют отличные инструменты для выполнения всех задач анимации, слабая их поддержка браузерами означает, что это не лучший подход, если анимация имеет решающее значение для вашего сайта;

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ

Ни один браузер Microsoft (ни Internet Explorer, ни Edge) не поддерживает анимацию SVG/SMIL. А браузер Chrome настолько ее не приемлет, что отправил сообщение поставщикам других браузеров с призывом о прекращении активной поддержки SVG/SMIL. С учетом CSS и JavaScript, предлагающих лучшие варианты анимации, эта часть спецификации SVG может увядать на корню.

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ

CSS-анимация не поддерживается в Internet Explorer 9 и более ранних его версиях (вообще, не только для SVG), а поддержка SVG вообще отсутствует в IE8 и более ранних его версиях. Браузеры IE 10 и 11 поддерживают анимацию свойств CSS, но не свойства SVG (`fill`, `stroke` и т. д.). В браузерах MS Edge и Firefox, выпущенных до 2017-го года, CSS-анимация работать не будет, если SVG добавлен с помощью элемента `img`. Устаревшие браузеры Chrome и Safari нуждаются в применении префикса `-webkit-`.

- анимация CSS — элементы SVG также можно анимировать с помощью CSS-переходов и ключевых кадров. Следует отметить, что CSS может анимировать только свойства CSS, а не значения атрибутов, что может стать ограничением для SVG, где для большей части геометрии и разметки используются атрибуты. Этот способ также весьма ограниченно поддерживается браузерами, хотя в настоящее время положение исправляется. В общем, CSS-анимация хороша лишь при формировании простых некритичных анимационных эффектов;
- JavaScript — с помощью JavaScript можно создавать сложные интерактивные анимации, которые конкурируют с функциональностью, предлагаемой Flash. Поддержка браузерами

этого способа значительно лучше, хотя всегда имеется вероятность того, что у некоторых пользователей JavaScript отключен и для них анимация будет недоступной. Если вы не желаете «изобретать велосипед», можете воспользоваться библиотеками JavaScript SVG Animation (см. врезку «Анимация SVG с помощью библиотек JS»).

Если вы пожелаете больше узнать о SVG-анимации, я рекомендую книги: Сары Драснер (Sarah Drasner) «SVG Animations» (издательство O'Reilly) и Кирупы Чинатамби (Kirupa Chinnathambi) «Creating Web Animations» (издательство O'Reilly).

Визуализация данных

Изображения SVG стали инструментом доступа к миру визуализации данных, поскольку они могут генерироваться динамически с использованием реальных данных. Например, вы можете сделать так, чтобы уровень температуры на изображении SVG-термометра повышался или понижался с учетом собранной пользователем реальной информации о погодных условиях, или же можно изменять индикаторы выполнения или круговые диаграммы в режиме реального времени в процессе обновления данных. На рис. 25.12 показаны примеры изображений SVG, используемых для визуализации данных из галереи D3.js — библиотеки JavaScript, созданной специально для «управляемых данными документов». Дополнительные сведения по этой теме можно получить по адресу: d3js.org.

Методы генерирования SVG с данными зависят от типа данных и применяемого языка программирования. Один из вариантов — перевести содержащий данные XML-документ в SVG с помощью XSLT (eXtensible Stylesheet Transformations) — языка XML, представляющего структурированные инструкции для перевода одного

АНИМАЦИЯ SVG С ПОМОЩЬЮ БИБЛИОТЕК JS

Далее приведены некоторые из доступных библиотек JavaScript, которые помогают более эффективно включать анимационные эффекты в ваши SVG-документы:

- **Snap.svg (snapsvg.io)** — Это многофункциональная библиотека SVG, созданная Дмитрием Барановским. Имеет открытый исходный код и находится в свободном доступе;
- **SVG.js (svgjs.com)** — это чрезвычайно скромная библиотека (всего 11 Кбайт!) для базовой SVG-анимации. Она имеет модульный характер, поэтому вы можете применять из нее только то, что вам необходимо;
- **Velocity (velocityjs.org)** — как следует из ее названия (velocity по англ. «скорость»), эта библиотека быстро отображает анимацию и имеет похожий на jQuery синтаксис, упрощающий ее применение;
- **Bonsai (bonsaijs.org)** — это надежная библиотека анимации SVG, включающая поддержку шрифтов, аудио, видео и изображений;

Анимация контуров — вам встречалось на веб-страницах изображение, которое создается будто бы на ваших глазах? Вероятно, это было SVG, анимированное с помощью аниматора контуров. Существует несколько JS-библиотек, которые могут анимировать контуры в вашем SVG: Walkway (connoratherton.com/walkway), LazyLinePainter (lazylinelpainter.info) и Vivus (maxwellito.github.io/vivus).

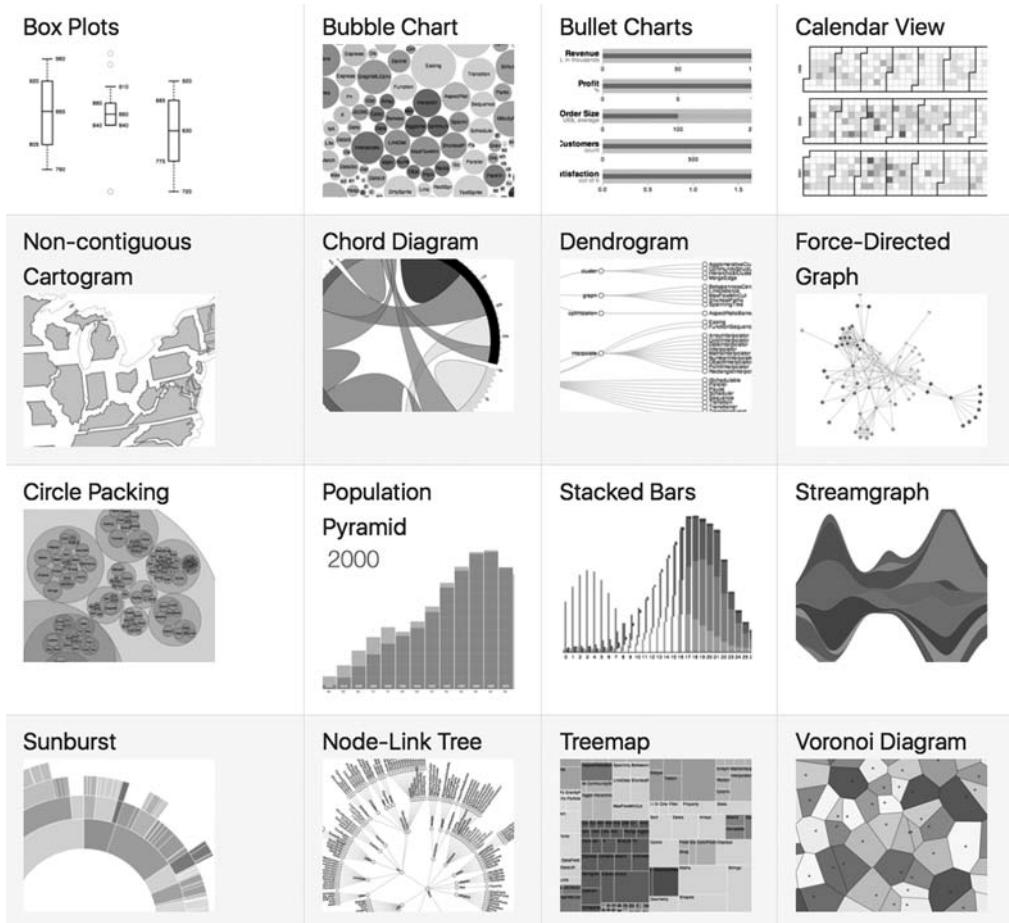


Рис. 25.12. Примеры сгенерированных SVG-данных на сайте [D3js.org](http://d3js.org). Обратитесь за дополнительными сведениями на сайт по адресу: github.com/d3/d3/wiki/Gallery

языка XML в другой. Понятно, что это некая продвинутая XML-магия, в которую мы не станем углубляться, но я полагаю, что вам следует об этом знать. Другие варианты включают JavaScript и языки шаблонов на стороне сервера, описание чего также выходит за рамки этой книги.

Если вы захотите узнать больше о сгенерированных данных SVG, лучше всего начать с книги Скотта Мюррея (Scott Murray) «Interactive Data Visualization for the Web» (издательство O'Reilly).

Доступность

В отличие от текста в изображениях растровых форматов, текст в SVG-файлах может быть доступен для поисковых систем и при правильной маркировке считываться программами чтения с экрана. И есть несколько условий, которые вы должны

учесть, чтобы ваши SVG-документы стали более доступными для программ чтения с экрана:

- воспользуйтесь SVG-элементом `title` для поддержки краткого имени для самого элемента `svg` или любого контейнера (такого, как `g`), либо для элемента, который он содержит. Элемент `title` должен быть первым дочерним элементом своего родителя;
- примените SVG-элемент `desc` для поддержки длинных текстовых описаний элементов;
- добавьте ARIA-роли в элемент `svg` и его компоненты — это гарантирует, что программы чтения с экрана интерпретируют их правильно и эффективно:
 - ◆ добавьте `role="img"` в элемент `svg`, но только если необходимо рассматривать его как единое неинтерактивное изображение. Дочерние элементы при этом не будут доступны отдельно;
 - ◆ добавьте `aria-labelledby="title desc"` в элемент `svg` для улучшения поддержки для `title` и `desc`;
 - ◆ если должны быть доступными только некоторые части SVG (текст, ссылки, интерактивные элементы и т. д.), не устанавливайте роли для `svg`, но внесите `role="presentation"` в фигуры (такие, как `circle`) и пути, чтобы программы чтения с экрана не объявляли о появлении каждой формы в этой графике.

ИНСТРУМЕНТЫ SVG

С технической точки зрения все что нужно для создания SVG-графики — это текстовый редактор (а также талант визуализатора плюс героическое терпение!), но вам будет гораздо приятнее, если всю тяжелую работу за вас сделает графическая программа. Дизайнеры часто создают сложные иллюстрации в графической программе, а затем переносят их в текстовый редактор для очистки кода и добавления сценариев и стилей вручную. Это вопрос предпочтений, основанный на ваших навыках и интересах.

Программы векторной графики

Наиболее подходящим инструментом для создания SVG-файлов является программа векторной графики — тот же Adobe Illustrator, хотя и редакторы растровых изображений, такие, как Photoshop и GIMP, используют формы, которые можно экспортить как SVG. Сейчас имеется много программ векторной графики — от дорогих до бесплатных, от полнофункциональных до простых.

Adobe Illustrator

Дедушка инструментов для создания векторной графики Adobe Illustrator доступен сейчас за ежемесячную плату как часть пакета Adobe Creative Suite. Illustrator — это векторный инструмент, но он исходно использует PostScript и нуждается

ЕСЛИ ВЫ РАБОТАЕТЕ В КОДЕ...

Скопировав форму, созданную в Illustrator, можно вставить ее в виде кода в текстовый редактор. Это удобно, если большую часть работы с SVG вы выполняете в коде.

жество советов по созданию в Illustrator оптимизированных SVG, и вам безусловно стоит их поискать.

Inkscape

Inkscape (inkscape.org) — это редактор векторных изображений с открытым исходным кодом, который создан специально для SVG (то есть SVG — его собственный формат).

INKSCAPE ДЛЯ MACOS

Версия Inkscape для macOS выполняется на XQuartz — варианте оконной системы X11 UNIX для Mac, поэтому пользователям Mac необходимо сначала ее загрузить и запустить ([доступна на www.xquartz.org](http://www.xquartz.org)).

много времени чтобы в нем разобраться. И вы определенно не найдете инструмента дешевле, поскольку Inkscape распространяется бесплатно!

в переводе своих векторных построений в формат SVG. И хотя он позволяет просто **Сохранить** (Save) рисунок в формате SVG, лучше все-таки выбирать команду **Экспортировать как** (Export As), поскольку полученная иллюстрация SVG будет оптимизирована для работы в Интернете. В Сети опубликовано множество советов по созданию в Illustrator оптимизированных SVG, и вам безусловно стоит их поискать.

Интерфейс Inkscape несколько изменился за последние несколько лет (рис. 25.13), но, если вы привыкли к инструментам Adobe, вам не потребуется

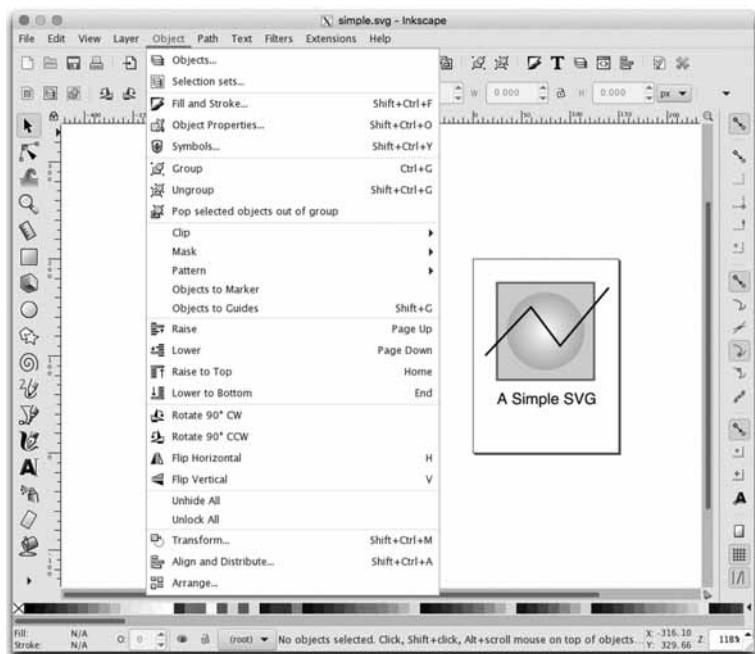


Рис. 25.13. Inkscape создан для работы с SVG, и на SVG настроены все его интерфейсные функции, поэтому его меню содержит опции **Fill and Stroke**, **Symbols**, **Clip** и **Mask**

ПРОПРИЕТАРНЫЙ КОД INKSCAPE

При сохранении SVG в Inkscape выберите опцию **Plain** (Простой) или **Optimized** (Оптимизированный). Имейте при этом в виду, что SVG-файлы, созданные в Inkscape, содержат много проприетарного кода, излишне увеличивающего размер файла, поэтому даже версия файла, сохраненная как оптимизированная, может содержать больше данных, чем это необходимо для нормального функционирования графики.

Инструменты, специфичные для SVG

Разработано много удобных, специально предназначенных для работы с SVG инструментов создания изображений, которые можно получить за небольшие деньги или же бесплатно. Поскольку они предназначены только для SVG, то содержат разумное количество опций и настроек, сопоставимое с возможностями SVG (никому не придется разбираться с сотнями ненужных им опций). Некоторые из них также позволяют просматривать и редактировать исходный SVG-код. Вот пара таких SVG-редакторов:

- **Boxy** (boxy-svg.com) — полнофункциональная графическая программа (рис. 25.14, слева), доступная для Mac, Windows и Linux за скромные 10 долларов США (на момент подготовки книги). Она имеет простой в использовании интерфейс, а также предлагает инспектор кода;
- **SVG-Edit** — это приложение (рис. 25.14, справа) работает непосредственно в браузере (svg-edit.github.io/svgedit/releases/svg-edit-2.8.1/svg-editor.html) или в качестве загружаемой программы (github.com/SVG-Edit/svgedit) и предоставляет все основные инструменты рисования, слои и возможность просмотра и редактирования SVG-кода. Кроме того, SVG-Edit допускает экспорт в форматы PNG, JPEG, BMP и WebP. И все это бесплатно, поэтому грех не воспользоваться такими возможностями.

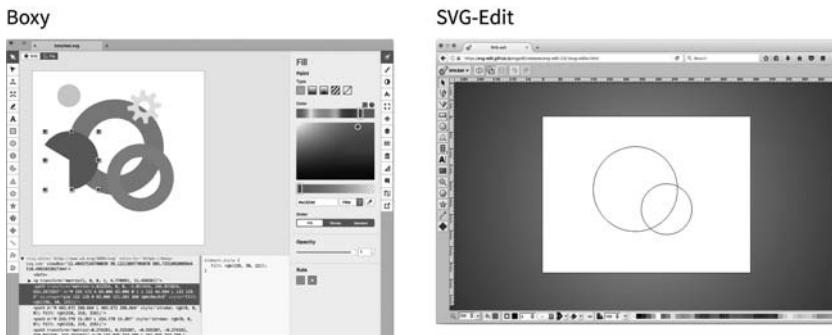


Рис. 25.14. Инструменты SVG Boxy (слева) и SVG-Edit (справа)

Инструменты дизайна интерфейса

Недавно появилось много новых инструментов, предназначенных для разработки интерфейсов веб-страниц и приложений, таких, как Sketch, AdobeXD и Affinity

Designer. Все они основаны на векторной графике и позволяют легко экспортировать компоненты в формате SVG. Как и в случае многих инструментов визуального проектирования, экспортирующих код, результаты не всегда так хорошо проработаны, как вам бы хотелось, особенно если вы далее намереваетесь работать с SVG в коде, применяя JavaScript, CSS и анимацию.

Редакторы кода

При создании кода SVG вручную или при работе с уже существующими файлами SVG следует убедиться, что ваш редактор кода оснащен расширением, предоставляющим возможность просмотра создаваемого изображения SVG по мере написания кода. Такие расширения доступны при работе со следующими редакторами:

- Atom Editor (свободно распространяется GitHub) — загрузите для него пакет SVG Preview с сайта по адресу: atom.io/packages/svg-preview;
- Brackets (свободно распространяется Adobe) — получите для него расширение SVG от Петера Флинна (Peter Flynn) на сайте по адресу: github.com/peterflynn/svg-preview.

СОВЕТЫ ПО СОЗДАНИЮ SVG

Создавая рисунок SVG в графическом инструменте, имейте в виду, что конечным результатом является текстовый файл. Как и для любого файла, который предназначен для размещения на веб-странице, необходимо добиться для него наименьшего размера. Если вы планируете выполнить стайлинг SVG с помощью CSS или манипулировать им с помощью JavaScript, необходимо, чтобы разметка была структурирована наилучшим образом.

Недостаток использования графических инструментов заключается в том, что вы не располагаете достаточными возможностями управления кодом, который они выводят. Большинство графических программ экспортируют неэффективный, избыточный и насыщенный проприетарными закладками SVG-код. Тем не менее вам доступны меры, которые можно предпринять в процессе создания SVG-изображений и после их экспорта, чтобы этот код был и прост, и функционален. Множество блогов содержат статьи, посвященные оптимизации SVG, но следующие советы, хотя и не всеобъемлющие, помогут вам встать на верный путь. Вам просто нужно будет ознакомиться с особенностями выбранного вами инструмента, чтобы предвидеть и исправлять его недостатки.

Наилучшие практические меры до экспорта

Решения, которые принимаются в процессе создания изображений, могут улучшить ваши SVG-результаты с точки зрения организации кода и уменьшения размера файла. Учитывая, что конечный продукт является текстовым файлом, это полезно и в плане его оптимизации. Далее приводятся несколько советов, которые помогут вам создать изображения SVG наилучшего качества при минимальных размерах файлов:

- определите область рисования или размер рисунка в пикселях — размеры области рисования должны соответствовать области просмотра (атрибуты `width` и `height` элемента `svg`);
- используйте слои для логической группировки элементов — если вы планируете добавление анимации или создание SVG-сценариев, созданный вами документ должен быть хорошо организован в процессе работы над ним — тогда вы сможете позднее получить доступ к нужным его фрагментам. При работе в Illustrator и в большинстве других инструментов слои преобразуются в элементы группы (`g`), а вложенные слои сохраняются как вложенные группы;
- придавайте элементам и слоям осмысленные имена — имена, которые вы придаете элементам и слоям, используются в качестве значений `class` и `id` в коде SVG, поэтому убедитесь, что они носят содержательный характер. Все имена должны быть записаны в нижнем регистре и без символьных пробелов — тогда они смогут быть использованы в качестве значений атрибутов;
- упрощайте контуры — чем больше точек и маркеров используются для определения контура, тем больше координат появляется в SVG-коде. А увеличение числа координат означает рост числа символов и увеличение размера файла. Воспользуйтесь любой функцией «упрощения контура», которую предлагает ваш инструмент. Рассмотрите возможность применения методов, уменьшающих количество элементов в файле, — таких, как объединение объектов, которое дает возможность отображать их как единый объект, и использование одного широкого штриха вместо двух штрихов и заливки. Убедитесь, что ваш инструмент использует для простых фигур элементы формы, такие, как `circle` и `rect`, вместо многоточечных контуров. Если ваш инструмент не позволяет реализовать это, вы можете вручную заменить код контура на простой элемент формы;
- обращайте внимание на десятичные знаки — учитывая, что большее количество символов приводит к большему размеру файла, можно сократить количество байтов в файле, ограничивая в ваших проектах количество символов после десятичных точек. Например, координата `x`, у «`100.3004598, 248.9998765`» требует больше данных, чем просто «`100.3, 249`». Многие инструменты позволяют ограничивать при экспорте количество десятичных знаков. Вы также при настройке документа можете выбрать «привязку к пикселям» — тогда точки всегда будут указывать на четные целые числа. Общее правило состоит в том, что чем меньше изображение, тем больше десятичных разрядов требуется для точного задания его координат. Большие изображения могут отображаться при более низкой точности без ущерба для их качества. Возможно, вам придется поэкспериментировать для установления правильного баланса между десятичными знаками и качеством изображения;
- избегайте растровых эффектов внутри SVG — эффективность SVG в том, что это векторная графика. При добавлении в SVG растровых изображений преимущество, связанное с малыми размерами файла, теряется. В требующих этого ситуациях вы, разумеется, можете добавлять в SVG растровое изображение, и это нормально. Однако имейте в виду, что некоторые эффекты в графических

программах, такие, как размытие, тени, свечение и т. п., весьма часто генерируют область растрового изображения, когда вы этого и не ожидаете, и это значительно увеличивает размер файла. Ряд эффектов фильтров, такие, как тени, можно более эффективно реализовать в коде после экспорта. Так что если обнаружится, что ваш SVG-файл имеет необычно большой размер, вероятнее всего причиной тому является растровое изображение;

- *уделите внимание шрифтам* — как и в любом случае назначения шрифта для документов, публикуемых в Интернете, нет никакой гарантии, что выбранный вами шрифт будет доступен на компьютере пользователя, то есть шрифт, примененный вами в SVG, может у него и не отображаться. Обязательно протестируйте назначаемые шрифты и предоставьте для них резервные варианты. Если текст в изображении SVG небольшой и не предусмотрено участие его в поиске или чтении вслух, подумайте, не лучше ли преобразовать такой текст в контуры;
- *используйте центрированные обводки (штрихи)* — и хотя это и не влияет на размер файла, лучших результатов можно добиться, если работать с обводками, установленными по центру контура, потому что именно таким образом SVG изначально обрабатывает обводки. Некоторые инструменты, такие, как Sketch, вносят корректировки для компенсации внешней или внутренней обводки, но лучший исходный момент — центрирование обводки.

Конечно, я могла бы привести и множество советов по работе с конкретными инструментами, но нельзя объять необъятное. Adobe на странице «About SVG» предлагает советы по оптимизации SVG в Illustrator (helpx.adobe.com/illustrator/using/svg.html). Если вы работаете со Sketch, то можете приобрести видеокурс Питера Новелла (Peter Nowell) «SVG Workflows in Sketch», который доступен на сайте по адресу: Sketchmaster.com.

Оптимизация экспортированных файлов

Даже при тщательном предварительном планировании экспортированные SVG-файлы все равно содержат значительную часть избыточного кода, ненужных метаданных, скрытых элементов и других объектов, которые можно безопасно удалить, что не повлияет на качество отображения SVG. Поэтому в силе остается совет пропускать SVG-файлы через оптимизатор, что позволит избавиться от ненужной избыточности кода и сократить размер файлов.

Оптимизатор SVGO

Лучшим SVG-оптимизатором на момент подготовки книги является SVGO (github.com/svg/svgo). Он применяет плагины, которые влияют на отдельные настройки, — такие, как удаление пустых атрибутов, удаление атрибута `xmlns` (просто отлично, если используется встроенный SVG), удаление комментариев и десятки других — так что вы можете выбирать, каким образом сжимать файл в зависимости от цели его применения.

Самый замечательный момент при работе с SVGO заключается в том, что его можно применять многими способами! Он основан на Node.js, поэтому можно использо-

вать его как модуль Node.js или включить в задачу Grunt или Gulp. Имеются плагины SVGO для Illustrator (SVGNow), Inkscape (SVGO-Inkscape) и Sketch (SVGO Compressor). Он также доступен в качестве папки действия (folder action) macOS, благодаря чему оптимизация выполняется просто при перетаскивании в него файлов. Полный список его параметров приведен на сайте SVGO.

Самый простой способ увидеть SVGO в действии — это воспользоваться созданным Джейком Арчибальдом (Jake Archibald) веб-инструментом SVGOMG (jakearchibald.github.io/svgomg/), который предоставляет для SVGO графический пользовательский интерфейс (GUI). SVGOMG позволяет переключать различные плагины оптимизации и просматривать результаты в режиме реального времени (рис. 25.15), что удобно в процессе разработки.

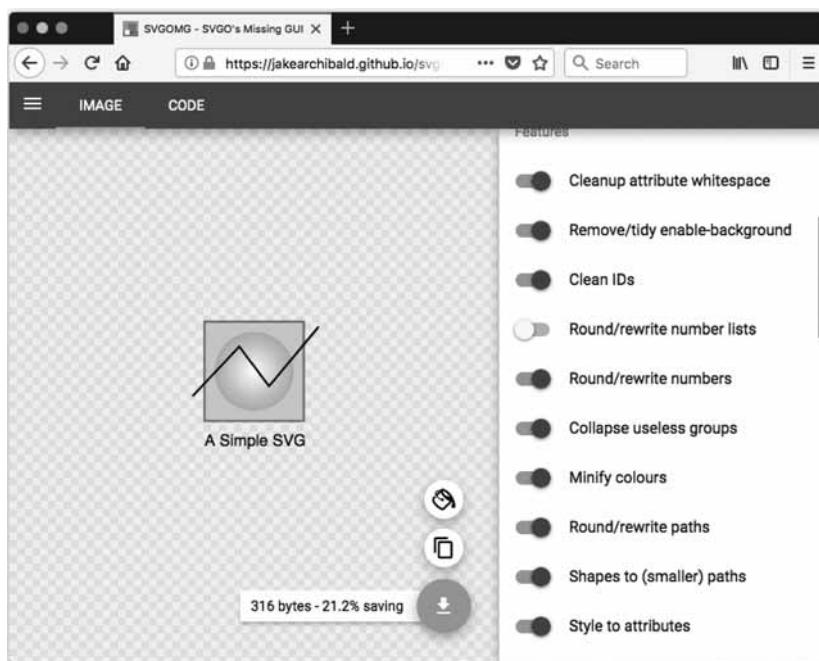


Рис. 25.15. Инструмент SVGOMG (jakearchibald.github.io/svgomg/) поддерживает графический пользовательский интерфейс (GUI) для оптимизационного инструмента SVGO. Вы можете переключать методы оптимизации, чтобы увидеть качество изображения и размер файла

Сжатие файла

Поскольку документы SVG представляют собой текстовые файлы, их можно сжать с помощью таких инструментов, как Gzip или Brotli:

- Gzip — это утилита на сервере, которая сжимает текстовые файлы с помощью ряда алгоритмов. Сжатие файлов SVG может уменьшить их до 16–25% от первоначального размера. GZIP-файлы SVG имеют расширение **svgz**. Чтобы воспользоваться возможностями Gzip для сжатия SVG, необходимо сконфигурировать

этую утилиту на сервере — описание этого процесса выходит за рамки книги, но его можно найти в большом числе учебных пособий в Интернете. Поверьте, сохранение файлов с помощью Gzip стоит приложенных усилий;

- Brotli — алгоритм сжатия с открытым исходным кодом, выпущенный Google в 2015-м году и опережающий Gzip в плане сжатия и производительности. Как и Gzip, Brotli должен конфигурироваться на сервере. Браузеры начали поддерживать контент, кодированный с помощью Brotli, в 2017-м году. На сайте MaxCDN имеется хорошее описание Brotli, доступное по адресу: www.maxcdn.com/one/visual-glossary/brotli/. Официальная страница Brotli GitHub находится по адресу: github.com/google/brotli.

