O'REILLY®

HTML5

РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

ПИТЕР®

Эстель Вейл

Estelle Weyl

Mobile HTML5

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Эстель Вейл

HTML5

РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2015

Э. Вейл

HTML5. Разработка приложений для мобильных устройств

Серия «Бестселлеры O'Reilly»

Перевели на русский *Н. Вильчинский, О. Сивченко*

Заведующий редакцией	*Д. Виницкий*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*Н. Рощина*
Художник	*Л. Адуевская*
Корректор	*Е. Павлович*
Верстка	*А. Барцевич*

ББК 32.988.02-018
УДК 004.438.5

Вейл Э.

В26 HTML5. Разработка приложений для мобильных устройств. — СПб.: Питер, 2015. — 480 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-496-01125-9

Создавайте сногсшибательные сайты и приложения для любых мобильных и настольных платформ. Для этого вам потребуется всего лишь добавить в ваш инструментарий веб-разработки технологии HTML5 и CSS3. Вооружившись этой практичной книгой, вы научитесь разрабатывать веб-приложения, которые не только хорошо работают на iOS, Android, Blackberry и Windows Phone, но и очень удобны в использовании.

Книга содержит множество примеров кода и разметки. Она поможет вам освоить работу с разнообразными инструментами HTML5 — в частности, с новыми веб-формами, масштабируемой векторной графикой (SVG), холстом (Canvas), localStorage и другими родственными API. Кроме того, в этом издании подробно рассмотрены таблицы стилей CSS3. Вы научитесь разрабатывать приложения, которые одинаково хорошо работают как на огромных мониторах, так и на крошечных экранах.

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

ISBN 978-1449311414 англ.

ISBN 978-5-496-01125-9

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 27.08.14. Формат 70×100/16. Усл. п. л. 38,700. Тираж 1000. Заказ 0000.
Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ГППО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.

Краткое содержание

Оглавление

Введение

В этой книге речь пойдет о разработке мобильных веб-приложений. Подчеркиваю — именно веб. Основное внимание в книге уделяется разработке приложений для мобильных устройств, в частности Android, iPod, iPhone, BlackBerry, различных планшетов. В этой книге не рассматривается нативная[1] разработка, для которой вам потребовался бы специальный инструментарий (SDK) для iOS или Android. Ни одна из рассматриваемых тем не является специфичной для той или иной операционной системы.

Мобильные веб-приложения — это сайты или веб-приложения, максимально эффективно использующие разметку, применяемую в настольных веб-приложениях, и комбинирующие ее с различным функционалом устройств, оснащенных возможностями сенсорного ввода. К таким веб-приложениям можно обращаться с мобильного телефона, планшета или ноутбука, но разрабатываются они в основном с помощью HTML, CSS и JavaScript. Кроме этих технологий, в таких приложениях могут применяться изображения, видео- и аудиоресурсы, а также серверные технологии.

Обновив свой арсенал и научившись использовать новейшие возможности HTML5, CSS3 и JavaScript, мы сможем создавать такие веб-приложения, которые будут восприниматься и функционировать практически как нативные аналоги. Поскольку мобильные веб-приложения основаны именно на Сети, они совместимы с ПК, смартфонами, планшетами и любыми другими устройствами, на которых работают современные браузеры. Именно в силу такой привязки к Сети мы можем напрямую распространять их среди пользователей, обходясь без посредничества Apple App Store или Google Play. Ведь процесс рецензирования нового приложения на этих рынках бывает чрезвычайно сложным, дорогостоящим и длительным.

Прочитав книгу, вы научитесь всему необходимому для создания таких приложений, которые работают в браузере на базе обычных HTML5[2], CSS3 и JavaScript. Все эти технологии вам уже знакомы, они легко портируются на большинство устройств. Все рассматриваемые темы не зависят от конкретной операционной системы.

Иными словами, весь материал, который мы изучим в этой книге, будет актуален не только для iPhone, iPad и устройств с Android, но и для других мобильных платформ, в частности Firefox OS и Windows Phone, а также современных браузеров настольных устройств и любых других устройств, где в принципе может рабо-

---

[1] Нативный — родной для данной платформы, немодифицированный, выпущенный производителем. — *Примеч. ред.*

[2] Под термином HTML5 мы также будем понимать проект, называемый HTML: The Living Standard. — *Примеч. авт.*

тать браузер. Таковы, например, игровые приставки вроде Wii. Да, основной темой данной книги является разработка для мобильных устройств, но материал этого пособия будет полезен и при разработке для множества других приборов, больших и малых. Единственное условие — на устройстве должен работать браузер, соответствующий современным веб-стандартам.