АДАПТИВНЫЕ ИЗОБРАЖЕНИЯ В ФОРМАТЕ SVG

Я уже не раз отмечала, что масштабируемая природа изображений SVG делает их особо пригодными для использования в адаптивных макетах, где они могут масштабироваться в соответствии с изменяющейся шириной элементов без потери качества. Хотя это абсолютно верно, реальность такова, что есть несколько нюансов, только учитывая которые можно обеспечить предсказуемое масштабирование SVG во всех браузерах.

При решении задач адаптивного (отзывчивого) веб-дизайна часто возникает необходимость растяжения или сокращения графики в соответствии с шириной текстового контейнера. Имея дело с растровыми изображениями, этого добиться легко — просто установите ширину элемента `img` на 100%, а его высоту — равной `auto` по умолчанию. Браузер по заданной ширине автоматически рассчитывает высоту и определяет *соотношение сторон* (отношение ширины к высоте) изображения на основе его размеров в пикселях. Это позволяет пропорционально масштабировать изображение, не растягивая и не искажая его.

При работе с SVG все реализуется немного сложнее. Во-первых, вы должны представить своей графике четкое соотношение сторон. Изображения SVG могут быть нарисованы в любом размере и не располагают внутренними соотношениями сторон по умолчанию. Во-вторых (в некоторых случаях), вам будет необходимо сгладить ошибки браузера, связанные с автоматическим расчетом высоты.

Область просмотра и окно просмотра

Чтобы разобраться, каким образом масштабируются изображения SVG, нужно четко понимать разницу между областью просмотра SVG (`viewport`) и его окном просмотра (`viewbox`). *Область просмотра*, определяемая атрибутами `width` и `height` элемента

РАЗМЕР ОБЛАСТИ ПРОСМОТРА

Когда SVG встроен с помощью элементов `img`, `object` или `iframe`, атрибуты `width` и `height` этих элементов устанавливают размер области просмотра.

svg, похожа на окно, через которое вы можете видеть область рисования.

Мы можем представить себе область просмотра как небольшое окно браузера (которое мы в этой книге «областью

просмотра» и называли), а можем как элемент `iframe`, отображающий HTML-документ. Точно так же, как нет никакой гарантии, что весь HTML-документ поместится в окне браузера (в элементе `iframe`), нет и гарантии, что *весь* рисунок SVG идеально впишется в его область просмотра. Он может оказаться меньше, а может оказаться больше и тогда будет обрезан. Размеры области рисования (также называемой *пользовательским пространством*) и размеры области просмотра не зависят друг от друга.

Ранее в этой главе мы говорили о том, что область просмотра использует набор координат (*систему координат области просмотра*), которые начинаются с нулевой позиции в верхнем левом углу и возрастают вправо и вниз. Пространство рисования имеет собственный набор координат — *пользовательскую систему координат*, которая устанавливается с помощью атрибута `viewBox` в элементе `svg` и работает аналогичным образом. Именно атрибут `viewBox` является ключом к возможностям адаптивного масштабирования при сохранении соотношения сторон рисунка.

Синтаксис атрибута `viewBox` следующий:

```
viewBox="x y width height"
```

Значения `x` и `y` определяют положение верхнего левого угла окна просмотра в области просмотра. Атрибуты `width` и `height` задают размеры и устанавливают соотношение сторон. Все значения могут быть разделены пробелами (как показано здесь) или запятыми. Координаты `x` и `y` могут иметь отрицательные значения, но ширина и высота всегда должны быть положительными целыми числами.

Документ SVG в следующем примере имеет и область просмотра (с ее значениями `width` и `height`), и окно просмотра (`viewBox`), установленное в значение 400×500 пикселов:

```
<svg width="400" height="500" viewBox="0 0 400 500">
    <!-- Контент рисования -->
</svg>
```

В этом примере окно просмотра соответствует области просмотра (такое поведение задано по умолчанию, если атрибут `viewBox` опущен), но давайте посмотрим, что произойдет, если изменить размеры окна просмотра. Сохраняя размеры области просмотра, попробуем уменьшить размер окна просмотра наполовину (`viewBox="0 0 200 250"`). В результате размер рисунка в области просмотра увеличивается в два раза (рис. ЦВ-25.16). Две системы координат (пользовательская и области просмотра), которые по умолчанию одинаковы, теперь имеют разные масштабы: одна пользовательская единица окна просмотра стала равна двум единицам области просмотра. Цель этого примера — продемонстрировать, что размеры и системы координат области просмотра и окна просмотра функционируют независимо друг от друга и могут использоваться для настройки масштаба.

АТРИБУТ VIEWBOX

Атрибут `viewBox` служит ключом к возможностям адаптивного масштабирования при сохранении соотношения сторон рисунка.

Атрибут `viewBox` служит ключом к возможностям адаптивного масштабирования при сохранении соотношения сторон рисунка.

Я могла бы занять половину главы описанием возможностей и эффектов атрибута viewBox, потому что очень хочу побольше рассказать вам об этом, но экономии места ради рекомендую замечательный учебник Сары Соуэдан (Sara Soueidan) «Understanding SVG Coordinate Systems and Transformations (Part 1) — The viewport, viewBox, and preserveAspectRatio» (www.sarasoueidan.com/blog/svg-coordinate-systems/). Подробно эта тема также раскрыта в книгах по SVG, рекомендованных в конце главы.

Однако сейчас вам просто надо иметь в виду, что атрибут viewBox служит ключом к определению ширины и высоты изображения SVG, и получающаяся в результате система пользовательских координат этого изображения будет отображаться в «окне» области просмотра независимо от его собственных размеров.

Атрибут *preserveAspectRatio*

Атрибут *preserveAspectRatio* в элементе *svg* отвечает за то, чтобы рисунок сохранял соотношение сторон, поэтому он весьма важен для создания SVG с пропорциональной адаптацией. Удобно, что он включен по умолчанию, и вам не придется специально включать его для получения желаемого эффекта. Установка его в значение *none* позволит вам растягивать и сжимать рисунок так, как вам понадобится, — будто вы имеете дело с растровым изображением.

Атрибут *preserveAspectRatio* также принимает значения ключевых слов, определяющих выравнивание окна просмотра в области просмотра. Они функционируют так же, как и процентные значения для CSS-свойства *background-image-position*:

```
<svg viewBox="0 0 300 200" preserveAspectRatio="xMaxYMax meet">
```

В приведенном примере значения *xMaxYMax* смещают окно просмотра максимально вправо и к нижней оси области просмотра (Макс эквивалентен 100%). Ключевое слово *meet* определяет, что размер окна просмотра должен соответствовать высоте или ширине области просмотра (поведение, аналогичное задаваемому атрибутом *contain* для фоновых изображений). Альтернативу ему определяет ключевое слово *slice*, которое масштабирует рисунок таким образом, что он покрывает всю область просмотра, даже если часть рисунка обрезается (поведение, аналогичное атрибуту *cover* для фоновых изображений).

И снова я рекомендую вам ознакомиться с возможностями атрибута *preserveAspectRatio* самостоятельно, сейчас же вам достаточно знать, что он выполняет работу по сохранению соотношения сторон без изменений, даже если воспользоваться заданными по умолчанию настройками и не подключать их явным образом.

Теперь, когда вы познакомились с работающими «за кулисами» механизмами масштабирования SVG, давайте рассмотрим методы, позволяющие сделать SVG-файлы на веб-страницах пропорционально масштабируемыми в зависимости от изменяющегося размера контейнера.

Адаптивные SVG, встроенные с помощью элементов *img* и *object*

Если вы встраиваете изображение SVG в исходный документ с помощью элементов *img* или *object*, процесс настройки его автоматического масштабирования весьма прост.

Прежде всего надо убедиться, что элемент *svg* в файле SVG включает атрибут *viewBox*, определяющий размер рисунка. По умолчанию соотношение сторон сохраняется, даже если не подключать атрибут *preserveAspectRatio*. Если необходимо, чтобы графика полностью заполняла ширину своего контейнера, опустите в элементе *svg* атрибуты *width* и *height*, поскольку они по умолчанию равны 100%, и такое их поведение нас вполне устраивает.

Рассмотрим встраиваемый рисунок элемент *img* или *object*. Ширина и высота, задаваемые в этом элементе, определяют размер, с которым SVG отображается в разметке. Поскольку свойства

width и *height* по умолчанию заданы как *auto*, если вы их опустите, изображение SVG примет полный размер (100%). Если задать только *width* или только *height*, браузер для вычисления того размера изображения, который не определен, воспользуется соотношением сторон окна просмотра.

Опора на размеры по умолчанию элементов *img* или *object* вполне прилично срабатывает в современных браузерах, но лучшие результаты можно получить при обращении к старым браузерам (в частности, к Internet Explorer), если явно задавать значения *width* и *height* элемента встраивания:

```
img {  
    width: 100%;  
    height: auto;  
}
```

Подобный подход лишен случайностей. В результате мы получаем SVG-файл с заданными по умолчанию значениями ширины и высоты, на 100% заполняющий элемент *img*, предусматривающий пропорциональное заполнение ширины его контейнера.

В следующем примере я встраиваю изображение SVG с помощью элемента *img* в элемент *div*, который здесь установлен на 50% ширины окна браузера, как это может иметь место в адаптивном макете

АТРИБУТЫ WIDTH И HEIGHT ДЛЯ ЗНАЧКОВ

Атрибуты *width* и *height* должны быть включены, если вы хотите установить для SVG фиксированный размер — это, например, касается значков, которые должны оставаться в одном размере независимо от разметки.

РАЗМЕРЫ, ЗАДАННЫЕ ПО УМОЛЧАНИЮ В СПЕЦИФИКАЦИИ HTML5

Если для встраиваемого объекта опущены значения *width* и *height*, Internet Explorer заставляет изображение SVG заполнять контейнер по ширине, но при этом высота изображения составит 150 пикселов. Дело в том, что спецификация HTML5 определяет заданные по умолчанию размеры для встроенных носителей, таких, как элементы *img*, *object* и *iframe*, ширину — 300 пикселов и высоту — 150 пикселов. Таким образом, высота, равная 150 пикселам в IE для SVG, не случайна — здесь учтено значение высоты, взятое из спецификации HTML5.

(рис. 25.17). Обратите внимание, что в исходном SVG-коде отсутствуют атрибуты `width` и `height`, а соотношение сторон определяет `viewBox`. По мере того как изменяются размеры окна (оно становится больше или меньше), SVG масштабируется вместе с ним, сохраняя четкое векторное качество при всех значениях масштабов. Контур на `div` отображает его границы, и я установила боковые поля в `auto`, что позволяет представлять его в окне браузера по центру:

SVG-разметка (файл `flowers.svg`)

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" viewBox="0 0 160 120">
    <!-- flower drawing -->
</svg>
```

HTML-разметка

```
<div class="container">
    
</div>
```

Правило стиля

```
.container {
    width: 50%;
    outline: 1px solid gray;
    margin: 2em auto;
}
/* Заплатка для IE */
img {
    width: 100%;
    height: auto;
}
```

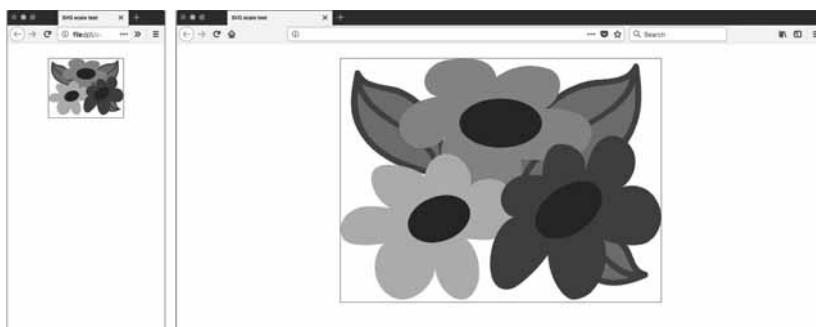


Рис. 25.17. SVG масштабируется автоматически и пропорционально размеру соответствующего контейнера

Адаптивные встроенные SVG-изображения

Скажу сразу, что сохранение соотношения сторон SVG, встроенных с помощью элемента `svg`, не так просто организовать, как это показано в предыдущем примере.

Чтобы браузеры подчинились, вам придется применить некий хитроумный прием, который я сейчас постараюсь объяснить максимально доходчиво.

Прежде всего, как и в случае со встроенным носителем (`img` или `object`), подключите атрибут `viewBox` и опустите в элементе `svg` атрибуты `width` и `height` (используя их заданные по умолчанию значения), если необходимо, чтобы ширина изображения заполняла контейнер на 100%.

В отсутствие правила стиля, специально переопределяющего заданные по умолчанию значения `svg`, ширина изображения получится 100%, и соответствующее соотношение сторон сохранится. Современные версии браузеров, как и ожидается, поддерживают такое поведение, пропорционально масштабируя SVG в соответствующем контейнере. Однако более старые браузеры (в число которых входят все версии Internet Explorer) этому не следуют, поэтому для сохранения пропорций встроенного SVG необходимо применить то, что на сленгге веб-дизайнеров называется «padding hack» («отступной взлом»).

Суть этого приема не вполне очевидна, но я постараюсь провести вас через него по шагам (рис. 25.18):

1. Поместите элемент `svg` в контейнер `div`.
2. Установите значение `height` для этого `div` в 0 или в 1 пиксел.
3. Примените к вершине элемента `div` такое количество отступов, которое придаст ей то же соотношение сторон, что и в SVG (здесь совсем немного математики).
4. Увеличив `div` до правильных пропорций с помощью отступов, поместите элемент `svg` строго (absolute) в верхний левый угол контейнера `div`.



Рис. 25.18. «Padding hack» позволяет встроенным SVG-изображениям сохранять соотношение сторон во всех поддерживающих SVG браузерах

ЕЩЕ О БРАУЗЕРНОЙ ПОДДЕРЖКЕ

Как масштабирование с помощью «отступного взлома», так и нормальное автоматическое масштабирование с соблюдением соотношения сторон могут нарушаться в макетах Grid и Flexbox. По состоянию на начало 2018-го года в поддержке их браузерами существуют серьезные различия, поэтому обязательно тщательно тестируйте свои разработки.

всегда поддерживающих одни и те же соотношения. При этом устанавливается и постоянное соотношение сторон для заполнения SVG.

Рассмотрим пример, показывающий, как работает «отступной взлом»:

Разметка

```
<div class="container">
  <svg version="1.1" viewBox="0 0 160 120">
    /* содержимое рисунка */
  </svg>
</div>
```

Правило стиля

```
.container {
  width: 100%;
  height: 0;
  padding-top: 75%; /* (120/160)*100% */
  position: relative;
}
svg {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```

В приведенной здесь разметке показан элемент `svg` в его контейнере `#container div`, а размеры окна просмотра установлены в 160×120 пикселов. Ширина контейнера установлена в 100% (она также может устанавливаться и с другим процентным значением) и высотой 0. Объявление `padding-top` расширяет `div` до подходящего соотношения сторон, которое вычисляется путем деления высоты SVG на его ширину и умножения результата на процентную ширину `div`:

$$(\text{svg-height} / \text{svg-width}) * \text{div-width\%}$$

Заполнение здесь установлено на 75% [$(120/160) \times 100\% = 75\%$], что соответствует соотношению сторон для SVG 4:3. Обратите также внимание, что контейнер относительно позиционирован, чтобы создать позиционируемый контекст для дочернего элемента `svg` (вспомните, в главе 15 мы подчеркивали, что элементы превращаются в контексты позиционирования с помощью свойства `position: relative`).

Этот прием несколько сложен (не зря его называют «взломом»!), но реально работает. В его основе лежит определенное поведение отступов. Величина отступов, когда она указана в процентных значениях, всегда основывается на ширине элемента (даже для верхних и нижних значений). Это означает, что по мере изменения ширины контейнера `div` изменяется и величина отступов,

Ну и еще один совет в завершение. Чтобы обеспечить в создаваемом вами пользовательском интерфейсе более настраиваемый интерфейс, не ограничивайтесь простым масштабированием, а измените с помощью адаптивного SVG также и дизайн используемых значков (см. врезку «Адаптивные пиктограммы»).

АДАПТИВНЫЕ ПИКТОГРАММЫ

Проблема простого масштабирования SVG (или любого другого изображения) состоит в том, что при небольших размерах изображений могут теряться их детали. Одним из подходов к решению этой проблемы является предоставление упрощенных версий пиктограмм (значков) или логотипов, если изображение слишком мало. Адаптивные (отзывчивые) пиктограммы используют медиазапросы, а также некоторые умные стили в SVG для проверки размера области просмотра и предоставления соответствующего количества деталей при различных размерах изображения. На рис. 25.19 показан созданный Джо Харрисоном (Joe Harrison) набор пиктограмм с объяснениями, приведенными по адресу: responsiveicons.co.uk.



Рис. 25.19. Пример адаптивных пиктограмм от Джо Харрисона (Joe Harrison)

Есть несколько способов для достижения подобного эффекта. Пиктограмма домика Джо выполнена с помощью SVG-спрайта. Вы также можете в зависимости от размера области просмотра применять открытие и скрытие групп элементов изображения SVG. Следующие статьи содержат ряд инструкций на этот счет:

- «*Making SVGs Responsive with CSS*» от Сары Соуэдан (Sara Soueidan) включает в конце раздел, посвященный созданию адаптивных SVG-файлов с медиазапросами. (tympanus.net/codrops/2014/08/19/makingsvgs-responsive-with-css/);
- «*Rethinking Responsive SVG*» от Ильи Пухальского (Ilya Pukhalski) описывает несколько адаптивных SVG-подходов (www.smashingmagazine.com/2014/03/rethinking-responsive-svg/).

Дополнительные ресурсы

В этом разделе рассказывается о том, что нужно знать о масштабировании SVG в адаптивных макетах. Тема эта весьма обширна, поэтому я рекомендую вам следующие статьи для получения дополнительной справочной информации и знакомства с соответствующими методиками:

- «*How to Scale SVG*» от Амелии Беллами-Ройдс (Amelia Bellamy-Royds) — о применении CSS-приемов (css-tricks.com/scale-svg/);
- «*Making SVGs Responsive with CSS*» от Сары Соуэдан (Sara Soueidan) — о советах Codrops (tympanus.net/codrops/2014/08/19/makingsvgs-responsive-with-css/).

ДАЛЬНЕЙШЕЕ ИЗУЧЕНИЕ SVG

Очевидно, что в этой главе я только прикоснулась к тематике, связанной с масштабируемой векторной графикой. Когда вы решите применять в работе изображения SVG, вам понадобится углубить свои знания в этой области. Имеется множество книг и онлайн-уроков о SVG, но именно приведенные далее источники помогли мне в моем собственном развитии, и я рекомендую их вам от всей души:

- Дж. Дэвид Айзенберг (J. David Eisenberg) и Амелия Беллами-Ройдс (Amelia Bellamy-Royds), «SVG Essentials, 2-е издание» (издательство O'Reilly);
- Амелия Беллами-Ройдс (Amelia Bellamy-Royds), Курт Кэгл (Kurt Cagle) и Дадли Стори (Dudley Storey), «Using SVG with CSS3 and HTML5» (издательство О'Рейли);
- Крис Коуэр (Chris Coyier), «Practical SVG»;
- Разработчик и эксперт по SVG Сара Соуэйдан (Sara Soueidan) написала много полезных статей о SVG в своем блоге по адресу: www.sarasoueidan.com.

КОНТРОЛЬНЫЕ ВОПРОСЫ

Я сообщила вам много информации о SVG. Попробуйте пройти этот тест, чтобы уточнить ваш уровень понимания материала. Ответы приведены в *приложении 1*.

- Первые четыре вопроса относятся к этому элементу SVG:

```
<rect x="0" y="0" width="600" height="400" />
```

1. Что такое `rect`?
 2. Какова функция `x` и `y`?
 3. Какова роль `width` и `height`?
 4. Поясните символ `/`.
- Теперь вернемся к нашим обычным вопросам:
5. Какова основная разница в SVG между обрезкой и маскированием?
 6. Назовите три способа для уменьшения размера SVG-файла.
 7. Какая версия SVG поддерживается наиболее широко?
 - a. SVG Tiny
 - b. SVG 1.1
 - c. SVG 2
 - d. SVG, Electric Boogaloo

8. Назовите три из пяти способов оформления элементов в SVG.
9. Что из следующего может быть использовано для анимации элементов в SVG?
 - a. Специальные элементы SVG для анимации
 - b. JavaScript
 - c. CSS переходы и анимация ключевых кадров
 - d. Все здесь указанное

И ... МЫ ЭТО СДЕЛАЛИ!

Привет! Вот и мы и добрались до конца книги! Я надеюсь, что вам пришелся по душе этот подробный обзор HTML, CSS, JavaScript и изображений. Знаю, что материал книги весьма насыщенный, но не забывайте, что вам не нужно выучить его сразу и, тем более, удерживать все это в голове. Хоть я и сама написала эту книгу, но постоянно возвращаюсь к ней, чтобы освежить в памяти атрибуты, свойства и значения. Однако с практикой многое из этого вы будете знать назубок.

Я надеюсь, что вы найдете массу возможностей применить свои новые знания. Желаю вам удачи в ваших веб-проектах!

ЧАСТЬ VI

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

ОТВЕТЫ

ГЛАВА 1

1. b, d, a, c
2. Консорциум W3C направляет развитие веб-технологий.
3. c, d, a, b
4. Frontend-разработка касается аспектов сайта, которые появляются в браузере или связаны с ним. Бэкенд-разработка включает приложения и базы данных, необходимые на сервере для функционирования сайта.
5. Инструмент FTP служит для передачи файлов между компьютерами через Интернет — например, между вашим локальным компьютером и сервером. Вы можете использовать инструмент FTP, предоставленный вашей хостинговой компанией, встроенный в редактор кода или в виде отдельного приложения.

ГЛАВА 2

1. 1. c; 2. j; 3. h; 4. g; 5. f; 6. i; 7. b; 8. a; 9. d; 10. e

ГЛАВА 3

1. При разработке сайта может встретиться ряд неизвестных факторов. Некоторые из них рассмотрены в этой главе:
 - Размер экрана или окна браузера.
 - Скорость интернет-подключения пользователя.
 - Подключен ли JavaScript.
 - Поддерживает ли браузер определенные функции.
 - Находится ли пользователь за рабочим столом или в пути (контекст и концентрация внимания).
2. 1. c; 2. d; 3. e; 4. a; 5. b

3. Четыре основные категории инвалидности включают:

- Нарушение зрения: убедитесь, что контент является семантическим и расположено в логическом порядке для считывания программой чтения с экрана.
- Нарушения слуха: предоставьте замещающие тексты для аудио- и видеоконтента.
- Нарушение мобильности: используйте средства, помогающие пользователям, не имеющим мыши или клавиатуры.
- Когнитивные нарушения: содержание должно быть простым и четко организованным.

4. Вы использовали бы диаграмму водопада, чтобы оценить эффективность своего сайта в процессе оптимизации.

ГЛАВА 4

1. Тег является частью разметки (скобки и имя элемента), применяемой для разделения элемента. Элемент состоит из содержимого и соответствующих тегов.

2. Рекомендуемая разметка для минимального документа HTML5 имеет следующий вид:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf8">
        <title>Title</title>
    </head>
    <body>
    </body>
</html>
```

3.

a. **Sunflower.html** — Да.

b. **index.doc** — Нет, должно заканчиваться на **.html** или **.htm**.

c. **cooking home page.html** — Нет, здесь не может быть пробелов.

d. **Song_Lyrics.html** — Да.

e. **games/rubix.html** — Нет, в имени не может быть слэшей.

f. **%whatever.html** — Нет, здесь не может быть символа процента.

4.

a. Пропущен атрибут src: ****

b. Косая черта (слэш) пропущена в конце тега: **Congratulations! **

c. В конце тега не должен находиться атрибут:

```
<a href="file.html">linked text</a>
```

d. Косая черта должна иметь другое направление (прямой слэш):

```
<p>This is a new paragraph</p>
```

5. Оставьте комментарий:

<!-- здесь начинается список продуктов -->

ГЛАВА 5

1. Далее приводится разметка для тематического разрыва между этими абзацами:
`<p>People who know me know that I love to cook.</p>
<hr>
<p>I've created this site to share some of my favorite recipes.</p>`
- 2.blockquote является элементом уровня блока, используемым для длинных цитат или цитируемого материала, который может состоять из других блочных элементов. Элемент q (quote) предназначен для коротких кавычек, которые идут в потоке текста и не приводят к разрывам строк.
3. Элемент pre.
4. Элемент ul является неупорядоченным элементом списка. Он применяется для списков, которые не должны отображаться в определенной последовательности. По умолчанию они отображаются с маркерами. Элемент ol представляет упорядоченный список, в котором последовательность имеет значение. Браузер для упорядоченных списков автоматически вставляет номера.
5. Используйте таблицу стилей для удаления маркеров из неупорядоченного списка.
6. `<abbr title="World Wide Web Consortium">W3C</abbr>`
7. dl — это элемент, используемый для идентификации всего списка. Элемент dt используется для идентификации только одного термина в этом списке.
8. Атрибут id используется для идентификации уникального элемента в документе, а имя в его значении может появляться в документе только один раз. Атрибут class применяется для классификации нескольких элементов в концептуальных группах.
9. Элемент article предназначен для автономного блока контента, который подходит для синдинкации (передачи на другой сайт) или может отображаться в другом контексте. Элемент section делит контент на тематически связанные фрагменты.

ГЛАВА 6

Упражнение 6.7

1. `<p>Go to the tapenade recipe</p>`
2. `<p>Try this with Garlic Salmon.</p>`
3. `<p>Try the Linguine with Clam Sauce </p>`

4. <p>About Jen's Kitchen</p>
5. <p>Go to Allrecipes.com</p>

Контрольные вопросы

1. ...
2. ...
3. ...
4. ...
5. ...
6. ...
7. ...
8. ...
9.
10.
11.

ГЛАВА 7

1. Атрибуты `src` и `alt` необходимы для того, чтобы документ оказался действительным. Если атрибут `src` опущен, браузеру не будет известно, какое изображение использовать. Можно оставить значение атрибута `alt` пустым, если альтернативный текст будет бессмысленным или неудобным при чтении в контексте.
2.
3. а) поскольку HTML-документы недействительны, если атрибут `alt` опущен, и
б) `alt` улучшает доступность, предоставляя описание изображения, если оно недоступно или не может быть выведено.
4. Имеются три вероятные причины отсутствия изображения: а) неверный интернет-адрес (URL), в результате чего браузер ищет неправильное место или неправильное имя файла (имена восприимчивы к регистру); б) файл изображения не находится в приемлемом формате; и с) файл изображения не имеет соответствующего расширения (**gif**, **jpg** или **png** с учетом соответствия).
5. x-дескрипторы определяют разрешение экрана, которое применяется для вывода на мониторы с высоким разрешением, w-дескриптор предоставляет фактический размер файла изображения, который применяется браузером, чтобы реализовать лучший выбор на основе ширины области просмотра.
6. Пиксель устройства — это цветной квадратик, из которых и составлен экран. Пиксели CSS (также называются «опорными пикселями») составляют сетку,

которую устройства используют для размещения изображения на экране. Пиксел CSS может состоять из нескольких пикселов устройства.

7. b, c, d, a, d
8. Элемент `picture` предоставляет набор изображений для браузера на выбор. Если область просмотра составляет 480 пикселов или шире, изображение будет отображаться с учетом 80% ширины области просмотра. Для меньших размеров области просмотра изображение заполняет 100% области просмотра. Имеется набор изображений в формате WebP для браузеров, которые их поддерживают, в противном случае браузер делает выбор из набора JPEG.
9. Дисковый кэш — это место, где браузеры временно хранят файлы, загруженные по сети для повторного использования. Использование кэшированных ресурсов устраняет необходимость повторных запросов к серверу для одного и того же файла и может повышать производительность.
10. Преимущества включают простую и известную разметку, отличную поддержку браузера, кэширование изображений и доступные запасные варианты. Недостатки состоят в невозможности манипулирования частями SVG с помощью таблиц стилей или JavaScript.
11. Преимущества встроенных SVG включают возможность использовать все функции SVG (анимация, создание сценариев и применение правил стилей CSS), хорошую поддержку браузера и небольшое число количества запросов к серверу. Недостатки состоят в потенциально громоздких объемах кода в HTML-документе, усложненном обслуживании изображений и отсутствии кэширования.
12. Если это чисто презентационное и не является частью редакционного контента страницы.
13. `image/svg+xml` служит MIME-типов SVG-файла. Возможно, потребуется включить его в качестве значения атрибута `type` в элемент `picture` или использовать его для настройки сервера на распознавание SVG-файлов как изображений.
14. `http://www.w3.org/2000/svg` является указателем на пространство имен SVG, стандартизированное W3C. Отображается как значение атрибута `xmlns` (пространство имен XML) в элементах `svg`.

ГЛАВА 8

1. Таблица сама по себе (`table`), строки (`tr`), ячейки заголовка (`th`), ячейки данных (`td`) и необязательный заголовок (`caption`).
2. Элемент `table` может непосредственно включать элементы `tr`, `caption`, `colgroup`, `thead`, `tbody` и `tfoot`.
3. Элемент `tr` может содержать только некоторое число элементов `th` и `td`.

4. Примените элемент `col`, если необходимо включить дополнительную информацию о структуре таблицы, указать ширину для ускорения отображения или добавить определенные свойства стиля в столбец ячеек.
- 5.
- `caption` должен быть первым элементом внутри элемента `table`.
 - Не должно быть текста непосредственно в элементе `table`, текст должен поступать в `th` или `td`.
 - Элементы `th` должны поступать в элемент `tr`.
 - Второй элемент `tr` имеет пропущенный закрывающий тег.
 - Отсутствует элемент `colspan`; должен быть `td` с атрибутом `colspan`.

ГЛАВА 9

- POST (вследствие вопросов, связанных с безопасностью).
 - POST (потому что используется тип ввода выбора файла).
 - GET (потому что можно добавить результаты поиска в закладки).
 - POST (поскольку это может приводить к длинной текстовой записи).
- Раскрывающееся меню: `<select>`
 - Круглые кнопки: `<input type="radio">`
 - `<textarea>`
 - Восемь флагков: `<input type="checkbox">`
 - Меню прокрутки: `<select multiple="multiple">`
- Атрибут `type` пропущен.
 - `checkbox` не является именем элемента, это значение атрибута `type` в элементе `input`.
 - Элемент `option` не является пустым. Он должен включать значение для каждой опции (например: `<option>Orange</option>`).
 - Необходимый атрибут `name` пропущен.
 - Ширина и высота текстовой области указана с помощью атрибут `cols` и `rows`, соответственно.

ГЛАВА 10

- Вложенный контекст просмотра работает как окно браузера внутри другого окна браузера. Можно создать его с помощью элементов `iframe` или `object` (вам бонус, если вы использовали оба).

2. Атрибут `sandbox` позволяет разработчикам устанавливать ограничения на то, что может делать вложенный контент, и это важно по соображениям безопасности.
3. Указать его с обязательным атрибутом типа в элементе `source` и настроить сервер на распознавание типа носителя.
4. a. контейнер; b. видеокодек; c. видеокодек; d. аудиокодек; e. контейнер; f. видеокодек; g. аудиокодек; h. контейнер.
5. Атрибут `poster` указывает изображение, которое появляется в проигрывателе видео, до того когда оно начнет воспроизводиться.
6. Файл `vtt` — текстовый файл в формате WebVTT, который содержит субтитры, подписи, описания, заголовки глав или метаданные, которые синхронизируются с видео- или аудиофайлами.
7. SVG — векторный формат, а `canvas` — пиксельный (растровый). SVG можно масштабировать без потери качества, а `canvas` зависит от разрешения и плохо масштабируется. Можно присваивать элементам стили в SVG с помощью CSS и воздействовать на них с помощью JavaScript, но манипулировать `canvas` можно только с помощью JavaScript.
8. `strokeRect()` и `fill()`.

ГЛАВА 11

1. селектор: `blockquote`; свойство: `line-height`; значение: `1.5`; объявление: `line-height: 1.5;`
2. Текст абзаца будет серым, поскольку при наличии противоречивых правил одинакового веса будет использоваться последний из приведенных в таблице стилей.
3.
 - a. Используйте одно правило с несколькими объявлениями, примененными к элементу `p`:

```
p {  
    font-family: sans-serif;  
    font-size: 1em;  
    line-height: 1.2em;  
}
```
 - b. Точки с запятой отсутствуют:

```
blockquote {  
    font-size: 1em;  
    line-height: 150%;  
    color: gray;  
}
```

- c. Каждое объявление не следует заключать в фигурные скобки — только весь блок объявлений:

```
body {background-color: black;
      color: #666;
      margin-left: 12em;
      margin-right: 12em;
}
```

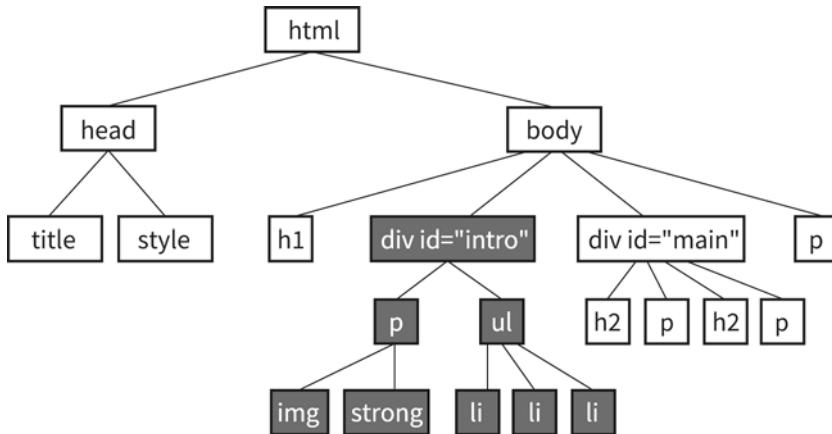
- d. Это можно сделать с помощью одного правила при помощи селектора типа сгруппированного элемента:

```
p, blockquote, li {
  color: white;
}
```

- e. В этом встроенным стиле отсутствует имя свойства:

```
<strong style="color: red">Act now!</strong>
```

4.



ПРИМЕЧАНИЕ. Цветовые стили, примененные к элементу `img`, отображаются только на границе (если она есть).

Рис. П1.1. Ответ к вопросу 4 из главы 11

ГЛАВА 12

1. a. 4; b. 1; c. 7; d. 3; e. 2; f. 9; g. 8; h. 5; i. 6.

2.

- a. `body {color: red;}`
- b. `h2 {color: red;}`
- c. `h1, p {color: red;}`
- d. `.special {color: red; }`
- e. `#intro {color: red;}`

f. `#main strong {color: red;}`
g. `h2 + p {color: red;}`

ГЛАВА 13

1. g. a, b и c
2. d. `rgb(FF, FF, FF)`
3. a. 5; b. 1; c. 4; d. 6; e. 2; f. 3.
4. a. 1; b. 3; c. 2; d. 6; e. 5; f. 4.

ГЛАВА 14

- Ⓐ `border: double black medium;`
- Ⓑ `padding: 2em;`
- Ⓒ `padding: 2em; border: 4px solid red;`
- Ⓓ `border: 4px solid red; margin: 2em;`
- Ⓔ `padding: 1em 1em 1em 6em; border: 4px dashed black; margin: 1em 6em;`
или
`padding: 1em; padding-left: 6em; border: 4px dashed; margin: 1em 6em;`
- Ⓕ `padding: 1em 50px; border: 2px solid black; margin: 0 auto;`

ГЛАВА 15

1. б неправильно. Плавающие элементы расположены напротив края содержимого, а не края отступа.
2. с некорректно. Плавающие элементы не использует свойств смещения, поэтому нет смысла включать `right`.
3. Очистите элемент `footer`, чтобы он начался ниже всплывающей боковой панели с обеих сторон: `footer {clear: both;}`
4. a) абсолютный; b) абсолютный, фиксированный; c) фиксированный; d) относительный, абсолютный, фиксированный; e) статичный; f) относительный; g) абсолютный, фиксированный; h) относительный, абсолютный, фиксированный; i) относительный.

ГЛАВА 16

1. Превратите его родительский элемент в гибкий контейнер, установив для `display` значение `flex`. Дочерний элемент автоматически становится гибким элементом без дополнительного кода.
2. a. 2; b. 4; c. 1; d. 3.

3. Пункты позиций свойства `align-items` позиционируют элементы относительно их flex-линии, в то время как `align-content` распределяет пространство вокруг нескольких линий и между ними. Оба свойства применяются к контейнеру, и оба связаны с позиционированием вдоль поперечной оси.

4. a. 4; b. 1; c. 3; d. 2.

5. a. 4; b. 3; c. 2; d. 5; e. 1.

6. Правила стиля для отображения элементов в порядке, показанном на рис. 16.49:

```
.box6 {order: -1;}
.box1, .box2, .box3 {order: 1;}
```

7. Основное различие между Grid Layout и Flexbox в том, что Grid создает разметки в двух измерениях: строках и столбцах, а Flexbox размещает элементы по одной оси.

Сходство между Grid и Flexbox предполагает:

- Для Grid и Flexbox превращение элемента в контейнер автоматически превращает его прямых потомков в элементы.
- Они оба основаны на языковом направлении документа.
- Оба могут создавать разметки на всю страницу (хотя Grid лучше подходит для этой задачи).
- Можно изменять порядок элементов с помощью свойства `order`.
- Оба используют Модуль Box Alignment для выравнивания элементов и контента.

8. Шаблон сетки для разметки, показанной на рис. 16.50:

```
grid-template-rows: 12em min-content 100px;
grid-template-columns: 300px 2fr 1fr;
```

Использование сокращенного свойства `grid`:

```
grid: 12em min-content 100px / 300px 2fr 1fr;
```

(Примечание: `auto` можно применять вместо `min-content` в обоих примерах.)

9. a. E; b. D; c. C; d. A; e. B.

10.

```
#gallery {
    column-gap: 1em;
}
```

11. d, a, c, b, e.

ГЛАВА 17

1. Адаптивный сайт предоставляет один и тот же HTML-источник по одному и тому же URL-адресу независимо от используемого устройства. Сайт т-дот отправляет документ поциальному URL-адресу, если получает запрос от мобильного устройства. Мобильные сайты также имеют тенденцию сокращать количество параметров и контента на главном экране.

2. Устанавливает размер области просмотра, которую мобильный браузер использует для отображения страницы с тем же размером, что и экран.
3. `img { max-width: 100%; }`
4. Устанавливает поле с левой и правой стороны страницы равным 10%, если область просмотра составляет 60em или шире.
5. Если разметка создана с помощью CSS Grid Layout, используйте единицы `fr` и `minmax()`, что сделает столбцы и строки гибкими при настройке ограничений. Если имеются элементы страницы во Flexbox, используйте flex-свойства, чтобы они могли увеличиваться и уменьшаться по мере необходимости. В противном случае применяйте процентные значения для элементов страницы, чтобы они пропорционально изменяли размер. Избегайте фиксированных измерений в пикселях.
6. Если для медиазапросов применяется `ems`, элементы страницы остаются пропорциональными размеру шрифта. Это может помочь сохранить согласованность длин строк.
7. В правиле `@media` из таблицы стилей, в правилах `@import`, которые вызываются из внешних таблиц стилей и в элементе ссылки на внешнюю таблицу стилей.
8. Используйте разборчивый шрифт, сделайте размер немного меньше, примените более тонкую линию и вводите меньшие поля.
9. Можете использовать аккордеон, чтобы скрыть и показать опции подменю, или поместить подменю на отдельную целевую страницу и сослаться на нее из основной навигации.
10. Протестируйте его на реальных устройствах, используйте эмулятор или обратитесь к сторонней службе тестирования.

ГЛАВА 18

1. Твининг (сглаживание) — это процесс в анимации, когда кадры генерируются между двумя состояниями конечных точек.
2. Переход будет иметь два ключевых кадра: один для начального состояния и один — для конечного:
3.
 - a. `transition-delay: 0.5s;`
 - b. `transition-timing-function: linear;`
 - c. `transition-duration: 0.5s;`
 - d. `transition-property: line-height;`
4. c. `text-transform` неанимируемое свойство.
5. `ease` является функцией времени по умолчанию. Начинается медленно, быстро ускоряется, а затем снова замедляется в самом конце.
6. `.2s` является значением `transition-duration` (продолжительность анимации).
7. Вопрос с подвохом! Они прибудут в то же время, через 300 мс после начала перехода. Функция синхронизации не влияет на общее количество времени, которое необходимо.

8.
 - a. transform: rotate(7deg);
 - b. transform: translate(-50px, -25px);
 - c. transform-origin: right bottom;
 - d. transform: scale(1.25);
9. Значение 3 указывает, что размер элемента должен быть в три раза больше исходного значения *высоты*.
10. perspective: 250; поскольку более низкие числовые значения указывают на более близкое расстояние и являются более существенными.
11. Граница составляет 3 пикселя по ширине при 50% анимации.
12.
 - a. animation-direction: reverse;
 - b. animation-duration: 5s;
 - c. animation-delay: 2s;
 - d. animation-iteration-count: 3;
 - e. animation-fill-mode: forwards;

ГЛАВА 19

1. d. Все из здесь приведенного.
2. d. a и c
3. e. b и d
4. Примените Flexbox или плавающие элементы (floats).
5. a. 2; b. 5; c. 1; d. 4; e. 3.
6. a. нет; b. да; c. нет (если класс .border необходим для отображения стилей).
7. На момент подготовки книги библиотека Modernizr обладала лучшей браузерной поддержкой, чем функция запроса свойств CSS. Как только все браузеры начнут поддерживать функцию запроса свойств CSS, CSS станет работать быстрее и надежнее, чем решение, нуждающееся в привлечении JavaScript.

ГЛАВА 20

1. Это программа, интерпретирующая команды, которые вы вводите в инструмент командной строки.
2. d. Все здесь перечисленное.
3. Стока символов, указывающая, что компьютер готов к приему команды.
4. Вы создали бы новый каталог с именем **newsite** в текущем каталоге.
5. Обеспечение более эффективного синтаксиса *для разработки* (традиционно известной как «предварительная обработка») и *оптимизации* (известной как «постобработка») стандартных файлов CSS.

6. Как только изучите синтаксис, Sass (или LESS и Stylus) могут сделать стили письма более краткими, и их будет легче редактировать. Также сведения о Sass могут потребоваться для некоторых веб-разработок.
7. Общие задачи постпроцессора CSS включают проверку ошибок, добавление префиксов поставщиков, вставку хаков, исправляющих ошибки в старых версиях IE, в том числе применение резервных вариантов для новых функций CSS, преобразование rem в пиксели, преобразование цветовых форматов в RGB и анализ структуры вашего CSS-кода. Этот список не является исчерпывающим, можно придумать и другие функции.
8. Это все, что вы можете сделать вручную из командной строки.
9. Это означает, что исполнитель задач Grunt настроен на «просмотр» файла и что при обнаружении любых изменений в этом файле он автоматически выполняет серию задач.
10. У каждого пользователя имеется локальная копия общего хранилища, с которой он может работать и в автономном режиме.
11. Когда файл подготовлен, это значит, что он добавлен в индекс Git, и Git отслеживает его, но он еще не зафиксирован.
12. Ветвь представляет последовательный ряд подтверждений и отражает поток развития. Развилка — копия чьего-либо не связанного с оригиналом репозитория, над которым можно работать.
13. Выталкивание означает объединение последней копии удаленного главного репо с вашей локальной версией. Вам необходимо получить свежую копию, чтобы убедиться, что вы располагаете самой новой версией, прежде чем отправлять изменения мастеру. Это помогает предотвратить конфликты, если другие пользователи вносят изменения в те же файлы.
14. Запрос на извлечение — это просьба владельцу репо, от которого вы сделали развилку, объединить ваши изменения.

ГЛАВА 21

Упражнение 21.1

```
1. var friends = [ "name" , "othername" , "thirdname" , "lastname" ] ;  
2. alert( friends[ 2 ] ) ;  
3. var name = "yourName" ;  
4. if( name === Jennifer ) { alert("That's my name too! " ) ; }  
5.  
    var myVariable = # ; |  
    if(myVariable > 5) {  
        alert ("upper") ;  
    } else {  
        alert ("lower") ;  
    }
```

Упражнение 21.2