Возможности приложений, работающих на нативных платформах, довольно последовательно развивались на протяжении минувших десяти с лишним лет. Однако в последние несколько лет и на веб-платформах активно развиваются возможности работы с веб-приложениями. Причем по надежности такие приложения практически не уступают нативным.

На iPhone появились холст, кэш приложений, возможности работы с базой данных и масштабируемая векторная графика (SVG). Все эти возможности были реализованы в браузере Safari 4.0, где появились средства для работы с аудио, видео и веб-работниками (технология web workers). Позже, в 2009 году, мы наблюдали распространение средств для работы с геолокацией и холстом — не только в iPhone, но и в Chrome, Opera, Firefox, Internet Explorer и Android.

Работая с браузерами, мы уже долгие годы можем пользоваться широчайшими возможностями HTML, CSS, DOM (объектной модели документа), SVG (масштабируемой векторной графики) и XHR. В этой книге мы расширим горизонты наших знаний и дополнительно ко всем этим технологиям изучим HTML5 и CSS3. Эти знания и умения необходимы для создания веб-приложений, которые могут поспорить с нативными, а также для реализации возможностей, которые уже поддерживаются в современных браузерах для мобильных устройств и ПК.

Сравнение нативных приложений и веб-приложений

Разумеется, вы можете продавать нативные приложения для iPhone на рынке App Store. Заманчивая перспектива. Нативные приложения для Android можно продавать через Google Play, Amazon, а также по многим другим онлайновым каналам. Но при работе с веб-приложениями вы запросто обходите такие рынки вместе с их процедурами рецензирования[1], ежегодными взносами[2], отчислениями с продаж и маркетинговыми правилами. Вы контактируете непосредственно с целевой аудиторией — через сайт или с помощью других маркетинговых механизмов. Да, вы упускаете крошечный шанс, что ваша программа будет замечена среди сотен тысяч других разработок, от которых ломится рынок приложений, но достоинства

[1] На самом деле в Apple приложения подвергаются цензуре. Никаких фривольных картинок. Никаких сцен насилия. Правда, «миленькие» сцены насилия иногда проходят цензуру — так что же, может быть, продвигать их, специально нацеливаясь на детскую аудиторию? — *Примеч. авт.*

[2] Вы обязаны платить в Apple ежегодный «взнос разработчика». Это необходимое условие для отправки нативных приложений на рынок App Store — независимо от того, насколько успешно продаются ваши приложения и одобрены ли они вообще. — *Примеч. авт.*

веб-приложений по сравнению с нативными значительно перевешивают неизбежные издержки.

Работая с веб-приложениями, проще выстраивать процесс сборки и итерации. Вы можете вносить изменения в ваше «живое» веб-приложение так часто, как вздумается, — даже по нескольку раз в день, если это потребуется.

Например, процесс рецензирования приложения для iPhone обычно занимает более трех недель. Когда ваше приложение наконец одобрено и пущено в дело, приходится ждать еще какое-то время, пока пользователи синхронизируют и обновят свои приложения. В случае с веб-приложениями, использующими только CSS3 и HTML5, все изменения, вносимые в программу, вступают в силу почти немедленно, но при этом такие приложения остаются доступными и для тех пользователей, которые остаются офлайн. В этом отношении веб-приложения ничуть не уступают нативным.

Если в титрах своего нативного приложения вы случайно забудете упомянуть начальника или неправильно напишете фамилию матери, эти несуразицы будут преследовать вас до тех пор, пока вы не продавите пустячное исправление через App Store. Но даже после этого такие огрехи исчезнут не сразу — пользователям еще понадобится синхронизировать свои экземпляры приложения с новой версией, появившейся на iTunes. На все это может уйти немало времени.

ПРИМЕЧАНИЕ

Я на редкость консервативный пользователь. Так, у меня iPhone самой первой модели, и на нем я так и не обновила оригинальные версии Bump, Twitterific и Gowalla. Полагаю, не я одна люблю пользоваться такими «антикварными» приложениями для iPhone. Не думайте, что все пользователи ваших нативных приложений регулярно их обновляют.