```
<script>
    var originalTitle = document.title;
    function showUnreadCount( unread ) {
        document.title = originalTitle + " (" + unread + " new
        message! )";
    }
    showUnreadCount( 3 );
</script>
```

Контрольные вопросы

1. Когда вы ссылаетесь на внешний файл *.js, можно повторно использовать одни и те же сценарии для нескольких документов. Недостатком служит то, что для этого потребуется дополнительный HTTP-запрос.
2. a. 1; b. 1two; c. 34; d. 2.
3. a. 10; b. 6; c. «2 remaining»; d. «Jennifer is longer. »; e. false (ложь).
4. Он проходит через несколько элементов, начиная с первого в массиве и заканчивая, когда больше не остается элементов.
5. Глобальные переменные могут «сталкиваться» с переменными с такими же именами в других скриптах. Лучше использовать ключевое слово var в функциях, тогда ваши переменные будут иметь локальную область видимости.
6. a. 2; b. 5; c. 4; d. 3; e. 1

ГЛАВА 22

1. Ajax — это комбинация HTML, CSS и JavaScript (с помощью XMLHttpRequest, который применяется для получения данных в фоновой области).
2. Он обращается к элементу со значением id "main".
3. Формирует список узлов из всех элементов section в элементе с помощью id "main".
4. Устанавливает цвет фона на странице (body элемент) как "papayawhip".
5. Формирует новый текстовый узел, который говорит: «Эй, я здесь!», вставляет его во вновь созданный элемент p и помещает новый элемент p в элемент с id "main".
6. d. Все здесь перечисленное.

ГЛАВА 23

1. Можно получить лицензию на эксклюзивные права на изображение, чтобы ваш конкурент не использовал ту же фотографию на своем сайте. Вам также известно, что источник изображения проверен (то есть изображение не украдено).

2. **ppi** означает "pixels per inch" («пиксель на дюйм») и является мерой разрешения.
3. Индексированный цвет — это режим для хранения информации о цвете пикселя. Форматы GIF и PNG-8 являются индексированными цветными изображениями.
4. 256 цветов в 8-битной графике и 32 цвета в 5-битной графике.
5. GIF может включать анимацию и прозрачность. JPEG не может.
6. GIF может содержать анимацию. Обычный PNG не может (хотя может иметь формат APNG).
7. PNG могут иметь несколько уровней прозрачности. GIF имеет только двоичную (включено/выключено) прозрачность.
8. Кумулятивное сжатие с потерями означает, что вы теряете данные изображения каждый раз при сохранении изображения в формате JPEG. Если открыть JPEG и снова сохранить его в формате JPEG, информации об изображении будет утрачено даже больше, чем при предыдущем его сохранении. Обязательно сохраните оригинал высокого качества и делайте с него копии JPEG при необходимости.
9. При двоичной прозрачности пиксель либо полностью прозрачен, либо полностью непрозрачен. Альфа-прозрачность разрешает до 256 уровней прозрачности.
10. **A** GIF или PNG-8, потому что это — текст, однородные цвета и четкие края.
B JPEG, потому что это — фотография. **C** GIF или PNG-8, потому что, хотя в нем есть несколько областей с фотографическими изображениями, большая часть изображения — плоские цвета с резкими краями. **D** GIF или PNG-8, потому что это — плоское графическое изображение. **E** JPEG, потому что это — фотография.

ГЛАВА 24

1. PNG-24 или PNG-8+альфа лучше всего поддерживаются. Форматы WebP и JPEG 2000 также включают альфа-прозрачность, но инструментов для работы с ними мало и они не имеют браузерной поддержки.
2. Настройка качества является наиболее эффективным инструментом для оптимизации JPEG.
3. Уменьшение количества цветов в цветовой палитре оказывает наибольшее влияние на размер индексированных цветных изображений.
4. Дизеринг разбивает сплошные участки цвета и приводит к большим файлам. Дизеринг должен быть отключен или ограничен.
5. Поскольку JPEG-сжатие хорошо работает при плавных переходах цвета и менее хорошо на твердых краях, размытие изображения немного улучшает сжатие и приводит к уменьшению размера файла.
6. sRGB: Да, потому что это кодировка RGB, используемая в Интернете.

7. Если изображение должно выглядеть четким на экранах высокой плотности.
8. Если на вашем сайте много изображений, такие сервисы, как Cloudinary и Akamai, автоматически создают и размещают несколько оптимизированных версий каждого изображения. Они помогают вам избежать создания всех изображений вручную.

ГЛАВА 25

1. `rect` является элементом SVG, который создает прямоугольник.
2. Позиции координат `x` и `y` — координаты позиционирования прямоугольного элемента в верхнем левом углу области просмотра SVG.
3. Атрибуты `width` и `height` устанавливают размеры области просмотра SVG — области, на которой будет отображаться рисунок.
4. В XML все элементы должны быть закрыты. Когда элемент является отдельным элементом (без открывающего и закрывающего тега), он закрывается символом косой черты (/) перед закрывающей скобкой.
5. Обрезка использует векторную форму, чтобы показать или скрыть части изображения. Маскирование основано на пикселях и использует яркость и затемнение растрового изображения, чтобы скрыть или показать замаскированное изображение.
6. Способы уменьшения размера SVG:
 - Упрощение контуров.
 - Сокращение количества десятичных знаков.
 - Использование фигур вместо сложных путей, когда это возможно.
 - Отказ от использования растровых изображений и эффектов в SVG.
 - Запуск через оптимизатор, такой, как SVGO.
 - Подключение Gzip-сжатия на сервере.
7. b. SVG 1.1 (и, к слову, SVG Electric Boogaloo — это не объект).
8. Вы можете выполнить стайлинг SVG с использованием:
 - атрибутов представления SVG;
 - встроенного атрибута `style`;
 - таблицы стилей в самом SVG (элемент `style`);
 - внешней таблицы стилей, вызываемой в SVG (для SVG, помещенных с помощью элемента `img`);
 - если SVG встроен, таблица стилей находится в HTML-документе, в котором выполняется отображение.
9. d. Все здесь перечисленное.

ПРИЛОЖЕНИЕ 2

ГЛОБАЛЬНЫЕ АТРИБУТЫ HTML5

Следующие атрибуты могут применяться совместно с любыми HTML-элементами.

Атрибут	Значения	Описание
accesskey	отдельный текстовый символ	Назначает клавишу доступа (комбинацию клавиш) для ссылки. Клавиши доступа также используются для полей формы. Пользователи могут получить доступ к элементу, нажав комбинацию клавиш <Alt>+<клавиша> (ПК) или <Ctrl>+<клавиша> (Mac). Пример: accesskey= "B"
class	текстовая строка	Назначает одно или несколько классификационных имен для элемента. Несколько значений разделены пробелами
contenteditable	true false	Указывает, может ли пользователь изменять значение элемента. Если значение является пустой строкой, оно определяется как "true". По умолчанию элемент наследует настройку редактирования у своего предка
dir	ltr rtl auto	Определяет направление строчного текста элемента («слева направо» или «справа налево») и ограничивает двунаправленное переупорядочение, изолируя текст от воздействия на окружающий контент. При установке значения auto для определения направления используется первая буква
draggable	true false	Значение true указывает, что элемент можно перетаскивать в пользовательском интерфейсе (событие, сконфигурированное с помощью JavaScript), то есть пользователь может, нажав на нем кнопку мыши и удерживая ее, переместить его на новую позицию в окне
hidden	В HTML — только значение списка: hidden В XHTML — включает имя атрибута: hidden="hidden"	Препятствует отображению элемента и его потомков в пользовательском агенте (браузере). Любые сценарии или элементы управления формой в скрытых разделах по-прежнему будут выполняться, но не будут представлены пользователю

Атрибут	Значения	Описание
<code>id</code>	<i>текстовая строка</i> (может не включать пробелов)	Назначает элементу уникальное идентифицирующее имя
<code>lang</code>	Языковой код ISO (см. www.loc.gov/standards/iso639-2/php/code_list.php)	Задает основной язык для контента элемента и значений его атрибутов. Если атрибут <code>lang</code> не указан, язык элемента совпадает с языком родительского элемента
<code>spellcheck</code>	<code>true</code> <code>false</code>	Указывает, нужно ли проверять орфографию и грамматику элемента. Если атрибут <code>spellcheck</code> не указан, элемент следует заданному по умолчанию поведению для этого элемента, возможно, наследуя проверку состояния родительского элемента
<code>style</code>	Разделенный точкой с запятой список правил стиля (пары <code>property: value</code>)	Связывает информацию о стиле с элементом. Например: <code><h1 style="color: red; border: 1px solid">Heading</h1></code>
<code>tabindex</code>	число (число)	Определяет, что элемент является фокусируемым, и указывает его положение в порядке табуляции для текущего документа. Значение должно быть от 0 до 32767. Применяется для перехода по ссылкам на странице или полям в форме, также удобно использовать для вспомогательных устройств просмотра. Допустимое значение <code>-1</code> позволяет удалять элементы из потока вкладок и делать их фокусируемыми только с помощью JavaScript
<code>title</code>	<i>текстовая строка</i>	Предоставляет заголовок или справочную информацию об элементе, обычно отображаемую в виде всплывающей подсказки. Если заголовок не указан, элемент наследует заголовок от элемента ближайшего предка с заголовком
<code>translate</code>	<code>yes</code> <code>no</code>	Указывает, нужно ли переводить значения атрибутов элемента и значения его дочерних узлов <code>Text</code> при локализации страницы, или оставлять их без изменений. Если не указан, элемент наследует состояние перевода от родителя

ПРИЛОЖЕНИЕ 3

СЕЛЕКТОРЫ CSS УРОВНЯ 3 И 4

В этом приложении приведены селекторы из документа Selectors Level 4 Editor's Draft (drafts.csswg.org/selectors-4/), вышедшего в декабре 2017-го года.

Обратите внимание, что селекторы, помеченные как *уровень 4* являются новыми, поэтому поддержка их со стороны браузеров может отсутствовать. Проверьте ее наличие на сайте CanIUse.com и не забудьте протестировать свою работу перед применением. Все остальные селекторы являются частью CSS3 и, в целом, поддерживаются хорошо.

Селектор	Тип селектора	Описание
Простые селекторы и комбинаторы		
*	Универсальный селектор	Соответствует любому элементу
A	Типовой селектор	Соответствует имени элемента
A, B	Составной селектор	Соответствует элементам А и В
A B	Комбинатор-потомок	Соответствует элементу В, только если он является потомком элемента А
A>B	Комбинатор-наследник	Соответствует любому элементу В, который является дочерним элементом элемента А
A+B	Последующий брат-комбинатор	Соответствует любому элементу В, который следует сразу за любым элементом А, где А и В имеют одного и того же родителя
A~B	Последовательный брат-комбинатор	Соответствует любому элементу В, которому предшествует А, где А и В имеют одного и того же родителя
Селекторы класса и идентификатора		
.classname A.classname	Селектор класса	Соответствует значению атрибута класса во всех элементах или в указанном элементе
#idname A#idname	Селектор идентификатора	Соответствует значению атрибута id в элементе
Селекторы атрибутов		
A[att]	Простой селектор атрибутов	Соответствует любому элементу A, для которого задан этот атрибут, независимо от его значения

Селектор	Тип селектора	Описание
Селекторы атрибутов		
A[att="val"]	Селектор точного значения атрибута	Соответствует любому элементу A, для которого указанный атрибут установлен в указанное значение
A[att="val" i] (уровень 4)	Нечувствительный к регистру селектор значения атрибута	Соответствует любому элементу A, для которого указанный атрибут установлен в это значение, даже если он не соответствует своей заглавной букве (даже в языках XML, которые могут быть восприимчивы к регистру). В следующем примере сопоставляются изображения с именами Icon.png, ICON.png, icon.png и т. д.: <pre>img[src="Icon.png" i] {border: 1px solid yellow;}</pre>
A[att~="val"]	Селектор значения частичного атрибута — сопоставляет любой элемент A, который имеет указанное значение, как одно из значений в заданном для указанного атрибута списке	Соответствует любому элементу A, который имеет указанное значение в качестве одного из значений в списке, заданном для данного атрибута. <pre>table[class~="example"] {background: yellow;}</pre>
A[att = "val"]	Селекторный атрибут префикса дефиса	Сопоставляет любой элемент A, имеющий указанный атрибут, со значением, равным или начинающимся с предоставленного значения. Чаще всего используется для выбора языков, как показано здесь: <pre>a[lang = "en"] {background-image: url(en_icon.png);}</pre>
A[att^="val"]	Селектор начального атрибута подстроки	Соответствует любому элементу A, который имеет указанный атрибут, и его значение начинается с предоставленной строки: <pre>img[src^="/images/icons"] {border: 3px solid;}</pre>
A[att\$="val"]	Селектор конечного атрибута подстроки	Соответствует любому элементу A, который имеет указанный атрибут, и его значение заканчивается с предоставленной строкой: <pre>img[src\$=".svg"] {border: 3px solid;}</pre>
A[att*= "val"]	Селектор атрибута произвольной подстроки	Соответствует любому элементу A, который имеет указанный атрибут, и его значение содержит предоставленную строку: <pre>img[title*="July"] {border: 3px solid;}</pre>

Селектор	Тип селектора	Описание
Селекторы псевдоклассов		
:any-link (уровень 4)	Селектор псевдокласса ссылок	Определяет стиль для ссылки независимо от того, посещалась ли она
:link	Селектор псевдокласса ссылок	Определяет стиль для ссылок, которые еще не посещались
:target	Целевой селектор псевдокласса	Выбирает элемент, который используется в качестве идентификатора фрагмента
:target-within (уровень 4)	Обобщенный целевой селектор псевдокласса	Выбирает элемент, который используется в качестве идентификатора фрагмента или содержит элемент с этими функциями
:visited	Селектор псевдокласса ссылок	Определяет стиль для ссылок, которые уже посещались
:active	Селектор псевдокласса действий пользователя	Выбирает любой элемент, который активирован пользователем, — например, ссылку, с помощью которой этот селектор выбирается
:hover	Селектор псевдокласса действий пользователя	Определяет стиль для элементов (обычно ссылок), которые появляются, когда на них наведена мышь
:focus	Селектор псевдокласса действий пользователя	Выбирает любой элемент, который в текущий момент имеет фокус ввода, — например, выбранную форму ввода
:focus-within (уровень 4)	Обобщенный входной селектор псевдокласса	Выбирает любой элемент, который имеет фокус ввода пользователя или содержит элемент, который имеет фокус ввода
:focus-visible (уровень 4)	Селектор псевдокласса действий пользователя	Выбирает любой элемент, который имеет фокус ввода пользователя, и пользовательский агент определил, что для этого элемента должно быть нарисовано кольцо фокусировки или другой индикатор
:drop(active) (уровень 4)	Перетаскиваемый селектор псевдокласса	Выбирает элемент, который является текущей целью перетаскивания для перетаскиваемого элемента
:drop(valid) (уровень 4)	Перетаскиваемый селектор псевдокласса	Выбирает элемент, который может получить элемент, который перетаскивается в текущий момент
:drop(invalid) (уровень 4)	Перетаскиваемый селектор псевдокласса	Выбирает элемент, который не может получить элемент, который перетаскивается в текущий момент, но может получить другой элемент
:dir(ltr) (уровень 4)	Псевдокласс направленности	Выбирает элемент с определенным направлением письма. В приведенной записи выбрано направление слева направо. Язык документа определяет и способ указания направления

Селектор	Тип селектора	Описание
Селекторы псевдоклассов		
:lang(xx)	Языковой селектор псевдокласса	Выбирает элемент, соответствующий двухсимвольному коду языка. a:lang(de) {color: green;}
:nth-child()	Структурный селектор псевдокласса	Выбирает элемент, который является n-м дочерним элементом своего родителя. Запись может включать число, запись или ключевые слова как нечетные, так и четные
:nth-last-child()	Структурный селектор псевдокласса	Выбирает элемент, который является n-м дочерним элементом своего родителя, считая от последнего
:nth-of-type()	Структурный селектор псевдокласса	Выбирает n-й элемент своего типа
:nth-last-of-type()	Структурный селектор псевдокласса	Выбирает n-й элемент своего типа, считая от последнего
:first-child	Структурный селектор псевдокласса	Выбирает элемент, который является первым дочерним элементом соответствующего родительского элемента
:last-child	Структурный селектор псевдокласса	Выбирает элемент, который является последним дочерним элементом соответствующего родительского элемента
:only-child	Структурный селектор псевдокласса	Выбирает элемент, который является единственным потомком своего родителя
:first-of-type	Структурный селектор псевдокласса	Выбирает элемент, который является первым братом своего типа
:last-of-type	Структурный селектор псевдокласса	Выбирает элемент, который является последним братом своего типа
:only-of-type	Структурный селектор псевдокласса	Выбирает элемент, который является единственным братом своего типа
:root	Древовидно-структурный селектор псевдокласса	Выбирает элемент, который является корнем документа. В HTML — это элемент <code>html</code>
:empty	Древовидно-структурный селектор псевдокласса	Выбирает элемент, который не имеет текста и дочерних элементов

Селектор	Тип селектора	Описание
Селекторы псевдоклассов		
:blank	Древовидно-структурный селектор псевдокласса	Выбирает элемент, который не имеет содержимого, кроме, возможно, пробелов
:enabled	Селектор псевдокласса UI	Выбирает элемент пользовательского интерфейса, если он включен
:disabled	Селектор псевдокласса UI	Выбирает элемент пользовательского интерфейса, если он отключен
:checked	Селектор псевдокласса UI	Выбирает активизированный (отмеченный) элемент пользовательского интерфейса (переключатель или флагок)
:read-write (уровень 4)	Селектор псевдокласса изменчивости	Выбирает элемент пользовательского интерфейса, если он может изменяться пользователем
:read-only (уровень 4)	Селектор псевдокласса изменчивости	Выбирает элемент пользовательского интерфейса, если он не может изменяться пользователем
:placeholder-shown (уровень 4)	Селектор псевдокласса изменчивости	Выбирает элемент управления вводом, в текущий момент отображающий текст заполнителя
:default (уровень 4)	Селектор псевдокласса по умолчанию	Выбирает элемент пользовательского интерфейса, который является заданным по умолчанию элементом, в группе связанных вариантов
:indeterminate (уровень 4)	Селектор псевдокласса с неопределенным значением	Выбирает элемент пользовательского интерфейса, который характеризуется неопределенным состоянием (и не является отмеченным, и не является неотмеченным).
:valid (уровень 4)	Валидность селектора псевдокласса	Выбирает элемент пользовательского интерфейса, соответствующий присущей ему семантике достоверности данных
:invalid (уровень 4)	Валидность селектора псевдокласса	Выбирает элемент пользовательского интерфейса, не соответствующий присущей ему семантике достоверности данных
:in-range (уровень 4)	Диапазон селектора псевдокласса	Выбирает элемент пользовательского интерфейса, значение которого находится в указанном диапазоне
:out-of-range (уровень 4)	Диапазон селектора псевдокласса	Выбирает элемент пользовательского интерфейса, значение которого не находится в указанном диапазоне
:required (уровень 4)	Дополнительный селектор псевдокласса	Выбирает элемент пользовательского интерфейса, который требует ввода

Селектор	Тип селектора	Описание
Селекторы псевдоклассов		
:optional (уровень 4)	Дополнительный селектор псевдокласса	Выбирает элемент пользовательского интерфейса, который не требует ввода.
:not(A)	Отрицание селектора псевдокласса	Выбирает элемент, который не соответствует простому селектору A. Может также использоваться с составными селекторами, в этом случае он выбирает элемент, который не соответствует ни A, ни B: :not(A, B) { color: #ccc; }
:matches(A, B) (уровень 4)	Соответствует любому селектору псевдокласса	Выбирает элемент, который соответствует A и/или B: :matches(h2, h3) { color: #ccc; }
E:has(rA, rB) (уровень 4)		Селектор реляционного (относительного) псевдокласса — выбирает элемент E, если любой из относительных селекторов rA или rB соответствует элементу при оценке с помощью элемента как элементы : scope. В следующем примере сопоставляются только элементы, содержащие img: a:has(> img) { margin: .5em 0; }
Селекторы псевдоэлементов		
::first-letter	Селектор псевдоэлемента	Выбирает первую букву указанного элемента
::first-line	Селектор псевдоэлемента	Выбирает первую строку указанного элемента
::before	Селектор псевдоэлемента	Вставляет сгенерированный текст в начале указанного элемента и применяет к нему стиль
::after	Селектор псевдоэлемента	Вставляет сгенерированный текст в конец указанного элемента и применяет к нему стиль
Грид-структурированные селекторы		
A B (уровень 4)	Грид-структурированный селектор	Выбирает элемент B, представляющий принадлежащую столбцу ячейку в сетке/таблице, которая представлена элементом A

ПРИЛОЖЕНИЕ 4

ОТ HTML+ К HTML5

Я не уверена, что какая-либо спецификация HTML имела столь же шумную славу, как HTML5. Она на самом деле предлагает столько многообещающих возможностей, что ее название стало чем-то вроде модного словечка со смыслом, выходящим далеко за рамки самой спецификации. Когда маркетологи и журналисты используют термин «HTML5», они при этом иногда имеют в виду любую новую веб-технологию, которая заменяет Flash. В этой книге вы познакомились с элементами HTML5, а здесь я подробнее расскажу вам о самой спецификации, так что вы сможете присоединиться к тем из нас, кто раздражен, когда слышит, как «HTML5» используется не по делу. Тем не менее широкая осведомленность о веб-стандартах, безусловно, весьма важна и облегчает нам работу, когда мы общаемся с клиентами.

Но сначала, я думаю, вам надо понять, как мы «дошли до жизни такой» и что делает HTML5 действительно прорывной технологией. И начну я с краткой истории HTML, а затем укажу на некоторые уникальные качества HTML5, включая его API.

КРАТКАЯ ИСТОРИЯ HTML

Представление о том, где мы были, очень важно, для того чтобы понять, куда мы идем. В нашем путешествии по HTML5 мы пересечем границу ранней Паутины, пройдем опасными полями битв Браузерных войн и посетим места показного соперничества с XML.

КАК ВСЕ НАЧИНАЛОСЬ...

История HTML — от первоначального чернового наброска Тима Бернерса-Ли (Tim Berners-Lee) в далеком 1991 году до разработки стандарта HTML5 в наши дни — является одновременно и захватывающей, и бурной. Ранние версии HTML (HTML+ в 1994 году и HTML 2.0

ИСТОРИЯ HTML ПЕРИОДА 1989–1998 ГОДОВ

Подробно история HTML периода 1989–1998 годов описана в выдержке из книги Дэвида Раггетта (David Raggett) «Raggett on HTML4» (издательство Addison-Wesley), доступной на сайте W3C (www.w3.org/People/Raggett/book4/ch02.html).

в 1995-м) основывались на одной из работ Тима и имели целью получить жизнеспособный вариант системы для размещения сетевых публикаций.

Но когда Всемирная паутина (как ее восхитительно тогда называли) захватила мир штурмом, создатели браузеров — сначала Mosaic Netscape, а позднее и Microsoft Internet Explorer — заявили: «Мы не станем ждать ваших дурацких стандартов». И дали людям то, чего они хотели, создав множество специфичных для каждого из браузеров элементов, улучшающих внешний вид страниц. Это жесткое противостояние и получило известность как «Браузерные войны». В результате в конце 1990-х годов стало обычной практикой создание двух совершенно разных версий сайта, ориентированных на каждый из браузеров «Большой Двойки». Пометки на сайтах с надписью «Лучше всего смотреть в Netscape» были нормой. Меня бросает в дрожь, когда я вспоминаю об этом.

Призыв к разуму

В 1996-м году недавно созданный W3C в попытке обрести почву под ногами выпустил свою первую Рекомендацию: HTML 3.2. Это был краткий перечень всех общеупотребительных на то время элементов HTML, включающий в себя и множество презентационных расширений HTML (таких, как элементы `font` и `center`), появившихся в результате вражды Netscape/IE в отсутствие альтернативы в виде таблиц стилей. Версии HTML 4.0 (1998) и HTML 4.01 (с небольшими поправками, заменившая ее в 1999-м году) были направлены на то, чтобы вернуть HTML в правильное русло, подчеркнув разделение структуры и представления и улучшив доступность. Все вопросы представления были при этом переданы недавно разработанному и получившему широкую поддержку стандарту каскадных таблиц стилей.

ПРОЕКТ ВЕБ-СТАНДАРТОВ

В 1998-м году в разгар «Браузерных войн» некое общественное объединение, назвавшее себя «Проект веб-стандартов» (Web Standards Project, WaSP), начало оказывать давление на создателей браузеров (в первую очередь, на Netscape и Microsoft), призывая их придерживаться открытых стандартов, разрабатываемых W3C. Не останавливаясь на этом, оно повело с сообществом веб-разработчиков разъяснительную работу, информируя его о серьезных преимуществах разработок на основе стандартов. Усилия этого объединения привели к революции в сознании веб-разработчиков, открыв для них удобные способы создания и поддержки сайтов. И теперь браузеры (даже Microsoft) уже кичатся поддержкой стандартов, продолжая внедрять инновации.

В 2013-м году WaSP объявил: «Наша работа завершена» — и распался. Об этой миссии, ее истории и справочных материалах можно прочесть на сайте WaSP (webstandards.org).

HTML 4.01 наряду с XHTML 1.0 — более требовательным «родственником» на основе XML (рассматривается далее) превратился в краеугольный камень процесса совершенствования веб-стандартов (см. врезку «Проект веб-стандартов»).

Знакомство с XML и XHTML

Примерно в то же время, когда HTML 4.01 находился в стадии разработки, специалисты из W3C узнали, что один из специализированных языков разметки не собирается учитывать их ограничения по части описания всех видов информации (химическая запись, математические уравнения, мультимедийные презентации, финансовая информация и т. п.), которые могут быть переданы через Интернет. В результате у них родилось решение: XML (eXtensible Markup Language, расширяемый язык разметки) — метаязык для создания языков разметки. Язык XML стал упрощенной версией SGML (Standardized Generalized Markup Language, стандартизированного обобщенного языка разметки), большого семейства метаязыков, которыми пользовался Тим Бернерс-Ли при создании своего первого HTML-приложения. Но язык SGML оказался более сложным, чем это было необходимо для Интернета.

Видение Сети консорциумом W3C заключалось в использовании в ней XML вкупе со многими специализированными языками разметки, функционирующими вместе даже в рамках одного документа. Конечно, чтобы осуществить это, каждый разработчик должен был бы весьма тщательно размещать документы, строго соблюдая синтаксис XML, чтобы исключить возможную путаницу.

Первым шагом на этом пути стало переписывание HTML в соответствии с правилами XML, чтобы он мог хорошо сочетаться с другим языками. В результате появился XHTML (eXtensible, расширенный, HTML). Первая его версия — XHTML 1.0 — почти идентична HTML 4.01, включает те же элементы и атрибуты, но имеет более строгие требования к синтаксису (см. врезку «Требования к разметке XHTML»).

ТРЕБОВАНИЯ К РАЗМЕТКЕ XHTML

Синтаксис XHTML соответствует строгим требованиям XML-разметки:

- имена элементов и атрибутов должны быть строчными (записаны в нижнем регистре). Тогда как в HTML имена элементов и атрибутов не восприимчивы к регистру символов;
- все элементы должны быть закрыты (завершены). Вы закрываете пустые элементы, добавляя косую черту (слэш) перед закрывающей скобкой (например:
);
- значения атрибута должны заключаться в кавычки. Допускаются одинарные или двойные кавычки, если они применяются корректно. Кроме того, перед значением атрибута и после него внутри кавычек не должно быть никакого лишнего пространства (символьных пробелов или символов возврата строк);
- все атрибуты должны иметь явные значения атрибутов. XML (и, следовательно, XHTML) не поддерживает минимизацию атрибутов — практику SGML, когда определенные атрибуты могут сокращаться только до значения атрибута. Таким образом, если в HTML можно написать `check`, указывая, что кнопка формы проверяется при загрузке формы, в XHTML необходимо указать: `check="checked"`;
- строго соблюдаются правильное вложение элементов. Некоторые имеющиеся элементы получили новые ограничения на вложение;
- начальные и конечные теги обязательны;

- ▶ • специальные символы всегда должны представляться символьными объектами (например: & для символа &);
- сценарии должны содержаться в разделе CDATA, чтобы они обрабатывались как простые текстовые символы, а не анализировались как XML-разметка. Это показано в следующем примере:

```
<script type="type/javascript">
// <![CDATA[
... Сценарий JavaScript...
// ]]>
</script>
```

В то время как HTML-анализаторы прощают неправильную разметку, ошибки в синтаксисе XHTML останавливают синтаксический анализатор на таких строках. Пропустить код XHTML через валидатор — отличный способ выявления синтаксических ошибок перед запуском страниц.

Но W3C на этом не остановился. Учитывая свое видение Сети на основе XML, они начали работу над XHTML 2.0 — еще более смелой попыткой заставить все работать «правильнее», чем в HTML 4.01. Проблема возникла в том, что он не был обратно совместим со старыми стандартами и поведением браузеров. Процесс написания и утверждения XHTML 2.0 без привлечения браузеров затянулся на годы. Так и не обретя поддержку браузеров, XHTML 2.0 в этом и увяз.

Привет HTML5!

Тем временем...

В 2004-м году Apple, Mozilla и Opera создали рабочую группу по технологиям веб-гипертекстовых приложений (WHATWG, Web Hypertext Application Technology Working Group, whatwg.org), причем отдельно от W3C. Главной целью WHATWG было дальнейшее развитие HTML для удовлетворения новых требований таким образом, чтобы это соответствовало реальной практике авторских разработок и поведению браузеров (в отличие от идеала «с нуля», описанного в XHTML 2.0). Исходные документы этой группы: Web Applications 1.0 и Web Forms 1.0 — легли в основу в HTML5. И сегодня WHATWG поддерживает всемогущий «живой» стандарт HTML.

В конечном итоге на основе работы, проделанной WHATWG, W3C создал свою рабочую группу по HTML5 (HTML5 Working Group), и в октябре 2014-го года HTML5 получил официальный статус Рекомендации. По состоянию на начало 2018-го года последней версией HTML5 является HTML 5.2 в ранге Proposed Recommendation («Предлагаемой рекомендации»).

Работа над спецификацией HTML5 ведется обеими организациями в тандеме, иногда с небольшими несоответствиями. В большинстве случаев поставщики браузеров ориентируются на вариант WHATWG.

А что же XHTML 2.0? В конце 2009-го года консорциум W3C официально «положил конец его страданиям», распустив рабочую группу и сосредоточив свои ресурсы и усилия на совершенствовании HTML5.

Вот так мы здесь и оказались. Теперь поглубже познакомимся с HTML5.

HTML5: БОЛЬШЕ, ЧЕМ РАЗМЕТКА

Предшествующие версии HTML касались в основном элементов разметки контента, предназначенного для просмотра на веб-страницах. HTML5 — это набор новых методов для решения таких задач, которым ранее требовалось специальное программирование или запатентованная технология подключаемых модулей, таких, как Flash или Silverlight. Решения HTML5 включают компоненты разметки и сценариев, предлагают применение API для таких действий, как размещение на странице аудио и видео, локальное сохранение данных, работу в автономном режиме, использование информации о местоположении и многое другое. Применяя HTML5 для решения общих задач, разработчики могут опираться на встроенные возможности браузеров и «не изобретать велосипед» для каждого приложения.

Многие из новых возможностей HTML5 требуют усовершенствованных навыков веб-разработки, поэтому вряд ли вы воспользуетесь ими сразу (если вообще это когда-либо произойдет), но, как всегда, я полагаю, что каждому полезно иметь базовое представление об этих возможностях. Именно «базовое представление» — вот моя задача сейчас. Более подробное знакомство с возможностями HTML5 вам обеспечат источники, упомянутые во врезке «Дополнительное чтение».

ДОПОЛНИТЕЛЬНОЕ ЧТЕНИЕ

Следующие книги помогут вам пополнить знания о HTML5:

- Рэйчел Эндрю (Rachel Andrew) и Джереми Кит (Jeremy Keith) «*HTML5 for Web Designers*», 2-е издание;
- Марк Пилигрим (Mark Pilgrim) «*HTML5: Up and Running*» (издательство O'Reilly и Google Press);
- Брюс Лоусон (Bruce Lawson) и Реми Шарп (Remy Sharp) «*Introducing HTML5*», 2-е издание (издательство New Riders).

Компонент разметки HTML5

В этой книге вы изучали элементы и атрибуты HTML5.

Язык HTML5 основан на HTML 4.01 Strict (Строгий) — версии HTML, которая не содержит каких-либо презентационных или других устаревших элементов и атрибутов. Это означает, что подавляющая часть языка HTML5 включает известные элементы, которые использовались годами, и браузерам известно, что с ними делать.

Однако язык HTML5 представил и ряд новых элементов, типов форм ввода и глобальных атрибутов. Это сделало многие несоответствующие им элементы и атрибуты HTML 4.01 официально устаревшими.

HTML5 И XML

На HTML5 также можно выполнять разработки, следуя более строгому синтаксису XML (так называемая *XML-сериализация HTML5*). Некоторые разработчики предпочитают аккуратно оформленный XHTML (имена элементов в нижнем регистре, значения атрибутов в кавычках, закрытие всех элементов и т. д.), и такой способ записи по-прежнему возможен, хотя и не обязательен. Впрочем, в ряде случаев может потребоваться, чтобы документ HTML5 — для обеспечения совместной работы с другими XML-приложениями — был написан в соответствии с требованиями XML, и в таких случаях этот документ должен использовать XML-синтаксис.

Одно из отклонений HTML5 от предыдущих версий HTML заключается в том, что HTML5 стал первой спецификацией, содержащей подробные инструкции, как браузеры должны обрабатывать плохо организованную и устаревшую разметку. Они основаны на инструкциях по поведению устаревших браузеров, но на этот раз предусматривают стандартный протокол, которому производители браузеров должны следовать, когда их браузеры сталкиваются с неправильной или нестандартной разметкой.

Тип *DOCTYPE* без *DTD*

Как отмечалось в главе 4, HTML-документы должны начинаться с объявления типа документа (*DOCTYPE*), определяющего, какой HTML-версии следует документ. Объявление HTML5 является коротким и понятным:

```
<!DOCTYPE html>
```

Сравните это с объявлением для документа Strict HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/HTML4.01/strict.dtd">
```

Почему столь сложно? Для документов, написанных с помощью HTML 4.01 и XHTML 1.0 и 1.1, объявление должно указывать на общедоступный *DTD* (Document Type Definition, Определение типа документа) — документ, описывающий все элементы языка разметки и правила их применения. DTD является «пережитком» SGML и менее полезен в Сети, чем предполагалось первоначально, поэтому разработчики HTML5 просто не использовали его. В результате объявление *DOCTYPE* стало намного проще.

И HTML 4.01, и XHTML 1.0 имели три отдельных DTD (для версий Traditional, Strict и Frameset каждой из спецификаций), поэтому приходилось отслеживать большое число мелких деталей. Для получения полного списка объявлений *DOCTYPE* (включая ссылки DTD) для HTML 4.01, XHTML, SVG и других типов документов обратитесь по адресу: www.w3.org/QA/2002/04/valid-dtd-list.html.

Для того чтобы проверить, соответствует ли ваш документ правилам той спецификации, которой он должен следовать, применяйте декларацию *DOCTYPE*

и используйте *валидаторы* — программное обеспечение, проверяющее правильность всей разметки в документе.

Знакомство с API

Спецификации HTML, предшествовавшие HTML5, только документировали разрешенные на языке элементы, атрибуты и значения. Это хорошо для простых текстовых документов, но разработчики HTML5 сосредоточились на том, чтобы упростить создание веб-приложений, требующих сценариев и программирования. По этой причине в HTML5 определен ряд новых API — интерфейсов, облегчающих взаимодействие с приложением.

API (Application Programming Interface, интерфейс прикладного программирования) — документированный набор команд, имен данных и т. п., позволяющий одному приложению взаимодействовать с другим. Например, разработчики Twitter описали имена каждого типа данных (пользователи, твиты, отметки времени и т. д.) и методы доступа к ним в документе API (dev.twitter.com/docs), и на его основе другие разработчики получили возможность подключать каналы и элементы Twitter в свои программы. Поэтому существует столь много Twitter-программ и виджетов. Amazon.com также открывает данные о своем продукте через API. Да и большинство других издателей также признают, что их контент доступен через API.

Вернемся в HTML5, который включает API-интерфейсы для задач, традиционно требующих проприетарных плагинов (таких, как Flash) или пользовательского программирования. Основная идея состоит в том, что если браузеры предлагают такие возможности изначально — со стандартизованными приемами доступа к ним — разработчики могут задействовать всевозможные изящные подходы и рассчитывать, что они сработают во всех браузерах, точно так же, как мы сегодня рассчитываем на возможность легко вставлять на страницы изображения. Разумеется, у нас есть различные обходные пути, которыми мы можем пользоваться, пока эти передовые возможности не получат повсеместную поддержку, но стремимся мы именно к ним. Некоторые API содержат компоненты разметки — например, встраивание мультимедиа с помощью новых элементов HTML5 `video` и `audio`. Другие действия выполняются полностью «за кулисами» с помощью JavaScript или серверных компонентов — например, создание веб-приложений, которые работают, даже когда нет подключения к Интернету (Offline Web Application API, API автономных веб-приложений).

Консорциум W3C и группа WHATWG работают над множеством API-интерфейсов для использования с веб-приложениями, и все они находятся на разных стадиях завершения и внедрения. Многие из них имеют собственные спецификации, отдельные от спецификации HTML5, но они, как правило, прикрыты широким «зонтиком» HTML5, охватывающим веб-приложения. HTML5 включает следующие спецификации для таких API:

ВАЛИДАТОРЫ

Чтобы проверить, является ли HTML-документ действительным, используйте онлайн-валидатор консорциума W3C (validator.w3.org). Валидатор, специфичный для HTML5, также доступен по адресу html5.validator.nu.

ПОЛЕЗНАЯ ОСОБЕННОСТЬ HTML5

HTML5 стремится сделать HTML более удобным для создания веб-приложений. По этой причине в HTML5 определен ряд новых API — интерфейсов, облегчающих взаимодействие с приложением.

- *Media Player API* — для управления аудио- и видеопроигрывателями, встроенными в веб-страницу с помощью новых элементов `video` и `audio`;
- *Editing API* — предоставляет набор команд, которые можно использовать для создания в браузере текстовых редакторов, позволяющих пользователям вставлять и удалять текст, форматировать текст полужирным шрифтом, курсивом или гипертекстом и выполнять другие действия. Кроме того, имеется новый атрибут `contenteditable`, позволяющий прямо на странице редактировать любой элемент контента;
- *Session History API* — предоставляет историю браузера с использованием возможностей кнопки **Back** (Назад);
- *API (Drag and Drop API)* — добавляет возможность перетаскивания выделенного текста или файла в целевую область на текущей веб-странице или на какой-либо другой. Атрибут `draggable` определяет элемент для выделения и перетаскивания.

А далее упомянуто лишь несколько API-интерфейсов, разрабатываемых в W3C с собственными спецификациями:

- Canvas API — элемент `canvas` добавляет динамическое двумерное пространство для рисования на странице. Элемент `canvas` мы рассматривали в главе 10;
- Service Workers API — в этой спецификации описывается метод, позволяющий веб-приложениям функционировать в автономном режиме;
- Web Storage API — позволяет сохранять данные в кэше браузера, что позволяет приложению применять их позже. Традиционно это выполнялось с помощью файлов «cookie», но веб-хранилище API (Web Storage API) дает возможность сохранять больше данных. Оно также определяет, ограничены ли данные одним сеансом (`sessionStorage` — когда окно закрыто, данные очищаются) или основаны на домене (`localStorage` — все открытые окна, указанные в этом домене, имеют доступ к данным);
- Geolocation API — позволяет пользователям делиться своим географическим местоположением (долготой и широтой), чтобы оно стало доступно для сценариев в веб-приложениях. При этом приложение предоставляет функции, учитывающие местоположение пользователей, — например, предлагает близлежащий ресторан или находит других пользователей именно в вашем регионе;

Web Sockets API — создает *сокет*, который служит открытым соединением между клиентом браузера и сервером. Это позволяет передавать информацию между кли-

СПИСКИ ВСЕХ API

Для ознакомления со списком всех API обратитесь к документу, написанному Эриком Уайльдом (Erik Wilde) «HTML5 Overview» (html5-overview.net). Консорциум W3C также ведет список всех поддерживаемых им документов, многие из которых являются API, на сайте, доступном по адресу: www.w3.org/TR/tr-title-all.