Если вы разрабатываете веб-приложения на HTML5, ваши программы могут оставаться доступными и в офлайновом режиме — как и нативные аналоги. В то время как на обновление нативного приложения могут уйти целые недели, веб-приложение можно обновлять принудительно — новая версия загрузится при первом же подключении устройства к Интернету. Мы подробно обсудим такие обновления в главе 6, когда будем говорить об офлайновых приложениях.

При разработке веб-приложений с применением HTML5 вы опираетесь на те знания HTML и CSS, которые у вас уже есть. Мы будем работать на базе уже имеющихся навыков, ничего принципиально нового вам изучать не придется — ни новых технологий, ни других платформ. И никакого нового языка, который работал бы всего на одной платформе!

Пользуясь браузерной разметкой, состоящей из HTML и CSS, вы со временем обеспечиваете кросс-платформенность своих приложений. Нативные приложения для iPhone будут работать на iPod touch, iPhone и, скорее всего, на iPad, но никогда не будут работать на Windows, Blackberry или Android. Нативные приложения для Android будут работать только на устройствах с операционной системой Android, но не на устройствах с iOS. Нативные приложения для GoogleTV также никогда не будут работать на iOS и т. д. В отличие от нативных программ, ваши приложения, написанные на HTML5/CSS3, будут работать в мобильных браузерах WebKit,

IE10, Blink, Opera Mobile (не mini), а также Firefox. Кроме того, ваши мобильные приложения будут функционировать на любых устройствах, где есть современные браузеры, по умолчанию поддерживающие возможности HTML5 и CSS3.

Большинство веб-приложений, построенных с применением HTML5 и CSS3, уже готовы к работе в современных браузерах. HTML5 и CSS3 не поддерживаются в Internet Explorer 8 и ниже, а в Internet Explorer 9 поддерживаются частично. Была проделана огромная работа по обеспечению поддержки в Internet Explorer 10 большинства этих возможностей, описываемых в постоянно развивающихся спецификациях.

С тех пор как в 2008 году вышла первая версия инструментария для разработки под iPhone (iPhone SDK), большинство приложений, создаваемых для iPhone, являются нативными. До выхода SDK в наличии имелись только веб-приложения. Пользователи стали переходить с веб-приложений на нативные аналоги лишь по той причине, что язык HTML5 еще не был доработан. В настоящее время мобильные браузеры уже поддерживают многие API HTML5, и мы можем создавать быстрые, отзывчивые и эстетически привлекательные веб-приложения.

И наконец, видео! В iPhone, iPod и iPad технология Flash не поддерживается и никогда не будет поддерживаться. Однако на всех устройствах с iOS есть браузер Safari WebKit, поддерживающий HTML-элемент `<video>`. Об этом элементе мы поговорим в главе 5.

Напутствие (прощание со старым Internet Explorer)

В настоящее время широко распространились хорошо стандартизированные и дальновидно продуманные браузеры. Причина такого изобилия подобных программ — появление все новых мобильных гаджетов. В таких условиях дисциплина веб-разработки может окончательно вступить в текущий век и распрощаться с наследием ушедшего.

Прорабатывая материал этой книги, забудьте о существовании старых версий Internet Explorer. Интернет непрерывно развивается, причем он движется вперед семимильными шагами. Вы не решались изучать HTML5 и CSS3 лишь потому, что эти технологии не поддерживаются в IE версий 6, 7 или 8? Вы не найдете этих древних браузеров на мобильных устройствах, даже на обычных ПК популярность этих программ стремительно падает. Перестаньте вставлять себе палки в колеса!

Поскольку унаследованные и нестандартизированные браузеры по-прежнему широко распространены — особенно это касается Internet Explorer версий 6–8, — многим веб-разработчикам по-прежнему не удается создавать по-настоящему интересные сайты. Пока мы идем на поводу у IE6 и IE7, потворствуя всем их причудам, мы вынуждены работать с архаичным кодом. Поэтому, решаясь на внедрение довольно сложных, пусть и неновых, а тем более относительно новых стандартов, мы всегда делаем это не без некоторого мандража. В этой книге речь пойдет о технологиях, которыми мы сможем вооружиться именно потому, что нам не придется работать с оглядкой на какие-то допотопные браузеры.

Читая эту книгу, старайтесь освоить HTML5 и CSS3 в максимально полном объеме. Не перестраховывайтесь: «Ой, а вдруг это не сработает в браузере X?». Правильная установка: «О, какая интересная возможность!». Набивайте руку. Изучайте синтаксис. Когда поддержка всех этих новых функций наконец-то появится во всех браузерах, вы уже наработаете себе гандикап. А пока это время не наступило, вы успеете получить ряд важнейших навыков и, возможно, уже напишете замечательное веб-приложение.

Браузерный ландшафт

Все мобильные и настольные версии браузеров Safari, Chrome, Firefox, Opera и IE10 поддерживают современные веб-стандарты. в том числе HTML 4.01, XHTML и отчасти HTML5. Таблицы стилей CSS 2.1 в этих браузерах поддерживаются практически в полном объеме, а возможности CSS3 — большей частью. Также в этих браузерах хорошо поддерживается JavaScript, в том числе технологии AJAX. Не менее полно поддерживается объектная модель документа (DOM) уровня 2. Операционная система Windows включилась в эту игру с некоторым опозданием, но новые смартфоны с ней поддерживают HTML5. Признайтесь, в 2010 году у вас был хоть один знакомый, который имел смартфон с Windows Phone? Только в феврале 2012 года, когда я в очередной раз поинтересовалась в большой аудитории, есть ли у кого-то из присутствующих смартфон с Windows, один слушатель ответил: «У меня есть». В настоящее время популярность смартфонов с Windows постепенно растет. В ходе работы мы проигнорируем устаревшую операционную систему Windows Mobile, однако учтем нужды тех пользователей, которые приобретают новые смартфоны с Windows Phone.

Основная тема этой книги — проектирование и разработка сайтов для мобильных браузеров. В ней мы научимся использовать ультрасовременные веб-технологии. Ранее мы уже договорились, что не будем отвлекаться на функциональные ограничения, существующие в архаичных браузерах. Тем не менее я стремлюсь к тому, чтобы мои сайты нормально (пусть и не совершенно идентично) отображались в любых браузерах. Думаю, вы тоже ставите перед собой такую цель. При необходимости я буду делать краткие отступления и описывать нюансы, давать советы и подсказывать уловки, помогающие обработать конкретную возможность в относительно старых немобильных браузерах.

Сравнение веб-приложений и нативных приложений. Краткий исторический экскурс

Не прошло и недели с момента выхода на рынок первого iPhone (это было в июне 2007 года), как в Сан-Франциско состоялась первая конференция iPhoneDevCamp. На момент выхода первой модели iPhone еще не существовал SDK (инструментарий разработчика) для этого устройства. Поэтому все приложения, работавшие на первом iPhone, были именно веб-приложениями.

В те дни операционная система для iPhone была значительно менее мощной, чем современные мобильные системы, имеющиеся на рынке. Для передачи всех

данных использовалась сеть EDGE, поэтому любые загрузки протекали удручающе медленно. Учитывая подобные ограничения, основное внимание при разработке первых приложений для iPhone уделялось максимальному снижению объема загрузок. Одна порция данных должна была содержать менее 10 Кбайт основного контента, менее 10 Кбайт изображений и менее 10 Кбайт JavaScript.

Участники первой конференции iPhoneDevCamp подготавливали каждый свою документацию, вместе учились разрабатывать интересные приложения для iPhone (все это были веб-приложения). Сначала не существовало стандартного события `onOrientationChange`. Вместо него приходилось использовать таймер, который регулярно проверял текущую ориентацию устройства. Классы CSS переключались с помощью JavaScript на базе возвращаемого значения.

В первые выходные после выпуска iPhone Джо Хьюитт написал iUI. Это была первая библиотека JavaScript и CSS для iPhone. Автор поделился ею со всеми присутствовавшими разработчиками. Он, Николь Лазарро и трое других специалистов создали Tilt. Это была первая игра для iPhone, в которой была задействована функция отслеживания движений, уже имевшаяся на этом устройстве. Дори Смит написала iPhone Bingo — первую из игр для iPhone, написанных на чистом языке JavaScript. Ричард Херрера, Райан Кристиансон, Вай Сето и я разработали первый AJAX-мэшап (интегрированное веб-приложение) с функциями Twitter и просмотра спортивных трансляций профессиональной бейсбольной лиги. В этой программе пользователь мог виртуально просматривать любые бейсбольные трансляции и сразу же отправлять твиты о них. Это было поразительное раскрепощение: впервые я смогла применить сразу несколько фоновых изображений, граничных изображений, селекторов CSS3, а также настройки непрозрачности. При этом не приходилось беспокоиться об обеспечении поддержки всех этих возможностей в разномастных браузерах, браузерных версиях и операционных системах.