ентом и сервером в режиме реального времени без каких-либо задержек, характерных для традиционных HTTP-запросов. Вы можете представить себе веб-сокет как продолжительное телефонное соединение между браузером и сервером — в отличие от общепринятого стиля радиообмена (прием-передача)

между браузером и сервером (снимаю шляпу перед Джен Симмонс за такую аналогию). Это полезно для многопользовательских игр, чатов или постоянных новостей, тикеров или социальных сетей.

КУДА ДВИГАТЬСЯ ДАЛЬШЕ?

Имея в наличии систему, которая, кажется, работает, следует, тем не менее, ожидать, что спецификация HTML будет по-прежнему подвергаться незначительным доработкам, чтобы соответствовать изменяющимся требованиям, учитываяющим то, как мы используем Сеть и устройства, с помощью которых мы в нее входим. Эти незначительные изменения, как правило, включают добавление атрибутов, элементов и API, а также еще не реализованных функций. Другими словами, я не слышала пока разговоров о таком сейсмическом сдвиге, как переход с XHTML на HTML5.

Консорциум W3C присвоил HTML 5.2 (5-я основная версия HTML и вторая ее небольшая редакция) статус Рекомендации 14 декабря 2017-го года. На момент выхода книги из печати существует рабочий проект HTML 5.3. Между тем группа WHATWG поддерживает свою «живую» версию HTML, которая постоянно обновляется и служит основой для большинства поставщиков браузеров.

Не забывайте, что HTML все время развивается, и как веб-дизайнеру или разработчику вам необходимо следить за производимыми в нем изменениями. Так что новые возможности, достойные изучения, будут появляться всегда.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

АВД 591
Авторинг 31
Адаптивное изображение 813
Адаптивный
 веб-дизайн 66, 591
 шаблон макета 610
Альтернативный текст 189
Альфа-канал 808
Альфа-прозрачность 778, 806
Анимация 781
 CSS 850
 JavaScript 850
 SMIL 850
 контуров 851
 по ключевым кадрам 650
Аргумент 687
Артефакт 773
Ассоциация
 неявная 271
 явная 271
Атрибут 96
 action 243
 alt 187, 189
 class 144
 colspan 228
 headers 232
 height 190
 href 161, 162
 id 143
 method 243
 name 245
 rowspan 228, 229
 sandbox 287
 scope 232
 sizes 208
 src 187, 188
 srcset 206
 width 190, 191
 глобальный 146
 логический 98
 простой селектор 411
 селектор точного значения 411
 селектор частичного значения 411
Аудиоформат
 MP3 293
 MPEG4 audio 293

Ogg Vorbis 293
WAV 293

Б

Базовая линия 372
Библиотека JavaScript 758
Битовая глубина 802, 830
Битрейт 292
Блок
 граница 456
 добавление тени 476
 объявлений 316
 ограничительный 642
 отступ 451
 поле 468
 стиль границы 457
 установка размеров 447
 цвет границы 460
 ширина границы (толщина) 458, 460
элементов
 внешний край 446
 внутренний край 445
 область контента 445
 отступ 446
Блокнот 81
Блочная модель 328, 445
Блочное свойство padding 453
Бот 179
Брандмауэр 46
Булево значение 725
Бэкенд 46

В

Валидатор 903
Веб-безопасный цвет 393
Веб-палитра 393
Веб-шрифт 342
Ветка 705
Видеоформат
 HLS 293
 MPEG-4 291
 WebM 292
Висячий отступ 374
Вложенный контекст просмотра 286
Внедренный медиаобъект 598
Всемирная паутина 43
Всплывающее окно 178

-
- Встроенный контент 285
Выбор
 на основе направления изображения 211
 на основе области просмотра 207
Выталкивание 708
- Г**
- Гибкая сетка 593
Гибкое изображение 593
Гипертекстовый язык разметки 53
Глиф 361
Горизонтальная линейка 111
Градиент 431
 линейный 432
 линия 432
 луч 434
 повторяющийся 435
 радиальный 434
Граница
 отделенная 666
 свернутая 667
- Д**
- Двоичная прозрачность 777
Дерево Node 746
Диаграмма
 водопада 73
 каркасная 26
 сайта 26
Дизеринг 829
Доменное имя 45
Дот-файл 687
- Е**
- Единица измерения
 em 333
 fr 557
 rem 332
 абсолютная 330
 ключевое слово 353
 относительная 331
- З**
- Заголовок HTTP 90
Закрывающий тег 88
- И**
- Идентификатор фрагмента 175
Извлечение 708
Изображение
- SVG 194
адаптивное 203, 598
без лицензионных платежей 770
векторное 184
мозаичное фоновое 414
оптимизация 823
размытие 826
разрешение 785
растровое 785
с управляемыми правами 769
фоновое 413
Индексированный цвет 775
Индексный файл 50
Инструкция 722
 if/else 727
Интернет 43
Интерфейс командной строки 683
Инtranет 46
Информационная архитектура 24
- К**
- Капитель 357, 359, 361, 362
Каскад 323
Каскадная таблица стилей 56, 311
Каталог 686
 корневой 172
 относительный корневой 172
Кернинг 363
Клиент 45
Клонирование 707
Ключевой кадр 650
Кнопка
 Reset 254
 Submit 254
Кодировка символов 89
Код состояния HTTP 59
Коллекция клипов 771
Колонтитул
 верхний 122
 нижний 123
Команда
 cd 687
 cp 688
 man 688
 mkdir 688
 mv 688
 rm 688
Командная строка 684
 приглашение 685
Комбинатор 365
Комментарий 95, 722

Контекст позиционирования 507
 Контур 461
 Кривая Безье 632

Л

Лигатура 361
 Лицензия Creative Commons 770
 Локальный корневой каталог 84

М

Макет
 Flexbox 520
 многоколоночный 521
 текущий 595
 фиксированной ширины 596
 Массив 726
 Масштаб
 @1x 820
 @2x 821
 Матовый цвет 808
 Медиазапрос 440, 593, 598
 Меню
 прокрутки 262
 раскрывающееся 261
 Метаданные 89
 Метаязык 837
 Метод
 GET 244
 POST 244
 Механизм визуализации 47
 Миксин 693
 Модель border-box 448
 Модуль
 CSS3 Transforms 638
 CSS Transitions 627
 Мультимедийный тип 599
 Навигация 126, 613

Н

Неопределенный тип данных 724
 Нормальный поток 481

О

Область
 контента 448
 просмотра 594, 860
 единица ширины 208
 рисования 857
 Оболочка 864
 Объект
 document 747
 браузера 737

Объектная модель документа 32, 92, 745
 Окно просмотра 813, 840, 860
 Оператор
 === 727
 математический 727
 сравнения 726
 Оптимизация пост-экспорта 858
 Ось
 главная 524
 поперечная 524
 Открытый исходный код 45
 Отсечение 842
 Отступ 373

П

Падающая тень 844
 Папка 686
 Паритет контента 608, 609
 Переключатель 258
 Переменная 693, 723
 область видимости 735
 Переменный шрифт 612
 Переход 628
 Пикセル 204
 аппаратный 204
 независимый от устройства 205, 788
 опорный 205, 788
 устройства 204
 физический 204
 Пиксельная плотность 787
 Плавающий элемент
 макет 490
 очистка 487
 Плагин 288
 Подвыборка 803
 Поддомен 48
 Подтверждение 704
 Позиционирование
 абсолютное 487, 505
 относительное 502, 504
 стандартное 502
 указание позиции 503
 фиксированное 502, 514
 Поисковая оптимизация 34
 Поле
 ввода пароля 251
 коллапс 470
 с отрицательным значением 472
 строчного элемента 471
 Полизаполнение 215
 Полифил 755

- Picturefill 757
- Пользовательская
блок-схема 28
система координат 861
- Пользовательский агент 45, 234
таблица стилей 324
- Пользовательское пространство 861
- Поплавок 484
- Порядковая группа 544
- Порядок наложения 513
- Правило
@supports 675
стиля 315
объявление 315
селектор 315
- Предварительный загрузчик 208
- Представление веб-страницы 31
- Презентация 311
- Препроцессор 689
- Префикс
-webkit 361
поставщика 435
- Пример
s 131
u 132
- Прогрессивное улучшение 64
- Проектирование
взаимодействия 25
интерфейса пользователя 25
ориентированное на пользователя 26
пользовательского опыта 25
- Прозрачность
двоичная 807
по альфа-каналу 806
- Протокол 43
HTTPS 244
- Профиль цвета 782
- Псевдокласс
действий пользователя 404
динамический 403
лингвистический 408
логический 408
местоположения 408
структурный 408
- Пункт 788
- Путь относительный 165
- P**
- Рабочий каталог 704
- Размер
основной 525
поперечный 525
- Разметка 88
- Разработка
бэкенда 33
фронтенда 30
- Разрешение экрана 204
- Раскадровка 27
- Растяжение
ячейки 228
столбца 228
строки 229
- Репозиторий 704
- Роль 149
- C**
- Сайт
m-дот 68
динамический 59
статический 59
- Свободно типизированный язык 716
- Свойство
flex-контейнера 523, 528, 542, 575
flex-элемента 530, 536, 538, 539
grid-контейнера 554, 558, 560, 574, 577, 579, 580
grid-элемента 568, 578
блока 446, 450, 458, 460, 462, 468, 476, перехода 632, 634, 636
плавающего изображения 496
плавающего элемента 487
позиционирования 502
текста 372–377, 378, 380, 381, 383–385, 386, трансформации 639
фонового изображения 413, 414, 416, 422, 424, 426, 428
шрифта 340, 349, 355–357, 359, 361, 364,
- Сгенерированный контент 409
- Сглаживание 803, 807
- Селектор
ID 367
атрибута 411
ближайших братьев и сестер 365
дочерний 365
значения атрибута в произвольной подстроке 412
значения атрибута начальной подстроки 411
значения атрибута последней подстроки 412
значения атрибута, разделенного дефисами 411
класса 367
контекстный 365

- потомка 365
псевдокласса 403, 407
псевдоэлемента 408
сгруппированный 330
типа элемента 315
элемента универсальный 369
- Семейство шрифтов 340
Сервер 45
Сетка: настройка 553
Сжатие
 GIF 781
 без потерь 778
 с потерями 773
- Символ
 невидимый 152
 неоднозначный 152
 специальный 153
 экранирование 151
- Система доменных имен 45
- Система контроля версий 701
 распределенная 703
- Слияние 706
- Смарт-объект 822
- Событие 737
 обработчик 738
 привязка 737
- Содержащий блок 507
- Сокет 904
- Сокращенное свойство шрифта
 font 359
- Состояние
 активное 406
 наведения 405
 фокуса 404
- Спам 179
- Список
 вложенный 115
 неупорядоченный 112
 определений 114
 упорядоченный 113
- Спрайт 673, 846
- Сылка
 внешняя 164
 горячая 188
 на фрагмент другого документа 176
 почтовая 179
 телефонная 180
 фрагмент документа 175
- Стек шрифтов 341
- Стилизация
 таблицы 665
 формы 661
- Стиль 57
 встроенный 319
 значение 315
 наследование 320
 свойство 315
- Стоп-цвет 432
- Стратегия
 авторизации 65
 написания сценариев 318
 стилизации 65
- Строка 725
- Структура документа 109
- Сценарий 57
 вложенный 719
 внешний 720
 размещение 720
- Таблица стилей
 вложенная 318
 внешняя 318, 438
 комментарий 319
- T**
- Твининг 628
- Тег 55
 открывающий 88
- Текстовая дорожка 298
- Текстовое поле
 многострочное 249
 однострочное 248
- Тип MIME 185
- Точка прерывания 603, 815
- Трансформация
 добавление наклона 643
 плавная 645
 позиции 642
 размера 642
 трехмерная 647
 углов 640
- У**
- Удаленный репозиторий 707
- Узел
 вставка элемента 753
 замена другим узлом 753
 создание нового элемента 752
 удаление 754
- Унифицированный идентификатор
 ресурса 49
- Условная загрузка 609

Ф

Фавикон 793
Фигура 118
Фиксация 703
Фильтр: примитив 844
Флажок 259
Фон
 повторяющийся 416
 положение 419
 прикрепление 424
 размер 426
Форма 239
 метка 270
 обтекания текстом 497
Формат

изображения
 APNG 779
 GIF 780
 PNG 774
 PNG-8 775
 PNG-24 778
 WebP 782

контейнера мультимедийного объекта 291
прогрессивный JPEG 774

Фразовый контент 108
Фронтенд 46
Функция 732
 возврат значения 734
 пользовательская 733
 собственная 732

Х

Художественное направление 813

Ц

Цвет
 HSL 396
 HSLa 397
 RGBa 396
 RGB 392
Цветовая
 модель
 CMYK 394
 HSL 394
 RGB 394
 палитра 777
Цветовое пространство
 RGB 773
 Truecolor 773
Центрированная обводка 858

Цикл 729
 for 730

Читата: длинная 116

Ч

Чересстрочная развертка 781, 802
Числовое значение 724

Ш

Ширина устройства 594
Шрифт
 OpenType 342
 TrueType 342
 WOFF 342
 x-высота 362
 без засечек 345
 курсивный 345
 моноширийный 345
 с засечками 345
 системный 360
 фантазийный 345

Э

Экран с высокой плотностью 813
Экстранет 46
Элемент
 a 161
 address 126
 article 124
 aside 125
 audio 297
 b 131
 blockquote 116
 body 123
 br 138
 button 255
 canvas 301, 849
 caption 226
 cite 134
 code 134
 colgroup 226
 data 137
 datalist 253
 dd 115
 defs 840
 del 138
 div 141
 dl 114
 dt 114
 em 129
 script 719
 svg 197, 840, 846
 use 846

A

Ajax 759
API 290, 903
ARIA 149

C

Canvas API 301
CSS Grid Layout 549
CSS reset 669
CSS-процессор 690

D

DNS 45
DOM 32, 92, 745
ветвь 746
узел 746
dpi 786
DTD 902

F

Flexbox 520
Flex-значение
абсолютное 542
относительное 542
Flex-контейнер 522
вложенный 523
Flex-линия 522
Flex-элемент 522
изменение порядка 542

G

Git 702
GitHub 702
Grid 562
Grid-контейнер 551
Grid-элемент 551

H

HTML 53
HTML5 shim/shiv 756
HTML-таблица 225
HTTP 44
HTTPS 49
HTTP-сервер 44

I

ID-селектор 315
IP-адрес 45

J

JavaScript 57
jQuery 759

M

Modernizr 678

N

Normalize.css 670

P

ppi 786

S

Selectivizr 757
SEO 34
SMIL 850
sRGB 393
SVG 835
адаптивный документ 860
координаты 841
сжатие 859

T

Terminal 684
TextEdit 82
TRBL 454
Truecolor 393

U

URI 49
URL 49
URL-адрес
абсолютный 162
относительный 163
URL-ссылка 47

W

W3C 32
WebVTT 299
WOFF 837
W-дескриптор 207

X

XML 837
XML-блок символьных данных 849
XSLT 851
Х-дескриптор 207
Хеш 705

```

191 <body>
192 <div id="container">
193 <header>
194 <nav>
195   <ul>
196     <li><a href="">Menu</a></li>
197     <li><a href="">News</a></li>
198     <li><a href="">About</a></li>
199     <li><a href="">Contact</a></li>
200   </ul>
201 </nav>
202 </header>
203 <h1></h1>
204 <p>The delicious baked goods you love at Black Goose Bistro...now available to go!</p>
205 </header>
206
207 <main>
208 <h2>Fresh from the Oven</h2>
209 <h3>Breads</h3>
210 <p> Our breads are made daily from highest-quality whole grain flour, water, salt, and yeast or sourdough starter. Simply and naturally, and never any preservatives. Patience is key to achieving the proper level of fermentation and baking each loaf to perfection. Available in whole grain, sourdough, olive loaf, classic rye, and potato-onion.</p>
211 <p class="more"><a href="">Learn more about our baking process...</a></p>
212
213 <h3>Muffins</h3>
214 <p> Every day, we have offer a large selection of muffins, including blueberry, multi-berry, bran, corn, lemon-poppyseed, and chocolate. Our muffins are made from scratch each day. Stop by to see our seasonal muffin flavors!</p>
215 <p class="more"><a href="">Learn more about how we make our muffins...</a></p>
216 </main>
217

```

Рис. ЦВ-1.6. Sublime Text — один из примеров специального редактора кода

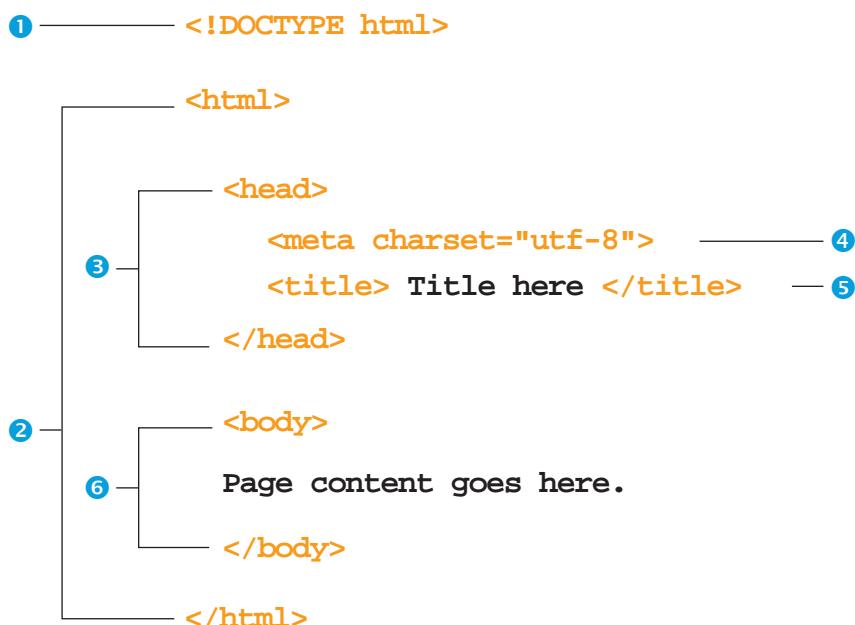


Рис. ЦВ-4.8. Минимальная структура HTML-документа включает элементы head (заголовок) и body (основная часть), входящие в область действия корневого элемента html

The screenshot shows a web browser window with the title "Black Goose Bistro". The page content is organized into several sections, each enclosed in a red rectangular box:

- Black Goose Bistro**
- The Restaurant**

The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients.
- Catering**

You have fun. *We'll handle the cooking.* Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.
- Location and Hours**

Seekonk, Massachusetts; Monday through Thursday 11am to 9pm; Friday and Saturday, 11am to midnight

Рис. ЦВ-4.11. Красные контурные линии выделяют структуру элементов начальной страницы

The screenshot shows the same web browser window after applying CSS styling. The layout has been simplified:

- A large orange circular logo featuring a black silhouette of a goose is centered at the top.
- The text "BLACK GOOSE BISTRO" is displayed below the logo in a bold, uppercase font.
- The content is organized into sections with thin black borders:

 - The Restaurant**

The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients.
 - Catering**

You have fun. *We'll handle the cooking.* Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.
 - Location and Hours**

Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm;
Friday and Saturday, 11am to midnight

Рис. ЦВ-4.16. Страница «Black Goose Bistro» после применения стилевых правил CSS

(X)HTML5 validation results for ex4-1 index.html

Validator Input:

Choose File _no file selected
 Show Image Report
 Show Source

Group Messages

1. Error: The character encoding was not declared. Proceeding using windows-1252.
2. Error: Non-ascii characters found without seeing a doctype first. Expected <!DOCTYPE html>.
From line 1, column 1 to line 11, column 75
Black Goose Bistro—The Restaurant—"The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients."
—Catering—"You have fun. We'll handle the cooking. Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers."
—Location and Hours—"Beekok, Massachusetts"—Monday through Thursday 11am to 8pm; Friday and Saturday, 11am to midnight"
3. Error: Element `head` is missing a required instance of child element `title`.
From line 1, column 1 to line 11, column 75
Black Goose Bistro—The Restaurant—"The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients."
—Catering—"You have fun. We'll handle the cooking. Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers."
—Location and Hours—"Beekok, Massachusetts"—Monday through Thursday 11am to 8pm; Friday and Saturday, 11am to midnight")

Content model for element `head`:

If the document is an `iframe` or a `document`, or if title information is available from a higher-level protocol: Zero or more elements of [metadata content](#), of which no more than one is a `title` element and no more than one is a `base` element.

Otherwise: One or more elements of `metadata content`, of which exactly one is a `title` element and no more than one is a `base` element.

A `head` element's `start tag` can be omitted if the element is empty, or if the first thing inside the `head` element is an element.

A `head` element's `end tag` can be omitted if the `head` element is not immediately followed by a `space character` or a `comment`.

There were errors. (Tried in the text/html mode.)

The Content-Type was text/html. Used the HTML parser.

Total execution time 2 milliseconds.

About this Service • More options

Рис. ЦВ-4.19. Валидатор (X)HTML5 Validator, предназначенный для выявления ошибок в HTML-документах (html5.validator.info)

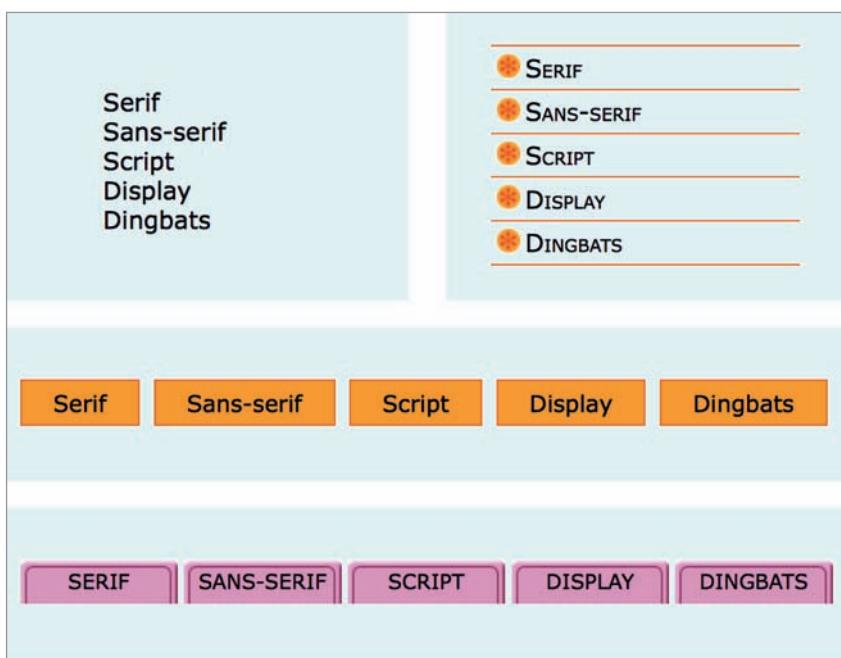


Рис. ЦВ-5.4. С помощью таблиц стилей можно придать одному и тому же неупорядоченному списку различный вид: показаны четыре варианта его оформления

Element	Description	Categories	Parent(s)	List of elements		Attributes	Interface
				Children			
a	Hyperlink	flow; phrasing*; interactive	phrasing	transparent*		globals; href; target; rel; media; hreflang; type	HTMLAnchorElement
abbr	Abbreviation	flow; phrasing	phrasing	phrasing	globals		HTMLElement
address	Contact information for a page or section	flow; formatBlock	flow	flow*	globals		HTMLElement
area	Hyperlink or dead area on an image map	flow; phrasing	phrasing*	empty		globals; alt; coords; shape; href; target; rel; media; hreflang; type	HTMLAreaElement
article	Self-contained syndicatable or reusable composition	flow; sectioning; formatBlock	flow	flow	globals		HTMLElement
aside	Sidebar for tangentially related content	flow; sectioning; formatBlock	flow	flow	globals		HTMLElement
audio	Audio player	flow; phrasing; embedded; interactive	phrasing	source*; transparent*	globals; src; preload; autoplay; mediagroup; loop; controls		HTMLAudioElement
b	Keywords	flow; phrasing	phrasing	phrasing	globals		HTMLElement
base	Base URL and default target browsing context for hyperlinks and forms	metadata	base	empty	globals; href; target		HTMLBaseElement
bdi	Text directionality isolation	flow; phrasing	phrasing	phrasing	globals		HTMLElement
bdo	Text directionality formatting	flow; phrasing	phrasing	phrasing	globals		HTMLElement
blockquote	A section quoted from another source	flow; sectioning root; formatBlock	flow	flow	globals; cite		HTMLQuoteElement

w3c.org

PM	7:30	8:00	8:30	9:00	9:30	10:00	10:30
ABC	<i>The Adventures of Ozzie and Harriet</i>	The Patty Duke Show	Gidget	The Big Valley		Amos Burke — Secret Agent*	
CBS	Lost in Space		The Beverly Hillbillies #8 25.9 rating	Green Acres #11 24.6 rating	The Dick Van Dyke Show #16 23.6 rating	The Danny Kaye Show	
NBC	The Virginian #25 22.0 rating			Bob Hope Presents the Chrysler Theatre / Chrysler Presents a Bob Hope Special		I Spy	

wikipedia.org

PROVIDENCE/STOUGHTON LINE INBOUND : Weekday Effective 11/14/11																				
Train Number	800 AM	802 AM	902 AM	804 AM	904 AM	806 AM	832 AM	808 AM	906 AM	810 AM	908 AM	B12 AM	B34 AM	910 AM	B14 AM	912 AM	B16 PM	818 PM	914 PM	916 PM
TF Green Airport	05:05			06:13		06:52		07:15						09:23		11:45				
Providence	05:07	05:25		06:07		06:33		07:12		07:35		08:10		09:43		12:05	01:30			
South Attleboro	05:17	05:35		06:16		06:42		07:22		07:45		08:20		09:52		12:15	01:42			
Attleboro	05:27	05:45		06:28		06:52		07:32		07:55		08:30	09:00	10:02		12:25	01:51			
Mansfield	05:36	05:55		06:38		07:04	07:26	07:44		08:05		08:38	09:09	10:10		12:33	01:58			
Sharon	05:44	06:04		06:48		07:13	07:35		08:14		08:47	09:17	10:19		12:42	02:06				
Stoughton			06:28		06:56			07:48		08:28			09:40		10:40		02:20	03:23		
Canton Center			06:36		07:04			07:57		08:38			09:49		10:49		02:27	0		
Canton Junction	05:51	06:11	06:39		07:08		07:41		08:01	08:24	08:40	08:54	09:24	09:52	10:26	10:52	12:50	02:30	03:33	0
Route 128	05:56	06:16	06:44	06:58	07:14	07:24	07:47		08:07	08:30	08:45	08:59	09:26	09:57	10:31	10:57	12:55	02:16	03:38	
Hyde Park	06:01	06:21	06:49		07:19		07:52		08:13	08:36	08:49	09:04		10:02	10:36	11:02	01:00	02:39	03:43	0

mbta.org

Рис. ЦВ-8.1. Примеры таблиц, предназначенных для представления информации, сведенной в строки и столбцы, — например, собственно таблиц, графиков, календарей и расписаний

таблица

строка	ячейка заголовка Menu item	ячейка заголовка Calories	ячейка заголовка Fat
строка	данные ячейки Chicken noodle soup	данные ячейки 120	данные ячейки 2
строка	данные ячейки Caesar salad	данные ячейки 400	данные ячейки 26

Рис. ЦВ-8.2. Таблица состоит из строк, содержащих ячейки, являющиеся контейнерами для контента

<table>

```
<tr> <th>Menu item</th> <th>Calories</th> <th>Fat</th> </tr>
<tr> <td>Chicken noodle soup</td> <td>120</td> <td>2</td> </tr>
<tr> <td>Caesar salad</td> <td>400</td> <td>26</td> </tr>
</table>
```

Рис. ЦВ-8.3. Элементы, формирующие базовую структуру таблицы

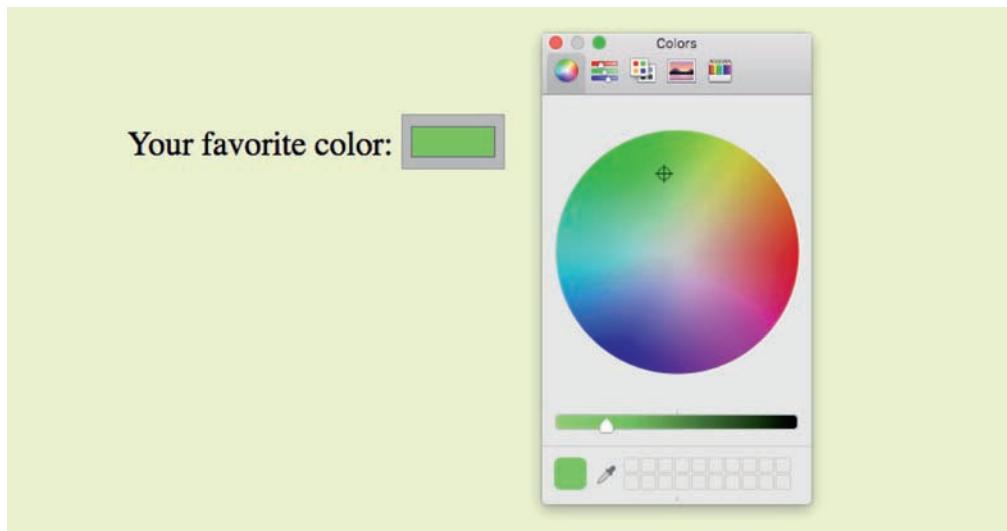


Рис. ЦВ-9.18. Элемент управления color (показан в Chrome на macOS)

Cooking with Nada Surf

I had the pleasure of spending a crisp, Spring day in Portsmouth, NH cooking and chatting with Daniel Lorca of the band Nada Surf as he prepared a gourmet, sit-down dinner for 28 pals.