В течение первых девяти месяцев существования iPhone на нем работали только веб-приложения и некоторое количество нативных приложений, которые контролировались Apple. Вне Apple разработка нативных приложений для iOS не велась. Поскольку ширина полосы передачи данных была сильно ограничена, а документация для разработки под Apple оставалась крайне дефицитной, веб-приложения для iPhone не произвели никакого фурора. Поскольку браузер Safari из iPhone WebKit не мог получить доступ к нативным возможностям операционной системы iPhone, разработка веб-приложений для iPhone так и не прижилась. Разработка приложений для iOS по-настоящему развернулась только после выхода инструментария разработчика (SDK).

Выход SDK. Появление сторонних приложений

Выход первой версии SDK для iPhone состоялся 6 марта 2008 года. Этот инструментарий позволял сторонним разработчикам (не являющимся сотрудниками Apple) создавать приложения для iPhone, а позже — и для iPod touch, и для iPad. Уже в июле 2008 года открылся рынок App Store. С появлением таких

возможностей (выход SDK и начало работы рынка App Store), не говоря уж о возможности для разработчиков свободно торговать своими приложениями на этом рынке, основное внимание во всем мире оказалось привлечено к нативной разработке для iPhone.

Тот факт, что с момента выхода SDK большинство приложений, создаваемых для iPhone, являются нативными, кажется вполне логичным итогом описанных событий, но ведь времена изменились! В 2008 году веб-приложения для мобильных устройств явно проигрывали нативным, поэтому специалисты во всем мире предпочитали нативную разработку веб-разработке. Достаточно сопоставить следующие факты.

2008 год. Недостатки мобильных веб-приложений

- Размер файлов в браузере Safari для iPhone не может превышать 10 Мбайт.
- Негде хранить данные веб-приложений, размер кэша очень ограничен.
- Большинство возможностей CSS3 и HTML5 не поддерживаются не только в Safari для iPhone, но и во всех других браузерах.

2008 год. Достоинства нативных приложений

- Удобство разработки с применением Xcode.
- Возможность продавать приложения на рынке App Store.

Но в 2013 году все кардинально изменилось. Аргументы в пользу веб-приложений практически уравновесили соответствующие контраргументы — если не превзошли их. Судите сами.

2013 год. Достоинства создания веб-приложений

- Проще вести разработку и выстраивать итерации (разработчик может сдавать готовый код даже несколько раз в день, итерации получаются очень быстрыми).
- Можно пользоваться имеющимися навыками работы с HTML и CSS (выстраивать новые навыки на базе старых, а не требовать от разработчиков осваивания совершенно новых умений).
- Везде одни и те же технологии, одна и та же платформа.
- Потенциал для кросс-платформенности.

2013 год. Недостатки создания нативных приложений для iPhone

- Процесс рецензирования приложения, необходимый для распространения его через App Store, может длиться три недели и даже больше.
- Риск того, что контент приложения не пройдет цензуру и, соответственно, программа не попадет на рынки приложений.
- Ежегодный взнос разработчика за пользование рынком App Store составляет $99, кроме того, Apple берет 30%-ную комиссию с вашей выручки.
- Приходится долго ждать появления на рынке подготовленной вами обновленной версии приложения и еще дольше — того, что пользователи обновят свои экземпляры вашей программы (при работе с HTML5 все сделанные изменения вступают в силу немедленно).

Что нового? Новые элементы и API

Разработка HTML5[1] велась на протяжении долгих лет. Начало ей было положено еще в 2004 году в рамках проекта, называвшегося Web Applications 1.0. Эта версия языка до сих пор не закончена, но некоторые его части уже доведены практически до совершенства и поддерживаются многими современными браузерами (зачастую — в полном объеме). В число первоклассных современных браузеров входят Safari, Chrome, Internet Explorer 10+, Firefox и Opera. IE8 и старше не относятся к этому списку. В IE9 частично обеспечивается поддержка HTML5, но этот браузер только тормозит развитие современной Сети. Итак, пусть не все браузеры поддерживают HTML5, такая поддержка существует во всех браузерах WebKit/Blink, Opera Mobile[2], устройствах с Firefox OS и на новых Windows Phone. Безусловно, пришло время как следует поэкспериментировать с HTML5.