When I first invited Nada Surf to be on the show, I was told that Daniel Lorca was the guy I wanted to talk to. Then Daniel emailed his response: "I'm way into it, but I don't want to talk about it, I wanna do it." After years of only having access to touring bands between their sound check and set, I've been doing a lot of talking about cooking with rockstars. To actually cook with a band was a dream come true.

Six-hour Salad

Daniel prepared a salad of arugula, smoked tomatoes, tomato jam, and grilled avocado (it's as good as it sounds!). I jokingly called it "6-hour Salad" because that's how long he worked on it. The fresh tomatoes were slowly smoked over woodchips in the grill, and when they were softened, Daniel separated out the seeds which he reduced into a smoky jam. The tomatoes were cut into strips to put on the salads. As the salad was assembled, Daniel checked on the grill after dark. I was on flashlight duty while Daniel checked for the perfect grill marks.

I wrote up a streamlined adaptation of his recipe that requires much less time and serves 6 people instead of fiftetimes that amount.

The Main Course

In addition to the smoky grilled salad, Daniel served tarragon cornish hens with a cognac cream sauce loaded with chanterelles and grapes, and wild rice with grilled ramps (wild garlicy leeks). Dinner was served close to midnight, but it was a party so nobody cared.

We left that night (technically, early the next morning) with full bellies, new cooking tips, and nearly 5 hours of footage. I'm considering renaming the show "Cooking with Nada Surf".

Рис. ЦВ-11.4. Представление документа в браузере после добавления небольшой таблицы стилей. Не только красивее — просто совершенно другой вид!

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

Hours: Monday through Thursday; 11 to 9, Friday and Saturday; 11 to midnight

Appetizers

This season, we explore

Black bean purses
Spicy black bean
Southwestern napoleons
Layers of light

Main courses

Big, bold flavors are

Jerk rotisserie chicken
Tender chicken
plantains and v
Shrimp sate kebabs v
Skewers of shrimp
peanut sauce a
Grilled skirt steak w
Flavorful skirt
mushrooms wi

* We are required to

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

HOURS: MONDAY THROUGH THURSDAY; 11 TO 9, FRIDAY AND SATURDAY; 11 TO MIDNIGHT

A P P E T I Z E R S

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden.

\$3.95

Southwestern napoleons with lump crab — *new item!*

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. \$7.95

M A I N C O U R S E S

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — *new item!*

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy.** \$12.95

Shrimp sate kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. \$12.95

Grilled skirt steak with mushroom fricassee

Flavorful skirt steak marinated in Asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. \$16.95

* We are required to warn you that undercooked food is a health risk.

Рис. ЦВ-12.1. Исходное (сзади) и доработанное (впереди) меню сайта «Black Goose Bistro», над которым мы будем работать в этой главе

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

HOURS: MONDAY THROUGH THURSDAY: 11 to 9, FRIDAY AND SATURDAY; 11 to midnight

Appetizers

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. **\$3.95**

Southwestern napoleons with lump crab — *new item!*

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. **\$7.95**

Main courses

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — *new item!*

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy. \$12.95**

Shrimp sate kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. **\$12.95**

Grilled skirt steak with mushroom fricassee

Flavorful skirt steak marinated in asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. **\$16.95**

* We are required to warn you that undercooked food is a health risk.

Рис. ЦВ-12.11. Текущее состояние меню бистро

Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts

HOURS: MONDAY THROUGH THURSDAY: 11 to 9, FRIDAY AND SATURDAY; 11 to midnight

A P P E T I Z E R S

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses

Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden.

\$3.95

Southwestern napoleons with lump crab — *new item!*

Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. **\$7.95**

M A I N C O U R S E S

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — *new item!*

Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy. \$12.95**

Shrimp sate kebabs with peanut sauce

Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. **\$12.95**

Grilled skirt steak with mushroom fricassee

Flavorful skirt steak marinated in asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. **\$16.95**

* We are required to warn you that undercooked food is a health risk.

Рис. ЦВ-12.20. Окончательный вид меню «Black Goose Bistro»

black #000000	gray #808080	silver #C0C0C0	white #FFFFFF
maroon #800000	red #FF0000	purple #800080	fuchsia #FF00FF
green #008000	lime #00FF00	olive #808000	yellow #FFFF00
navy #000080	blue #0000FF	teal #008080	aqua #00FFFF
orange (CSS 2.1) #FFA500			

Рис. ЦВ-13.1. 17 стандартных названий цветов в CSS2.1 (обратите внимание, что «gray» пишется через «а»)

Рис. ЦВ-13.2. В CSS3 включены 140 дополнительных имен цветов (имейте в виду, что на экране они выглядят совершенно иначе)

Цветовая модель RGB

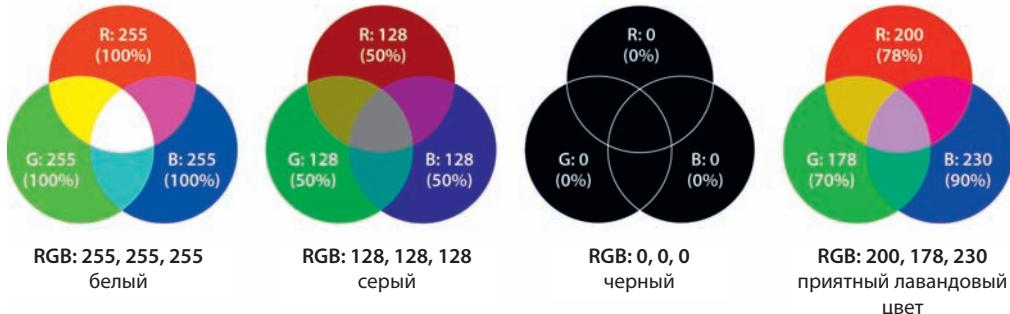


Рис. ЦВ-13.3. Компьютеры отображают цвета на мониторе, смешивая различное количество красного (Red), зеленого (Green) и синего (Blue) цветов (отсюда и RGB). Цвет в середине каждой схемы показывает, что получается при комбинации трех цветовых каналов в заданных соотношениях. Чем больше яркость в каждом канале (т. е. чем больше числовое значение цвета), тем ближе комбинация к белому цвету

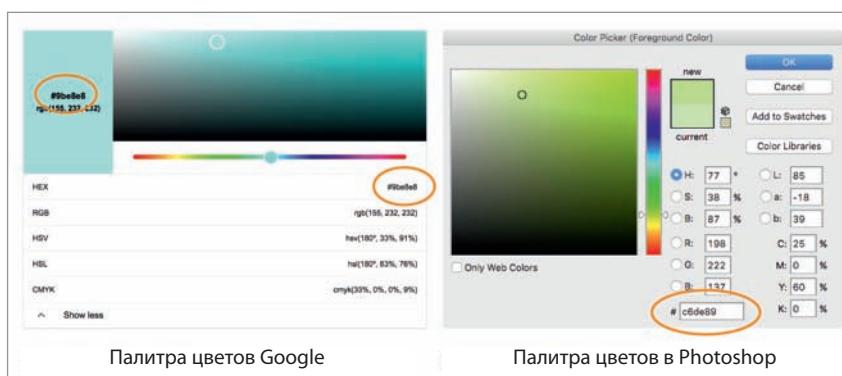


Рис. ЦВ-13.4. Палитры цветов, найденные по ключевым словам color picker: на Google.com (слева) и в Photoshop (справа)

Шестнадцатеричным значениям RGB должен предшествовать символ #.

#RRGGBB

Шестнадцатеричное значение красного цвета Шестнадцатеричное значение зеленого цвета Шестнадцатеричное значение синего цвета

Рис. ЦВ-13.5. Шестнадцатеричные RGB-значения состоят из трех двузначных чисел: по одной паре цифр для красного, зеленого и синего цветов

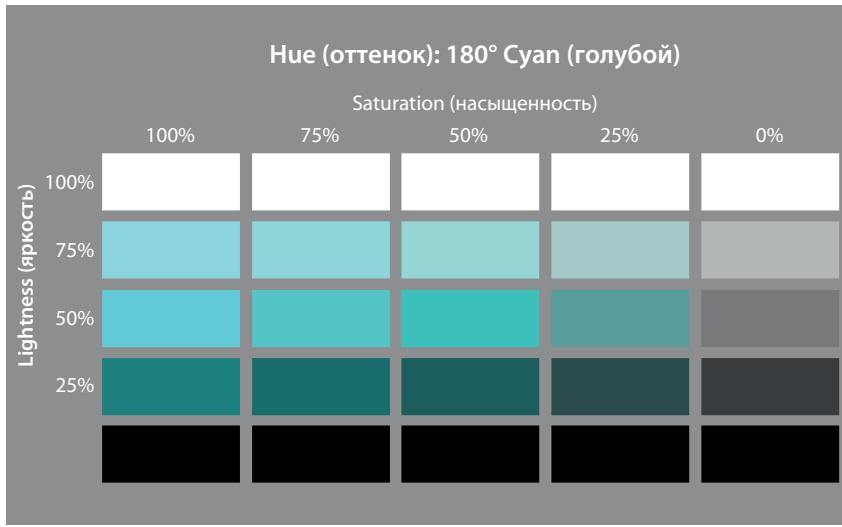


Рис. ЦВ-13.8. Один оттенок (Hue) в цветовой модели HSL с соответствующими значениями насыщенности и яркости

In the latitude of central New England, cabbages are not secure from injury from frost with less than a foot of earth thrown over the heads. In mild winters a covering of half that depth will be sufficient; but as we have no prophets to foretell our mild winters, a foot of earth is safer than six inches.

Рис. ЦВ-13.9. Применение цвета к переднему плану элемента

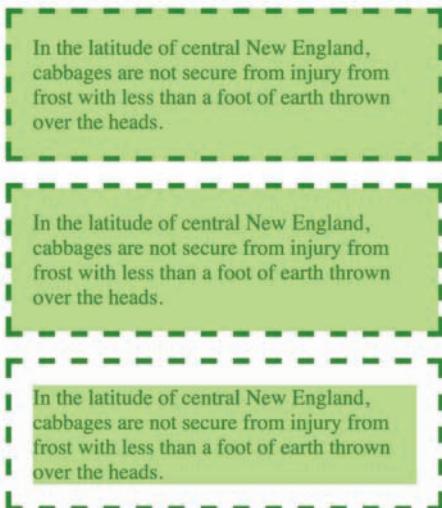
In the latitude of central New England, cabbages are not secure from injury from frost with less than a foot of earth thrown over the heads. In mild winters a covering of half that depth will be sufficient; but as we have no prophets to foretell our mild winters, a foot of earth is safer than six inches.

Рис. ЦВ-13.10. Окрашивание в зеленый цвет заднего плана элемента blockquote

Every variety of cabbage had their origin in the wild cabbage of Europe (*Brassica oleracea*)

Рис. ЦВ-13.11. Применение свойства background-color к встроенному элементу

```
blockquote {  
    padding: 1em; border: 4px dashed green; background-color: #C6DE89;  
}
```



background-clip: border-box;

background-clip: padding-box;

background-clip: content-box;

Рис. ЦВ-13.12. Свойство background-clip



opacity: .25;

opacity: .5;

opacity: 1;

Рис. ЦВ-13.13. Установка значения свойства opacity для элемента влияет как на цвет переднего плана, так и на цвет фона

| Samples of my work: |
|---|---|---|---|
| <ul style="list-style-type: none">• Pen and Ink Illustrations• Paintings• Collage | <ul style="list-style-type: none">• Pen and Ink Illustrations• Paintings• Collage | <ul style="list-style-type: none">• Pen and Ink Illustrations• Paintings• Collage | <ul style="list-style-type: none">• Pen and Ink Illustrations• Paintings• Collage |
| a:link | a:focus
a:hover | a:active | a:visited |
| Ссылки окрашены в бордовый цвет и не подчеркнуты. | Если указатель мыши находится над ссылкой, или ссылка имеет фокус, отображается фон, окрашенный в розовый цвет. | При щелчке мышью ссылка окрашивается в ярко-красный цвет. | Посещенная ссылка окрашивается в серый цвет. |

Рис. ЦВ-13.14. Изменение цвета и фона ссылок с помощью селекторов псевдоклассов



We are required to warn you that undercooked food is a health risk. **Thank you.**

Рис. ЦВ-13.16. Сгенерированный контент добавлен с помощью псевдоселекторов ::before и ::after



Рис. ЦВ-13.17. Страница меню «Black Goose Bistro» с примененными цветами

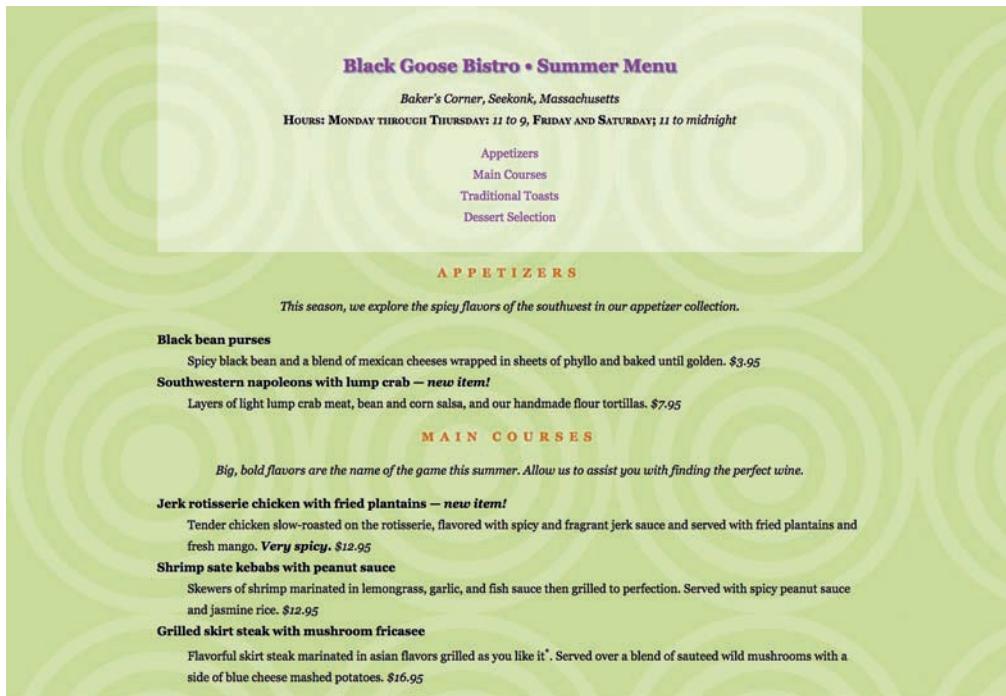
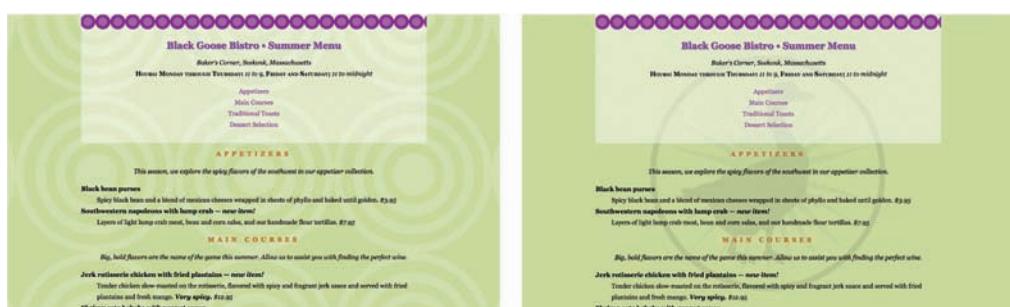


Рис. ЦВ-13.19. Меню с несложным мозаичным фоновым изображением



Рис. ЦВ-13.22. Добавление в заголовок горизонтального мозаичного изображения



Фоновая мозаика выровнена по центру

Неповторяющееся изображение позиционировано по центру

Рис. ЦВ-13.25. Результаты размещения исходного изображения: в виде мозаичного фона (слева) и единственного фонового логотипа (справа)

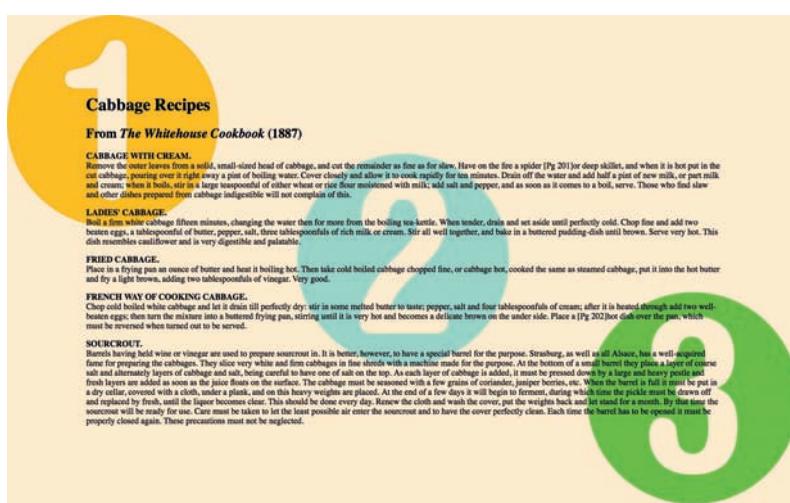


Рис. ЦВ-13.29. Добавление в элемент body трех отдельных фоновых изображений

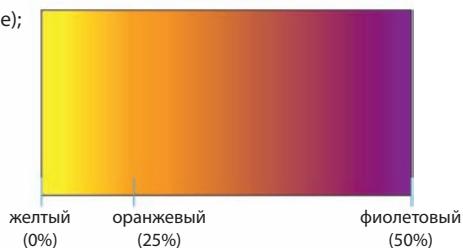


Рис. ЦВ-13.30. Заголовок меню бистро с двумя рядами фиолетовых кружков и небольшим графическим изображением гуся в элементе header

```
linear-gradient(180deg, aqua, green);
или
linear-gradient(to bottom, aqua, green);
```



```
linear-gradient(90deg, yellow, orange 25%, purple);
```



```
linear-gradient(54deg, red, orange, yellow,
green, blue 50%);
```



Рис. ЦВ-13.31. Примеры линейных градиентов

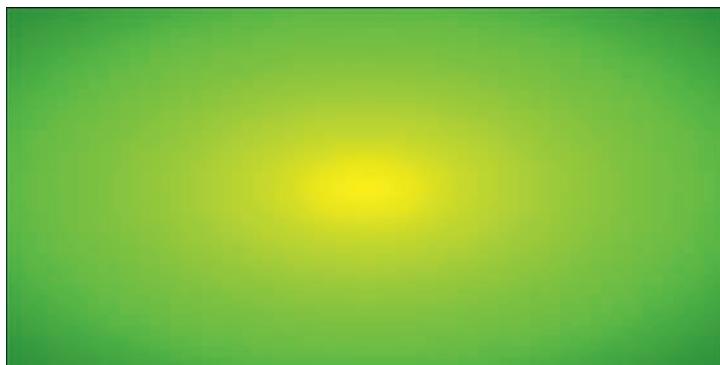


Рис. ЦВ-13.33. Минимальный радиальный градиент с заданными по умолчанию размером и расположением

```
radial-gradient(circle, yellow, green);
```



```
radial-gradient(200px 80px, aqua, green);
```



```
radial-gradient(farthest-side at right bottom, yellow, orange 50%, purple);
```



Рис. ЦВ-13.34. Примеры радиальных градиентов

repeating-linear-gradient(to bottom, white, silver 30px);



repeating-linear-gradient(45deg, orange, orange 12px, white 12px, white 24px);

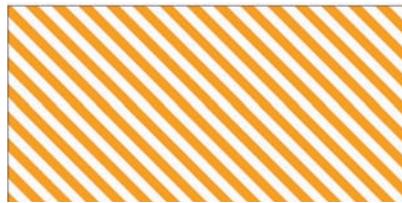


Рис. ЦВ-13.35. Примеры повторяющихся градиентов

The screenshot shows the 'Ultimate CSS Gradient Generator' interface. On the left, there's a 'Presets' section with a grid of color swatches. Below it, a 'Name' input field contains 'Blue Gloss Default' and a 'save' button. Underneath are sections for 'hue/saturation', 'reverse', 'import css', and 'import image'. On the right, there's a 'Preview' section showing a blue gradient bar, and a 'CSS' section containing the following code:

```
/* Permalink - use to edit and share this gradient:
http://colorzilla.com/gradient-editor/#007799+0,2999d+50,207cfa+51,7db9e#100;Blue+*/
background: #007799; /* Old browsers */
background: -moz-linear-gradient(top, #007799 0%, #2999d 50%, #207cfa 51%, #7db9e#100); /* FF3.6-15 */
background: -webkit-linear-gradient(top, #007799 0%, #2999d 50%, #207cfa 51%, #7db9e#100%); /* Chrome10-25, Safari1.4+ */
background: linear-gradient(to bottom, #007799 0%, #2999d 50%, #207cfa 51%, #7db9e#100%); /* W3C,
IE10+, Chrome11+, Opera12+, Safari1.4+ */
filter: progid:DXImageTransform.Microsoft.gradient
startColorstr="#007799",
endColorstr="#7db9e#100",gradientType=0 ); /* IE6-9 */
```

Below the CSS code, there are 'Color format' options (hex, RGB, CMYK) and a 'Comments' link. At the bottom, there are social sharing links for Twitter, Facebook, and Google+.

Рис. ЦВ-13.36. Инструмент Ultimate CSS Gradient Generator (www.colorzilla.com/gradienteditor) превращает создание CSS-градиентов в легкое занятие

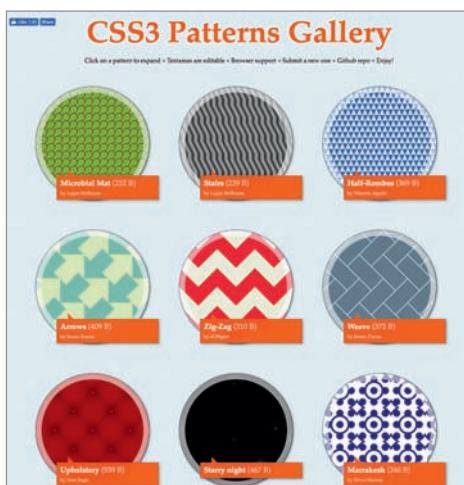


Рис. ЦВ-13.37. Галерея шаблонов CSS3, собранная Леа Веру (lea.verou.me/css3patterns). Дополнительные сведения по этой теме приведены в книге «CSS Secrets: Better Solutions to Everyday Web Design Problems» (O'Reilly)

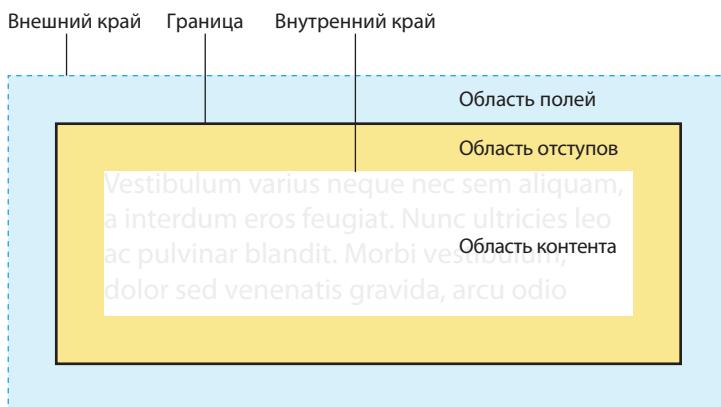


Рис. ЦВ-14.1. Компоненты блока элементов в соответствии с блочной моделью CSS

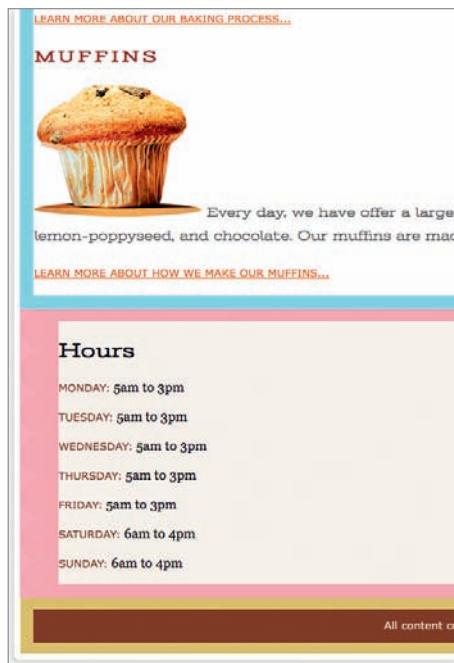


Рис. ЦВ-14.7. Подсвеченные области обозначают отступы, добавленные к main (синий цвет), aside (розовый цвет) и footer (желтый цвет). Подсветка здесь добавлена исключительно в демонстрационных целях и в браузере не отображается

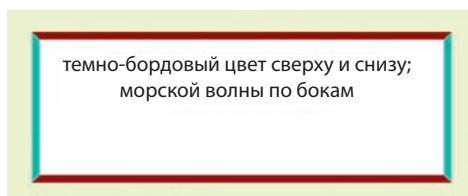


Рис. ЦВ-14.11. Указание цвета границ

p.top { margin-bottom: -3em; }

Перемещает следующий абзац вниз на 3 em.

In four or five days, if the weather is propitious, the young plants will begin to break ground, presenting at the surface two leaves, which together make nearly a square, like the first leaves of turnips or radishes.

In four or five days, if the weather is propitious, the young plants will begin to break ground, presenting at the surface two leaves, which together make nearly a square, like the first leaves of turnips or radishes.

p.top { margin-bottom: -3em; }

Следующий элемент перемещается на 3 em вверх.

In four or five days, if the weather is propitious, the young plants will begin to break ground, presenting at the surface two leaves, which together make nearly a square, like the first leaves of turnips or radishes.

Рис. ЦВ-14.19. Применение для полей отрицательных значений

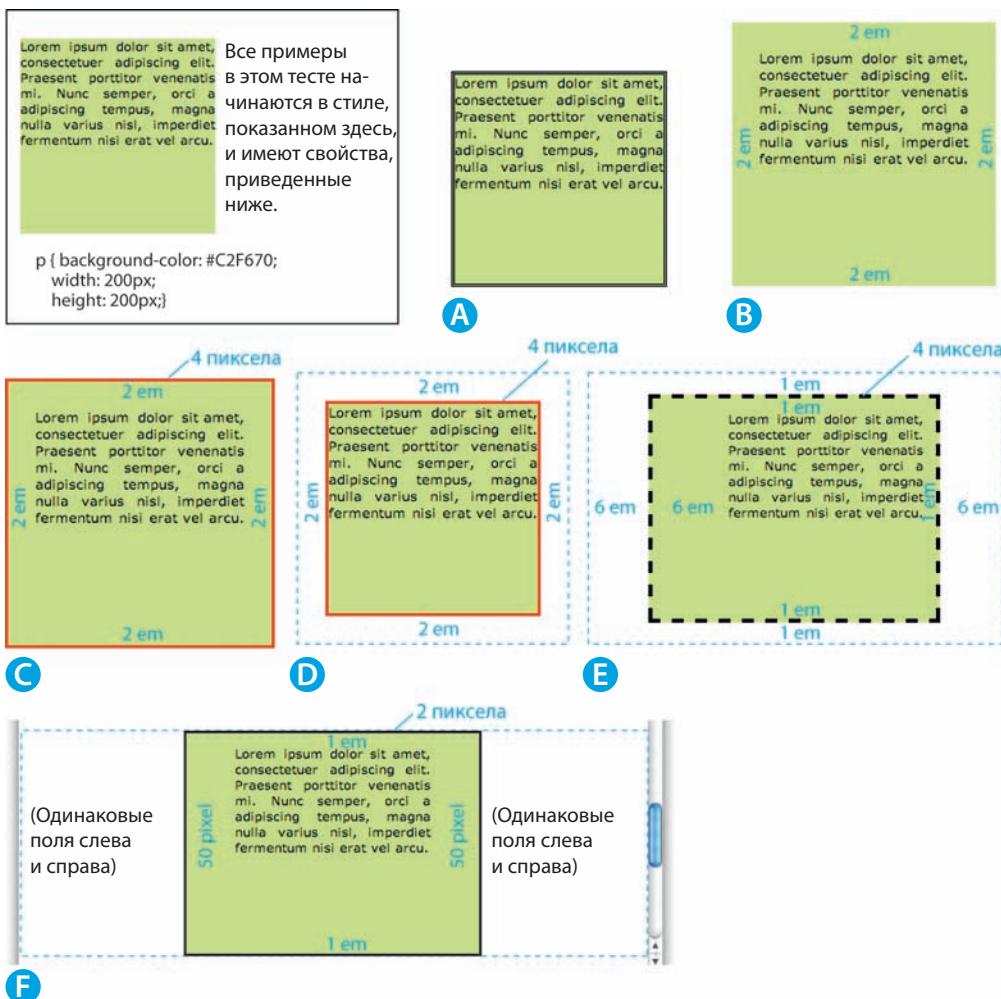


Рис. ЦВ-14.23. Напишите объявления для этих примеров

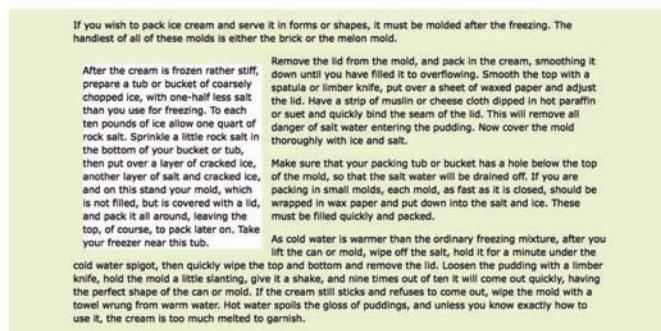
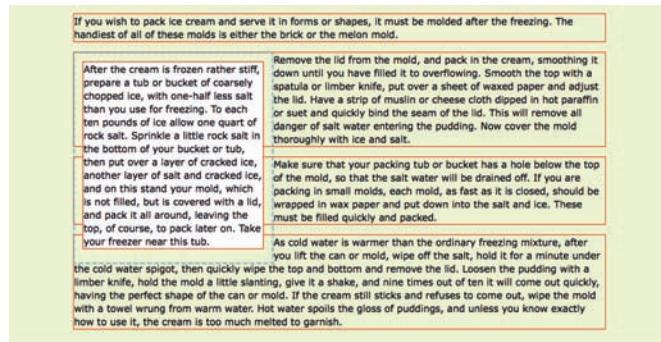


Рис. ЦВ-15.5. Плавающий элемент уровня блока



Рис. ЦВ-15.9. Элемент, содержащий плавающее изображение, не растягивается на всю его высоту, как отмечено с помощью синей границы

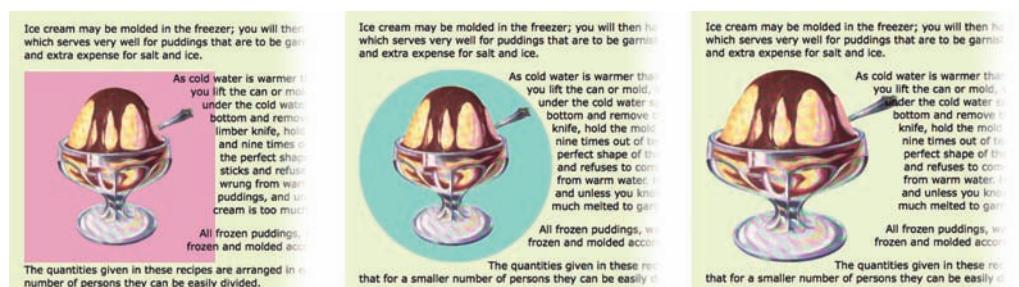
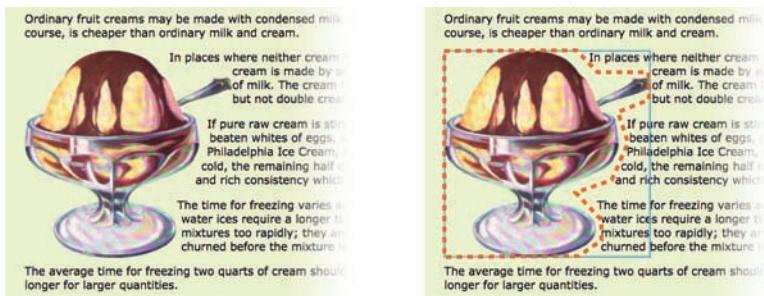


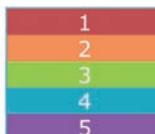
Рис. ЦВ-15.14. Применение ключевого слова circle() для создания различных вариантов формы обтекания изображений



Края изображения представлены синим квадратом, а фигура обтекания — оранжевым пунктиром

Рис. ЦВ-15.17. Собственная фигура обтекания, созданная с помощью значения polygon()

По умолчанию элементы div отображаются в виде расположенных вертикально блочных элементов. Включение режима flexbox заставляет их выстраиваться в ряд.



`display: flex;`

Блочный режим макета



Режим макета flexbox

Flex-контейнер



Рис. ЦВ-16.2. Применение режима гибкого отображения (flex display mode) превращает дочерние элементы во flex-элементы, которые выстраиваются вдоль одной оси. При этом что-либо делать с самими дочерними элементами не требуется

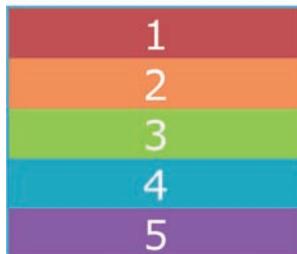
`flex-direction: row;` (default)



`flex-direction: row-reverse;`



`flex-direction: column;`



`flex-direction: column-reverse;`

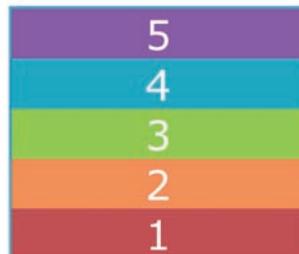
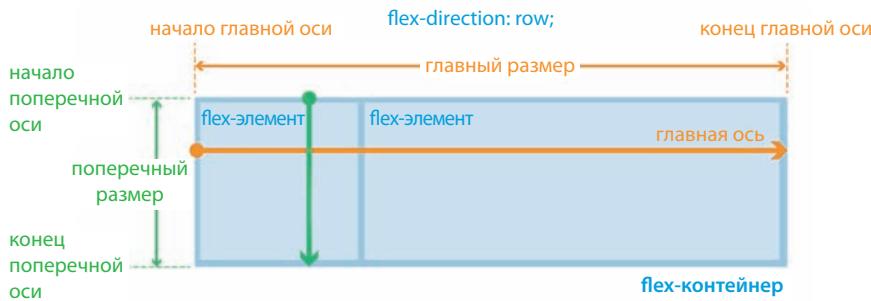


Рис. ЦВ-16.3. Примеры значений flex-direction: row, row-reverse, column и column-reverse

для языков, которые поддерживают чтение по горизонтали слева направо:

Если значение свойства `flex-direction` установлено равным `row`, главная ось — горизонтальная, а поперечная ось — вертикальная.



Если значение свойства `flex-direction` установлено равным `column`, главная ось — вертикальная, а поперечная ось — горизонтальная.

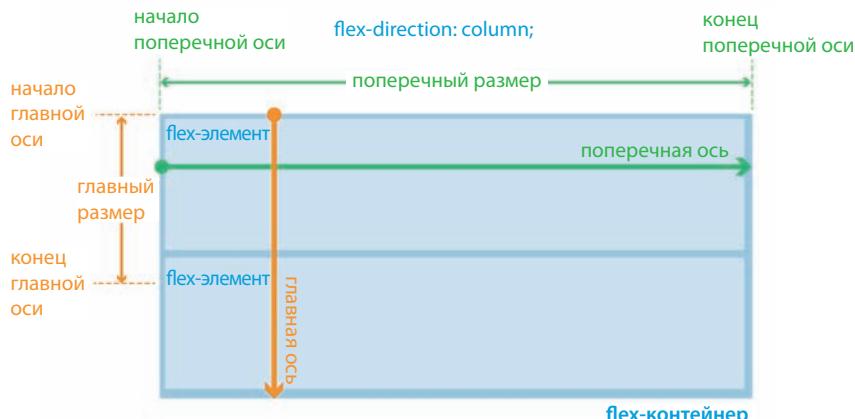


Рис. ЦВ-16.4. Компоненты flex-контейнера

`flex-wrap: wrap;`

1	2	3	4
5	6	7	8
9	10		

`flex-wrap: wrap-reverse;`

9	10		
5	6	7	8
1	2	3	4

`flex-wrap: nowrap;` (заданное по умолчанию)



Если разбиение отключено, flex-элементы при недостатке места ужимаются, а если больше не в состоянии ужиматься, то могут — при недостатке места в области просмотра — и обрезаться.

Рис. ЦВ-16.5. Сравнение эффектов применения ключевых слов `nowrap`, `wrap` и `wrap-reverse` для свойства `flex-wrap`

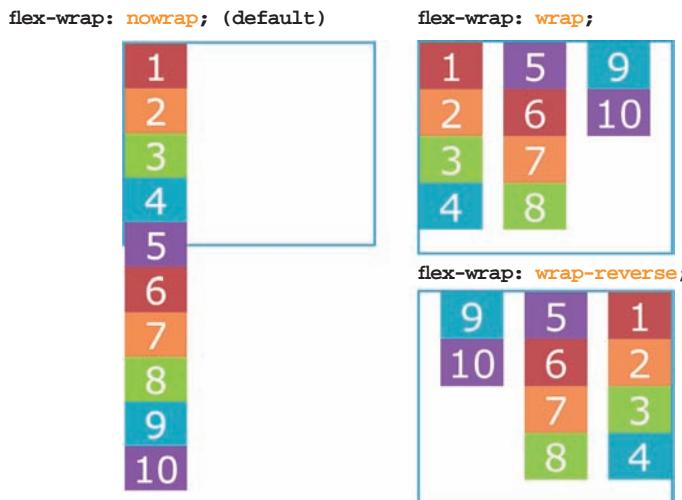


Рис. ЦВ-16.6. Сравнение эффектов применения ключевых слов nowrap, wrap и wrap-reverse, когда элементы находятся в столбце

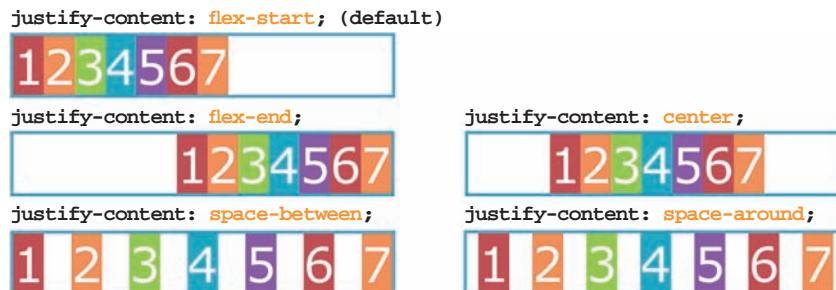


Рис. ЦВ-16.8. Значения, применяемые для выравнивания элементов вдоль главной оси с помощью свойства justify-content

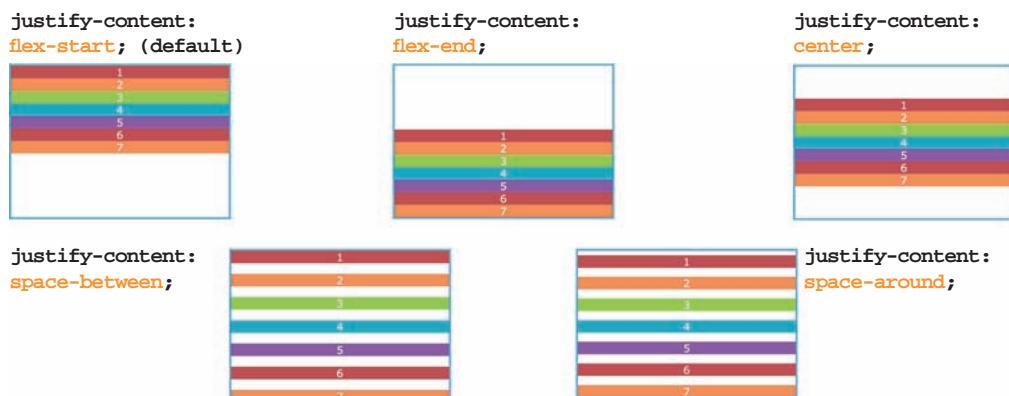


Рис. ЦВ-16.9. Значения, применяемые для выравнивания элементов вдоль вертикальной главной оси с помощью свойства justify-content (свойство flex-direction установлено в значении column)



Рис. ЦВ-16.10. Выравнивание вдоль поперечной оси с помощью свойства `align-items`



Рис. ЦВ-16.11. Используйте свойство `align-self`, чтобы переопределить для одного элемента выравнивание по оси своего контейнера

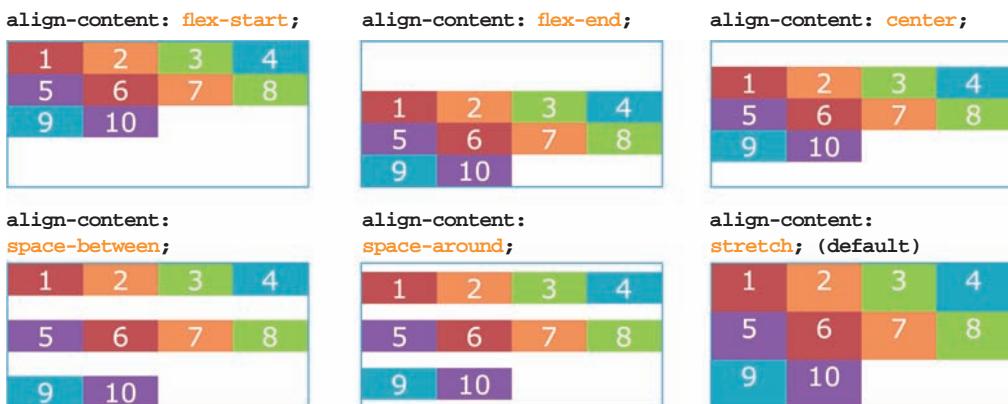


Рис. ЦВ-16.12. Свойство `align-content` распределяет пространство вокруг нескольких flex-линий. Это не приводит к какому-либо эффекту, если flex-элементы располагаются в одной строке

`flex: 0 1 auto;` (препятствует расширению)



`flex: 0 1 auto;` (позволяет расширение)



Рис. ЦВ-16.18. Если свойство `flex-grow` установить равным 1, свободное пространство строки распределяется между элементами поровну, и они растягиваются так, чтобы заполнить все пространство

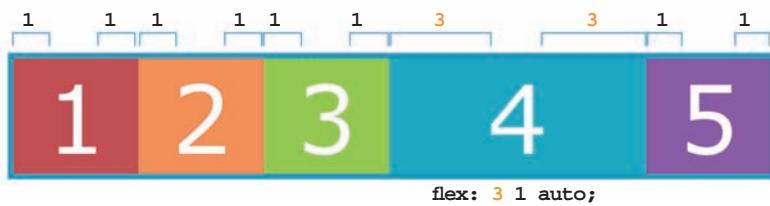


Рис. ЦВ-16.19. Присваивание определенному элементу иного значения свойства `flex-grow`: здесь элемент box4 установлен таким образом, чтобы занимаемое им место растянулось на величину, в три раза большую, чем места, занимаемые другими элементами

`flex: 3 1 100px;`



Даже если контейнер широкий, элементы не станут шире, чем их значение `flex-basis`, равное 100 пикселям, поскольку значение `flex-grow` равно 0.



Если контейнер узок, элементам разрешено сжиматься для заполнения пространства (`flex-shrink: 1`).

Рис. ЦВ-16.20. Влияние свойства `flex-basis` при заданной начальной ширине элементов

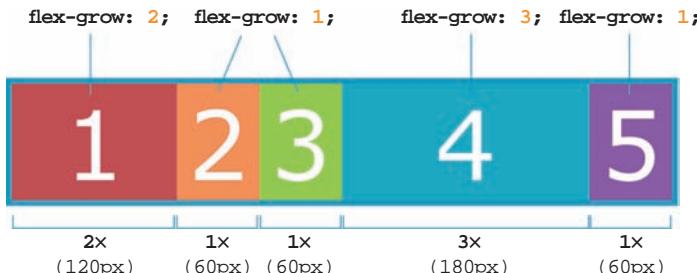


Рис. ЦВ-16.21. В случае перехода к абсолютным выражениям блоки получают размеры в соответствии с соотношениями своих flex-значений



Рис. ЦВ-16.22. Изменение порядка следования элементов с помощью свойства `order`: установка элемента `box3` в значение `order: 1` приводит к его отображению после всех других элементов



Рис. ЦВ-16.23. Установка элемента box2 в значение order: 1 приводит к тому, что этот элемент (объединенный в порядковую группу с элементом box3) отображается после элементов, имеющих заданное по умолчанию значение order, равное 0



Рис. ЦВ-16.24. Отрицательные значения свойства `order` приводят к отображению элемента перед элементами, имеющими заданное по умолчанию значение `order`, равное 0

Вид в браузере

Структура сетки выявлена
с помощью Firefox Grid Inspector

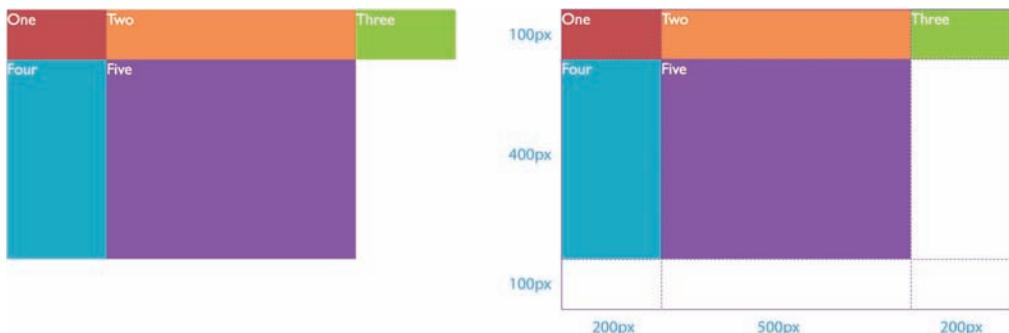


Рис. НВ-16.32. По умолчанию grid-элементы размещаются в ячейках сетки по строкам

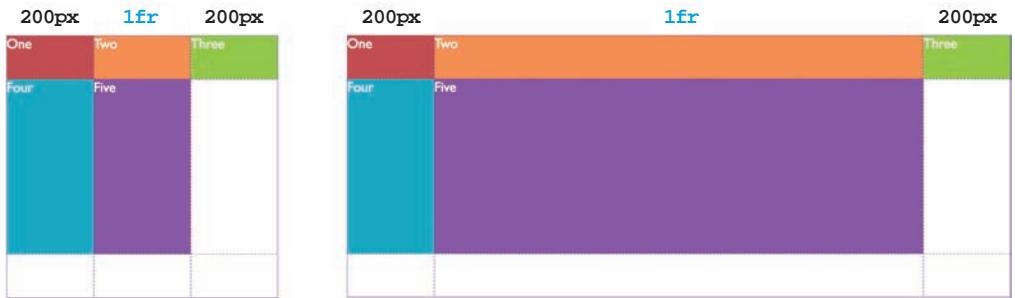
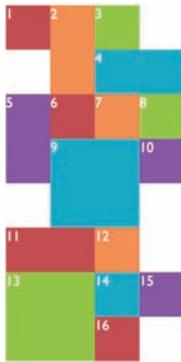


Рис. ЦВ-16.34. Если средний столбец имеет размер дорожки, равный 1fr, он займет оставшееся пространство в окне браузера и будет обеспечивать гибкость при адаптации окна браузера по ширине

Режим потока, заданный по умолчанию



Режим плотного потока с использованием ключевого слова `dense`



Рис. ЦВ-16.43. Сравнение режима, заданного по умолчанию (слева), и режима плотного потока (справа)



Рис. ЦВ-16.44. Добавление к сетке пространства между дорожками и столбцами

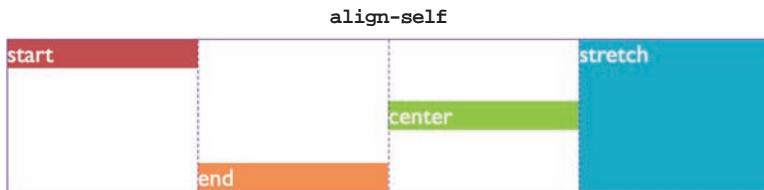
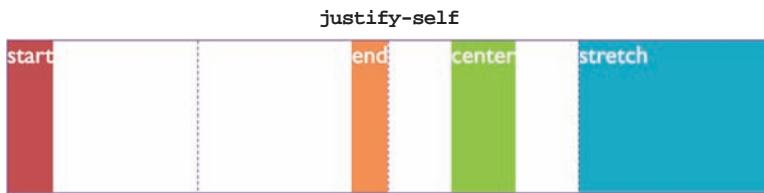


Рис. ЦВ-16.45. Значения свойств `justify-self` и `align-self`, применяемые для выравнивания grid-элементов в соответствующих им областях сетки. Эти значения аналогичным образом применяются и для свойств `justify-items` и `align-items`, обеспечивающих выравнивание всех элементов сетки

justify-content:



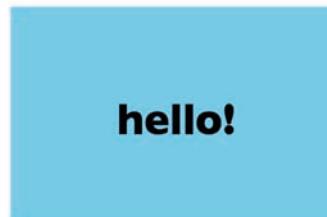
align-content:



Рис. ЦВ-16.46. Распределение свободного пространства в контейнере в зависимости от значений свойств `justify-content` (вверху) и `align-content` (внизу)



Когда окно просмотра находится в портретном режиме, цвет фона коралловый (coral).



Когда окно просмотра находится в ландшафтном режиме, цвет фона голубой (skyblue).

Рис. ЦВ-17.5. Изменение цвета фона в зависимости от ориентации области просмотра, реализуемое с помощью медиазапросов

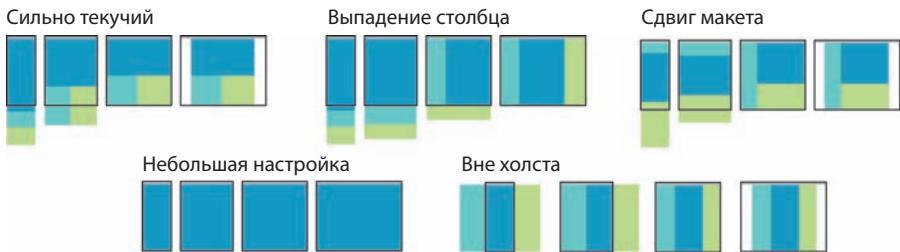
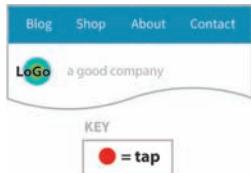
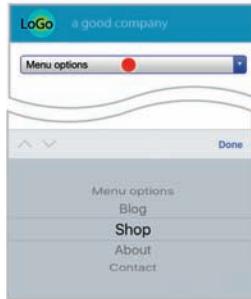


Рис. ЦВ-17.9. Примеры адаптивных макетов, описанных Люком Вроблевски (Luke Wroblewski)

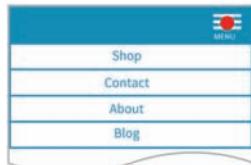
Панель навигации в верхней зоне области просмотра



Форма ввода select



Переключатель наложения (перекрывает содержимым пункта меню верхнюю часть основного контента)



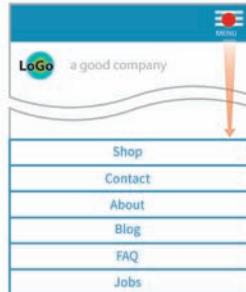
Сдвигающий переключатель (сдвигает основной контент страницы вниз)



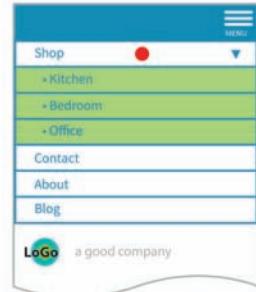
Приоритет +



Связь с меню в коллонтитуле



Аккордеонная субнавигация



Вне холста/вылетающее



Рис. ЦВ-17.11. Шаблоны адаптивной навигации (знаком ● показан выбранный пункт меню)

The left screenshot shows a landing page with a large image of bread at the top. Below it is a section titled "Fresh from the Oven" featuring a bread image. A horizontal line separates this from a "BREADS" section containing another bread image. A yellow callout box contains text about the bread's quality and ingredients. The right screenshot shows a similar layout but with a larger text block in the "BREADS" section, which has shifted to the left to accommodate the additional text.

Для заполнения свободного пространства вокруг изображения необходимо точка прерывания.

При достижении области просмотра ширины 480 пикселов изображение смещается влево.

Рис. ЦВ-17.14. Как только появляется достаточно места для обтекания изображений текстом (слева), они смещаются влево (справа)

This screenshot shows a two-column layout. The left column contains a "Fresh from the Oven" section with a bread image and a yellow callout box, followed by a "BREADS" section with a bread image and descriptive text. The right column contains a "MUFFINS" section with a muffin image and descriptive text, followed by a "Hours" section with a table of daily opening times. At the bottom, there is a footer with a copyright notice and a logo.

Рис. ЦВ-17.15. Этот среднего размера макет хорошо подходит для устройств размером с планшет

This screenshot shows a full-width layout where the content is centered horizontally. It includes a header with a logo, a "Fresh from the Oven" section, a "BREADS" section, a "MUFFINS" section, and a "Hours" section with a table of daily opening times. The layout is designed to fit well on a wide screen without significant horizontal scrolling.