HTML5 — это обобщающий термин, описывающий стандарты новых веб-API. Некоторые из этих стандартов (например, перетаскивание) описаны в спецификации HTML5, а другие (геолокация) — нет.

Вооружившись HTML5 и API, связанными с ним, мы можем не ограничиваться разработкой нативных приложений. Если бы мы попытались описать в этой книге все аспекты спецификации HTML5 и спецификаций API, связанных с этим языком, то на каждый экземпляр книги пришлось бы погубить целую елочку. Поэтому я не буду вдаваться здесь в лишние детали, а остановлюсь только на тех, которые вы можете реализовать и внедрить уже сегодня. Краткий обзор этих тем сделан в следующих подразделах.

Теги для семантической группировки

В HTML5 предоставляются новые теги, применяемые для определения логических групп элементов и разделов в рамках разметки. Если при определении верхних и нижних колонтитулов, навигационной области и т. д. применяются теги для семантической группировки, а не несемантические `<div>` и `<span>`, то поисковикам становится проще распознавать структуру вашего сайта. О новых элементах для семантической группировки поговорим в главе 3.

---

[1] Термин HTML5 уже стал собирательным. Лишь одно из его значений соответствует языку разметки HTML5 как таковому. Брюс Лоусон предложил использовать в качестве родового понятия аббревиатуру NEWT (New Exciting Web Technologies — новые захватывающие веб-технологии). Этот термин кажется мне глупым, но персонаж-талисман NEWT нравится. — *Примеч. авт.*

[2] В Opera Mini качественная поддержка HTML5 отсутствует и никогда не появится. Это браузер иного типа — так сказать, прокси-браузер, чьи возможности были намеренно ограничены ради экономии полосы передачи данных. Opera Mini запрашивает веб-страницы через серверы Opera, которые обрабатывают и ужимают данные перед отправкой их на мобильный телефон. В результате объем передаваемых данных радикально снижается. Такая предварительная обработка улучшает совместимость браузера с веб-страницами, которые изначально не были рассчитаны на просмотр с мобильных устройств, но интерактивность сайта и набор его возможностей значительно ограничены. — *Примеч. авт.*

Веб-формы

В Сети имеются миллионы форм. Все эти формы сопровождаются сценариями (скриптами), предназначенными для валидации адресов электронной почты, создания всплывающих календарей, обеспечения присутствия информации в полях, заполнение которых обязательно, когда форма попадает в фокус. В HTML5 можно полностью обойтись без валидации форм, требующей применения JavaScript! В HTML5 элементы форм обогатились новыми возможностями и методами для определения типов данных.

В главе 4 мы рассмотрим некоторые из усовершенствованных элементов форм. Научимся создавать нативные слайдеры, замещающий текст, поля для выбора календарной даты. Кроме того, научимся валидировать адреса электронной почты, обеспечивать наличие информации во всех полях, обязательных для заполнения, и отображать специальные варианты клавиатуры в зависимости от типа ввода. При решении всех этих задач мы совершенно не будем пользоваться JavaScript!

Масштабируемая векторная графика и холст

Работая с HTML5, можно больше не делать изображения встраиваемыми объектами. В этой версии HTML появились нативные элементы `<svg>` и `<canvas>`, оба они улучшены с помощью CSS и доступны через объектную модель документа. Добавляя любой из этих двух элементов, браузер предоставляет участок пустого холста, на котором вы можете рисовать программными средствами. Об элементах `<svg>` и `<canvas>` мы поговорим в главе 5.

Видео и аудио

В настоящее время существуют необходимые плагины (подключаемые модули) для любого браузерного видео и аудио. В HTML5 появилась нативная браузерная поддержка для видео и аудио. И такие медиаформаты обзавелись поддержкой сценариев! Браузеры с возможностями HTML5 нативно поддерживают форматы webM и mp4. Работая с объектной моделью документа, можно управлять видео и аудио, в том числе приглушать звук, перематывать ролик вперед и останавливать его. С помощью CSS можно оформлять программы-плееры. Пусть операционная система iOS никогда не будет поддерживать технологии Flash и Silverlight, все мобильные браузеры поддерживают видео и аудио, реализованные в виде возможностей HTML5. Мы поговорим об элементах `<video>` и `<audio>` в главе 5.