Рис. ЦВ-17.16. Двухколоночный макет подходит для области просмотра, превышающей 940 пикселов. Как здесь показано, для очень широких экранов (до 1200 пикселов) контейнер перестает расширяться в ширину и содержимое его центрируется горизонтально

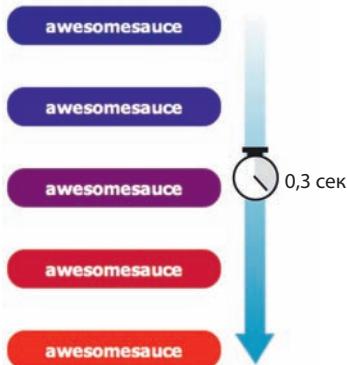


Рис. ЦВ-18.1. Цвет фона этой ссылки — если выбран классный соус организован переход — постепенно изменяется, переход в течение 0,3 секунды от синего к красному

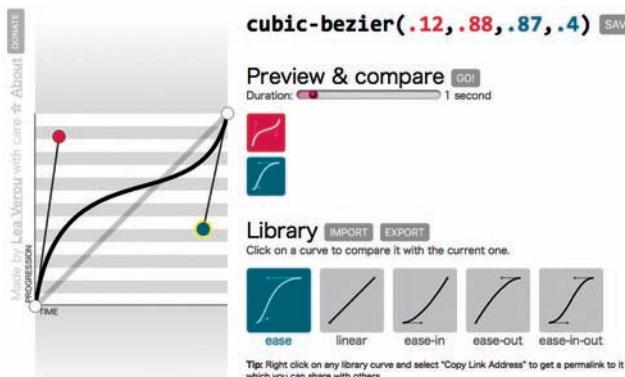


Рис. ЦВ-18.2. Примеры кривых Безье с сайта Cubic-Bezier.com. Слева показана моя кривая, которая начинается быстро, замедляется и быстро заканчивается

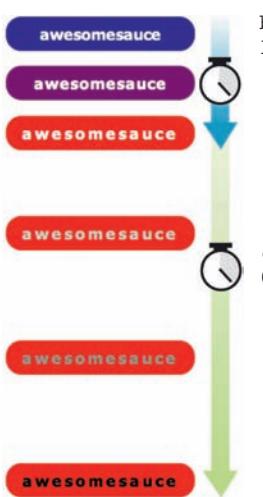


Рис. ЦВ-18.4. Цвет текста, цвет фона и расстояние между буквами изменяются с разной скоростью

Исходное (нормальное) состояние.



:hover, :focus

Изменение цвета фона и границ кнопки меню при наведении на нее указателя мыши.



:active

Кнопка меню (ссылка) нажата.



Рис. ЦВ-18.5. В этом упражнении создаются переходы между показанными здесь состояниями ссылок (кнопок меню)



Рис. ЦВ-18.18. Анимация по ключевым кадрам, выполняемая с помощью цветов радуги

```
td {  
    border: 3px solid purple;  
}  
table {  
    border-collapse: separate;  
    border-spacing: 15px 5px;  
    border: none;  
}
```



Рис. ЦВ-19.3. Модель таблицы с раздельными границами

Это видят пользователи:



Это действительно происходит:

`text-indent: -9999px;`

:jenware

Текст h1 сдвигается влево,
за пределы окна браузера.



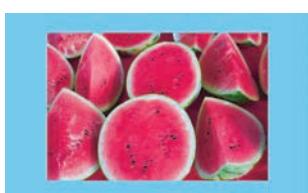
Окно браузера

Рис. ЦВ-19.7. Методика подмены текста изображением Phark скрывает HTML-текст, «выдавливая» его из поля видимого элемента с помощью большого отступа. Таким образом, отображается только фоновое изображение



Рис. ЦВ-19.8. Замена отдельных графических файлов одним изображением спрайта сокращает количество HTTP-запросов к серверу и повышает производительность сайта

Исходное изображение
(без применения эффекта)



Так это увидят пользователи
браузеров, поддерживающих режим
`mix-blend-mode: multiply;`



Запасной вариант для
не поддерживающих браузеров
`(opacity: .5)`



Рис. ЦВ-19.9. Исходное изображение (слева), результат применения свойства `mix-blend-mode` с ключевым словом `multiply` (в центре) и запасной вариант, созданный с помощью свойства `opacity` (справа)



Рис. ЦВ-23.2. Формат JPEG идеально подходит для фотографий (цветных или в оттенках серого) или любого изображения, содержащего плавные градации цвета



Исходное изображение

Максимальная степень сжатия

Рис. ЦВ-23.3. JPEG-сжатие отбрасывает некоторые детали изображения для уменьшения размера файла — при высокой степени сжатия качество изображения ухудшается, как показано на рисунке справа



Рис. ЦВ-23.4. Прогрессивные JPEG-файлы отображаются посредством вывода ряда проходов



Рис. ЦВ-23.5. Формат PNG-8 эффективен для хранения графических изображений, содержащих в основном сплошные цвета и резкие края

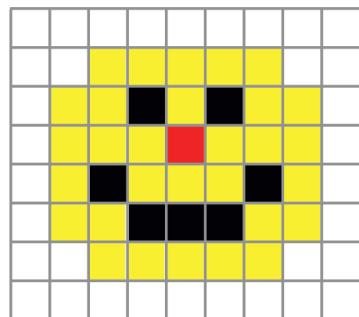
1	1	1	1	1	1	1	1	1
1	1	3	3	3	3	3	1	1
1	3	3	2	3	2	3	3	1
1	3	3	3	4	3	3	3	1
1	3	2	3	3	3	2	3	1
1	3	3	2	2	2	3	3	1
1	1	3	3	3	3	3	1	1
1	1	1	1	1	1	1	1	1

Пиксели в индексированном цветном изображении содержат числовые ссылки на таблицу цветов.

Таблица цветов сопоставляет числа со значениями RGB цвета. Это — карта для 2-битного изображения, содержащая 4 цвета.

1	2	3	4
---	---	---	---

Таблица цветов



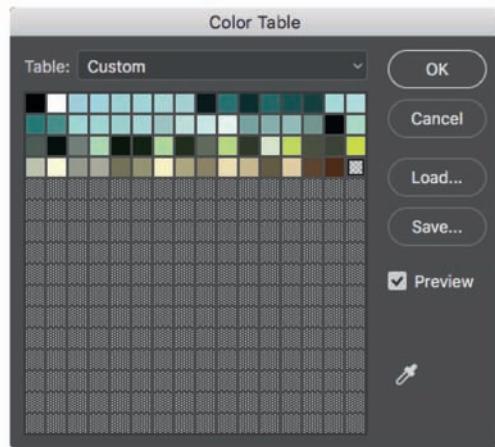
Изображение отображается с цветами, расположенными в соответствующих им местах.



Рис. ЦВ-23.6. 2-битное изображение и его таблица цветов



Изображение уменьшено до 64 индексированных цветов



Photoshop CC (2018)



GIMP

Рис. ЦВ-23.7. Таблицы цветов в Photoshop (слева) и GIMP (справа) отображают применяемые в изображении (вверху) 64-пиксельные цвета

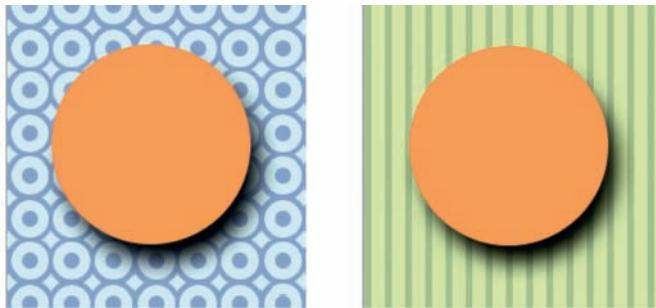


Рис. ЦВ-23.9. Альфа-канал позволяет реализовать несколько уровней прозрачности, что можно увидеть на примере отображения тени вокруг оранжевого PNG-круга

Сжатие GIF сохраняет в одном описании повторяющиеся цвета пикселов.



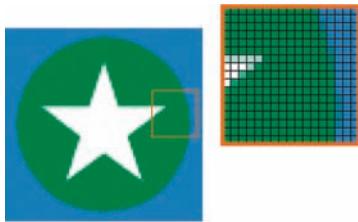
«14 синий»

В изображении с градациями цвета должна сохраняться информация о каждом пикселе в строке. Но более длинное описание приводит к большему размеру файла.



«1 синий, 1 аква, 2 светлый аква...» (и т. д.)

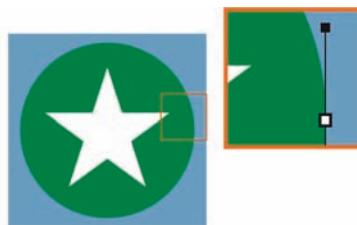
Рис. ЦВ-23.10. Упрощенная демонстрация схемы сжатия LZW. То, что на самом деле происходит при сжатии, конечно, сложнее, но этот пример дает вам возможность себе это представить



Растровые изображения представляют собой сетку (растр) разноцветных пикселов — наподобие мозаики.



Рис. ЦВ-23.11. GIF-файлы чересстрочно отображаются с помощью серий проходов, причем каждый из них более четкий по сравнению с предыдущим



Векторные изображения для построения объектов используют математические уравнения.

Рис. ЦВ-23.13. Растровая (слева) и векторная (справа) графика



Стандартные веб-изображения на 2-кратных дисплеях выглядят размытыми. Здесь изображение в формате PNG имеет ширину 350 пикселов и включено в элемент img, установленный с учетом значения ширины, также равного 350 пикселов.

Изображения на 2-кратных мониторах выглядят четкими, когда они имеют размер, который в два раза превышает размер, заданный в макете. Здесь PNG-изображение шириной 700 пикселов включено в элемент img, установленный с учетом значения ширины, равного 350 пикселов.

Рис. ЦВ-23.15. Типичная веб-графика выглядит слегка пикселизованной при отображении на 2-кратном мониторе

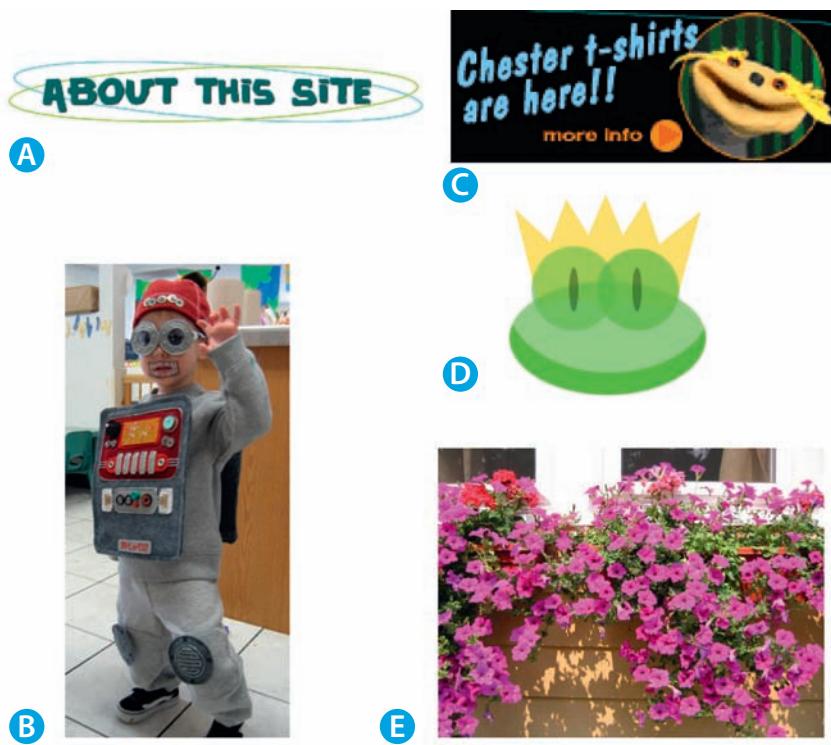


Рис. ЦВ-23.21. Выберите наилучший файловый формат для каждого изображения

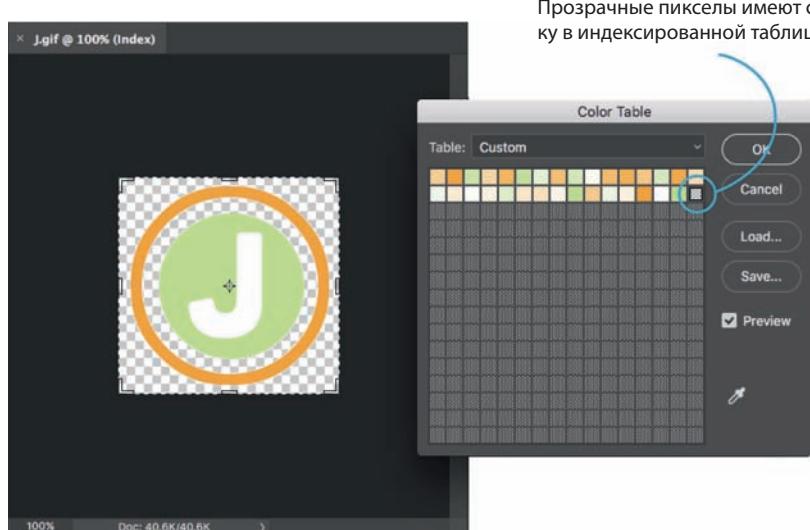


Рис. ЦВ-24.4. Прозрачность в индексированной таблице цветов рассматривается как один из цветов

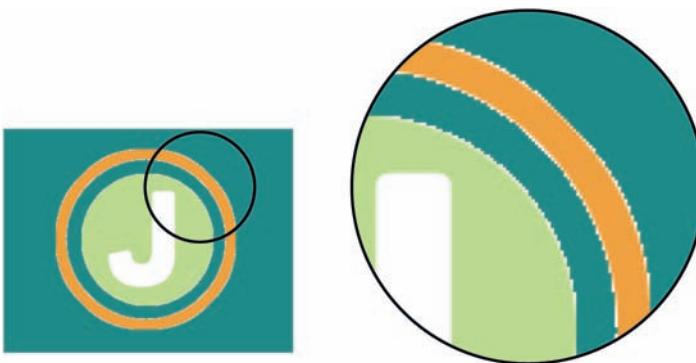


Рис. ЦВ-24.5. Это GIF-изображение с двоичной прозрачностью имеет ореол, поскольку зазубренные края исходного изображения смешиваются с светлым цветом, не совпадающим с бирюзовым цветом фона страницы

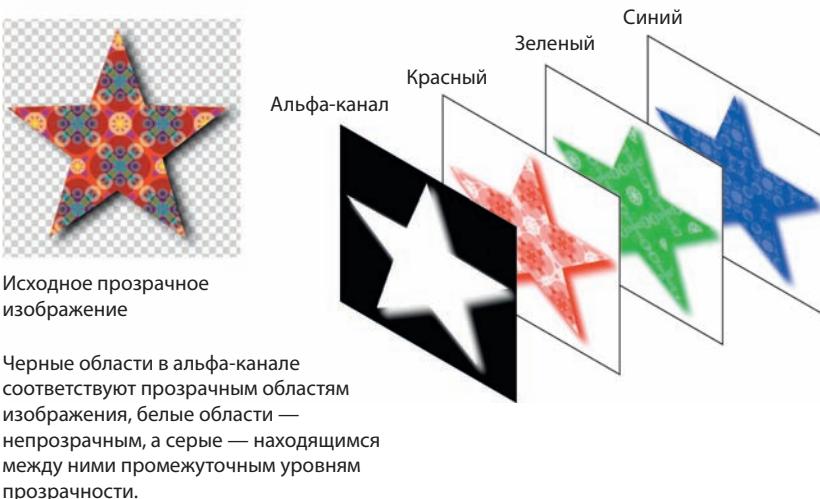
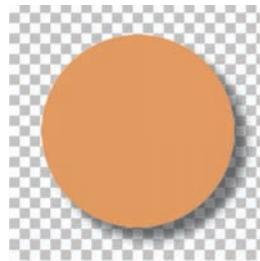


Рис. ЦВ-24.6. Информация о прозрачности сохраняется в виде отдельного (альфа) канала в 24-битных PNG-изображениях



Исходное изображение в формате PNG-24 с альфа-прозрачностью (8,4 Кбайт).



Сохранено в формате PNG-8 с 16 цветами и несколькими уровнями прозрачности (1,6 Кбайт).

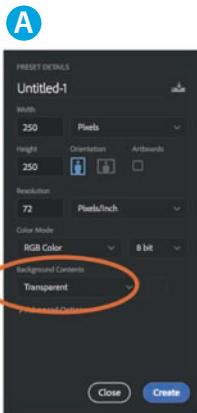


При использовании шаблона можно не заметить зубчатость краев.

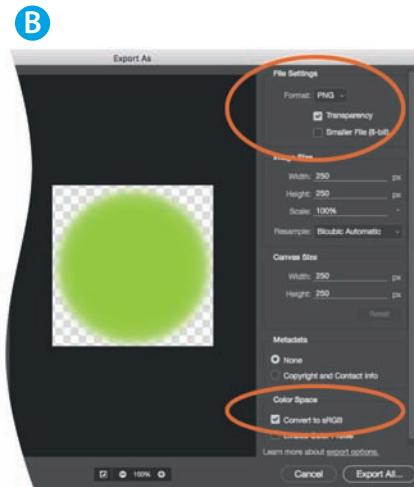


Моделирование таблицы цветов для PNG-8 с несколькими уровнями прозрачности для тени.

Рис. ЦВ-24.7. Имитация таблицы цветов PNG-8 с несколькими уровнями прозрачности. PNG-8 файл на 80% меньше аналогичного файла PNG-24 с очень близким качеством



Начните с прозрачного многослойного изображения.

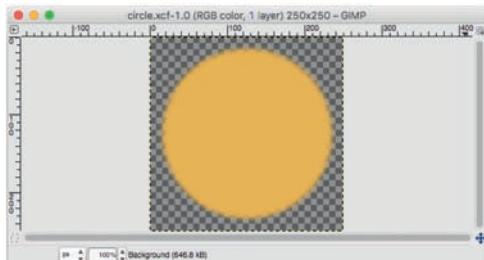
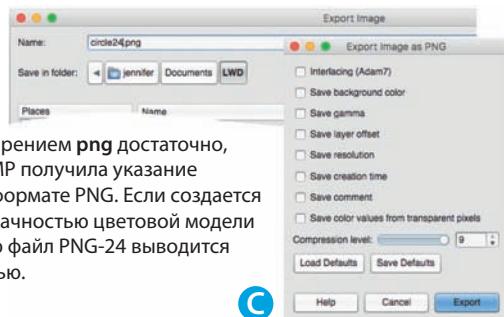


При выборе формата PNG можно остановиться на стандартном 24-битном файле (как по умолчанию) или выбрать параметр **Smaller File (8-bit)** — в любом случае сохраняются несколько уровней прозрачности.

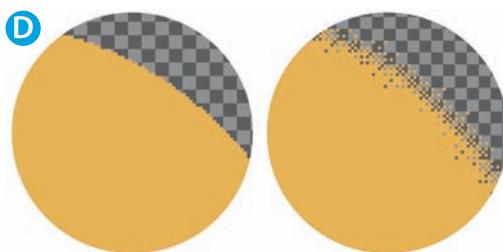


Формат GIF может сохранять только двоичную прозрачность, поэтому полупрозрачные пиксели смешиваются со сплошным белым цветом.

Рис. ЦВ-24.8. Экспорт изображения со сглаженными прозрачными краями в Photoshop CC

A**B**

Именем файла с расширением **png** достаточно, чтобы программа GIMP получила указание на сохранение его в формате PNG. Если создается изображение с прозрачностью цветовой модели RGB, то по умолчанию файл PNG-24 выводится с альфа-прозрачностью.

C

Преобразованные в индексированный цвет, мягкие края приобретают двоичную прозрачность с зазубренными краями (слева) или размытыми краями (справа). Я нахожу оба эти варианта неприемлемыми, поэтому в GIMP вам лучше выбрать вариант PNG-24.

Рис. ЦВ-24.9. Создание прозрачного изображения в программе GIMP

`background-color: white;`



`background-color: teal;`



GIF (binary)

PNG-8 (alpha)
[Photoshop CC only]

PNG-24 (alpha)

Рис. ЦВ-24.10. Разница между двоичной и альфа-прозрачностью становится очевидной при изменении цвета фона страницы



26 Кбайт

Качество: 60%, без размытия



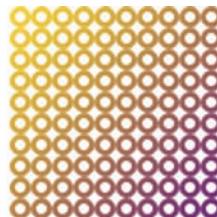
20 Кбайт

Качество: 60%, размытие по Гауссу
применяется ко всем областям, кроме лица

Рис. ЦВ-24.15. Применение размытия к менее важным частям изображения помогает уменьшить размер экспортруемого файла JPEG



gradient.jpg (12 Кбайт)



detail.jpg (49 Кбайт)

Рис. ЦВ-24.16. Сжатие JPEG лучше работает с плавными переходами цветов, чем с резкими краями и деталями



256 цветов (21 Кбайт)

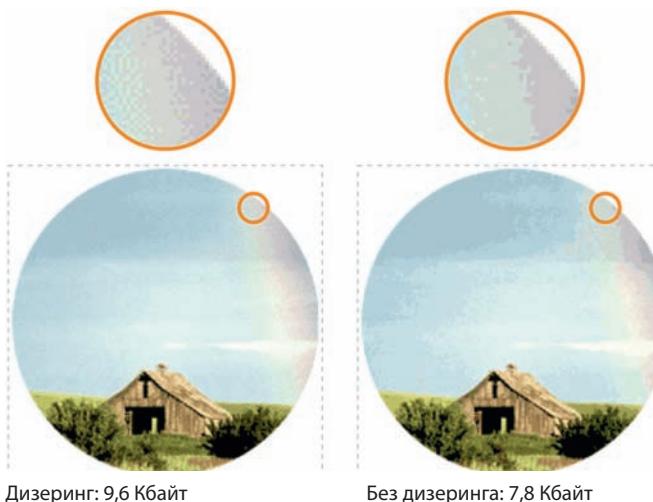


64 цвета (13 Кбайт)



8 цветов (6 Кбайт)

Рис. ЦВ-24.18. Уменьшение количества цветов в изображении уменьшает размер файла



Дизеринг: 9,6 Кбайт

Без дизеринга: 7,8 Кбайт

Рис. ЦВ-24.19. Отключение или уменьшение степени дизеринга (справа) уменьшает размер файла. Оба изображения имеют 32 пиксельных цвета и применяют адаптивную палитру



Рис. ЦВ-24.20. Для PNG-8 и GIF можно сохранять небольшие размеры файлов, заменяя градиенты и шаблоны однородными цветами

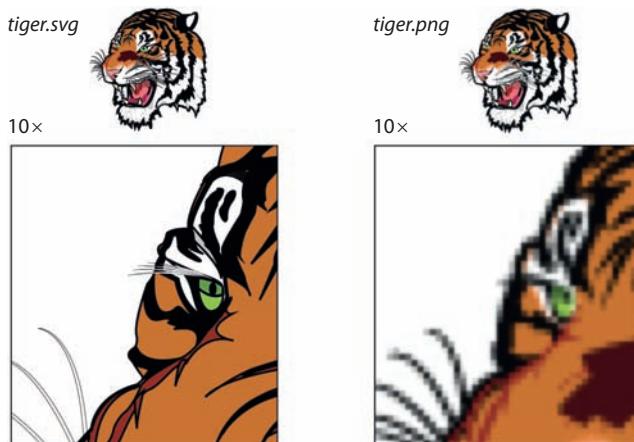


Рис. ЦВ-25.1. Векторные SVG-изображения масштабируются без потери качества

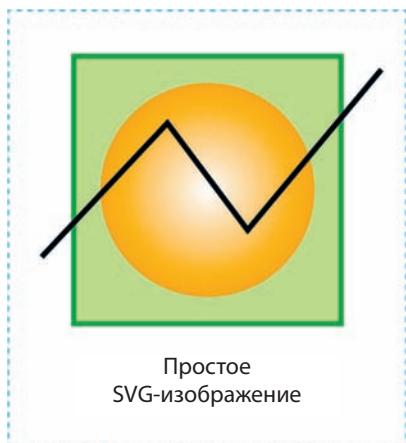


Рис. ЦВ-25.3. Базовое SVG изображение, simple.svg. Пунктирная линия добавлена для обозначения краев окна просмотра, но не входит в код SVG

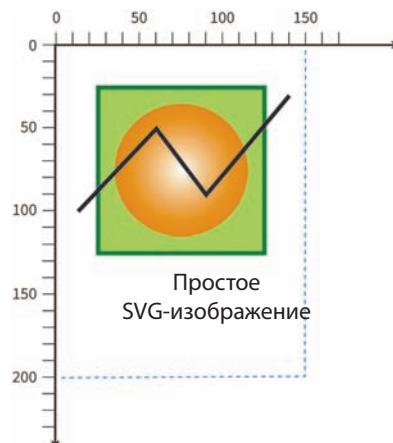


Рис. ЦВ-25.4. В SVG координаты x начинаются слева и увеличиваются вправо, а координаты y начинаются сверху и увеличиваются вниз. Исходная точка 0,0 находится в верхнем левом углу области просмотра

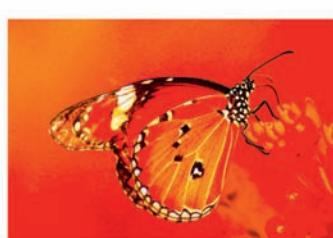


Рис. ЦВ-25.7. Примеры применения SVG-фильтров

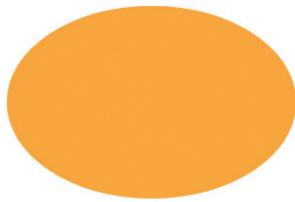


Рис. ЦВ-25.8. Фильтр размытия по Гауссу, примененный к элементу `ellipse`: вверху — эллипс без применения фильтра, внизу — с примененным фильтром

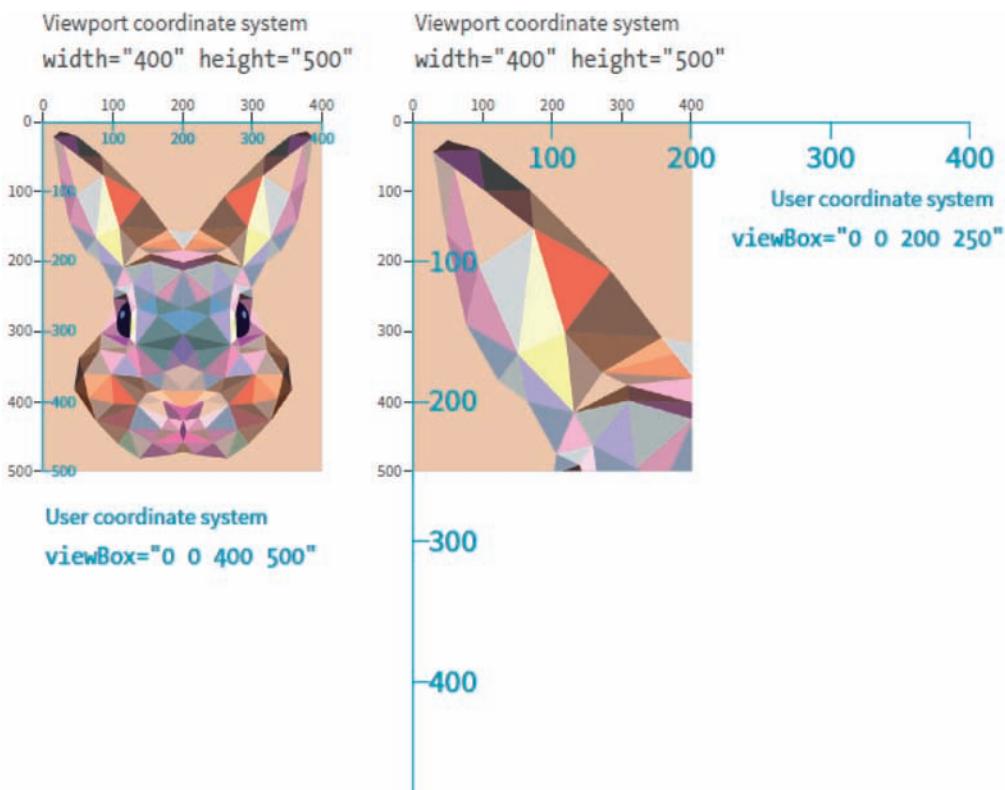


Рис. ЦВ-25.16. Область просмотра (ширина и высота) и окно просмотра