Геолокационный API

Геолокация не описана в спецификациях HTML5, а является *ассоциированным API* — кстати, очень полезным модулем. Геолокация — это процесс определения местоположения мобильных устройств или ПК. Геолокационный API рассмотрен в главе 6.

Офлайн-контент и хранилище данных

Очевидный факт: смартфоны — это мобильные устройства. Мобильный Интернет функционирует крайне ненадежно. API кэша приложений HTML5, локального хранилища данных и баз данных обеспечивают приятное и безотказное пользование веб-приложениями, даже если ваш мобильный оператор оказывает некачественные услуги. API, отвечающие за офлайновую работу ваших приложений, рассмотрены в главе 6.

Другие API

В главе 6 мы также коротко поговорим о микроданных, ARIA (движке для работы с базами данных) и веб-работниках. Следует отметить, что эти технологии визуально и функционально никак не отражаются на веб-странице или в веб-приложении. Тем не менее они довольно полезны. С помощью микроданных можно добавлять к содержимому машиночитаемую семантику, которая важна для улучшения работы поисковых роботов. ARIA (аббревиатура расшифровывается как Accessible Rich Internet Applications — стандарт доступности активных интернет-приложений) также никак не отражается на визуальном представлении вашего контента, но улучшает его доступность. Дело в том, что эта технология предоставляет атрибуты, помогающие объяснить роли и функции «взломанных» элементов. Так называются элементы, используемые для передачи нетипичной для себя (не задаваемой по умолчанию) информации. Наконец, мы кратко коснемся темы веб-работников. Веб-работники — это дополнительные фоновые потоки с кодом на JavaScript. Они позволяют выполнять интенсивные вычисления, связанные с кодом JavaScript, не нагружая при этом поток пользовательского интерфейса. Поверьте мне, глава 6 вас не разочарует!

Что нового в CSS

Каскадные таблицы стилей (CSS) также обогатились захватывающими новыми возможностями. Селекторы CSS3, описанные в главе 3, предоставляют нам методы для целенаправленной работы практически с любым элементом, имеющимся на странице. Для этого не потребуется добавлять ни одного класса. Эти возможности распространяются и на медиазапросы. В результате обеспечивается отличная адаптивная веб-разработка. В CSS3 появились новые альфа-прозрачные цветовые значения RGBA и HSLA, рассмотренные в главе 8. Читателей, занимающихся проектированием и прототипированием, наиболее заинтересуют главы 9 и 10. В них рассмотрены новые и не совсем новые возможности CSS3, в том числе:

- множественные фоны;
- переходы;
- трансформации;
- 3D-трансформации;
- градиенты;

- свойство `background-size`;
- свойство `border-image`;
- свойство `border-radius`;
- свойство `box-shadow`;
- свойство `text-shadow`;
- свойство `opacity`;
- свойство `animation`;
- свойство `columns`;
- свойство `text-overflow`.

Веб-шрифты. Новые веб-шрифты позволяют работать с более широким набором гарнитур — не только с традиционными *безопасными веб-шрифтами*, которых насчитывается около полудюжины. Реализации веб-шрифтов различаются в разных браузерах. В частности, сильно различается поддержка для iPhone и ПК. Все браузеры для смартфонов поддерживают `@font-face`, но при разработке для мобильных устройств следует пользоваться шрифтом без засечек, например Helvetica, Roboto или любым другим шрифтом, задаваемым в операционной системе по умолчанию. Я не могу рекомендовать взыскательным пользователям мобильных устройств, чтобы они скачивали огромные файлы со шрифтами. В главе 11 советую пользоваться как можно более компактными иконочными шрифтами, но вообще веб-шрифты в этой книге почти не рассматриваются.

Проблемы, специфичные для мобильной среды

Работая с браузерами для ПК, большинство пользователей путешествуют по Интернету с помощью мыши и клавиатуры. На смартфонах и планшетах мыши нет, и навигация по Сети выполняется голыми руками и пальцами: мы вращаем устройство, встряхиваем его, касаемся экрана и нажимаем на него. Но феномен кликащелчка на мобильном устройстве просто отсутствует. Даже самые худощавые пользователи страдают из-за проблемы «толстого пальца». Суть этой проблемы заключается в том, что точность навигации при работе пальцами гораздо ниже, чем при использовании мыши. Следует также учесть, что экран мобильного устройства довольно мал, а продолжительность концентрации пользовательского внимания при работе с ним гораздо меньше, чем при работе с привычным дисплеем. Поэтому при проектировании пользовательского интерфейса для мобильного устройства и необходимости уложить контент в очень ограниченном пространстве мы руководствуемся соображениями, отличающимися от тех, которые учитываем при программировании для ПК.

Работая дома с мобильными планшетами, мы обычно выходим в Интернет через Wi-Fi или другие точки беспроводного доступа. Мобильные телефоны также могут использовать такие точки доступа, но чаще всего обращаются к Интер-

нету через ненадежные и маломощные разделяемые службы. Кроме того, экраны смартфонов невелики. На них почти не остается места для лишних панелей инструментов, а полоса доступа слишком узка, чтобы в ней поместились огромные библиотеки JavaScript и изображения.

В главе 11 рассмотрены возможности адаптивного веб-дизайна. В главе 12 обсуждаются вопросы, связанные с проектированием. Обработчики уникальных событий для мобильных устройств и сенсорных экранов — тема главы 13. Производительность в мобильной среде, отладка и технологические ограничения мобильных устройств обсуждаются в главе 14.

Что особенного в этой книге

Мы, веб-разработчики, слишком задержались в прошлом. Мы вынуждены работать с оглядкой на браузер, которому уже более 12 лет. Если вам не приходится беспокоиться о кросс-браузерной совместимости, а кроме того, вы можете игнорировать ограничения, свойственные для CSS2, то веб-разработка становится гораздо интереснее. На мобильных устройствах устанавливаются продвинутые браузеры, в которых задействуются ультрасовременные технологии. Так пользуйтесь же этими технологиями!

Этот дивный новый мир открывается перед нами именно благодаря мобильным технологиям. WebKit с поддержкой HTML5 имеется на планшетах Android, смартфонах iPhone, телефонах OpenMoko и Blackberry, и этим список не ограничивается. Кроме устройств с Blackberry, Android и iOS, движок WebKit присутствует также в браузерах Bolt, Dolphin, Ozone и Skyfire. На сотовых телефонах устанавливаются и мобильные версии браузеров Firefox, Opera и IE. Наконец, существует множество «некомпьютерных» устройств, где по-прежнему встречается продвинутый вариант Opera, основанный на браузере Presto. Opera и Chrome портируются на Blink. Недалек тот день, когда полнофункциональные браузеры будут установлены не только на каждом мобильном телефоне, но и в телевизоре, автомобиле и даже холодильнике.

Размер экрана

Прямо сейчас работая на ПК, пользователи Internet Explorer могут чувствовать себя обделенными, так как этот браузер не соответствует многим новым и готовящимся стандартам. С распространением стандартизированных браузеров и одновременным сокращением популярности старых версий Internet Explorer мы вскоре сможем повсюду полагаться на CSS3. Разрабатывая приложения для мобильной среды, можно забыть об ограниченных возможностях CSS2. Тем не менее одна важная проблема никуда не исчезла: у нас жесткий дефицит экранного пространства! Один размер не подходит для любых устройств. Совершенно очевидно, что окно мобильного браузера меньше, чем окно браузера для ПК.

На некоторых сайтах может быть допустимым подход «один размер на все случаи жизни», но большинство HTML-файлов и CSS-документов не подходят сразу для всех браузерных размеров.

В зависимости от сложности контента и его компоновки можно выдавать пользователю разные варианты HTML и CSS в зависимости от конечного устройства.

Иногда удается просто временно скрывать определенный контент. В других случаях бывает достаточно выдать уменьшенную версию заголовка и более мелкие изображения. Кроме того, на большом экране можно использовать многостолбцовый макет, а на экране мобильного телефона — одностолбцовый. В зависимости от размера устройства может потребоваться изменить внешний вид страницы: например, на ПК удобнее читать трехстолбцовый макет, но на экране мобильного устройства целесообразнее разместить эти три столбца по порядку друг над другом.

Мобильный веб-дизайн должен быть максимально простым. На небольшом экране мобильного телефона вы в любом случае сможете уместить совсем мало информации. Прокрутка допускается только для сравнительно длинных статей, но не для главной или навигационной страницы.

Можно написать отдельный вариант разметки для мобильной версии вашего сайта, но это не обязательно. Вообще создание дополнительного варианта разметки оправданно лишь для веб-приложения, при написании обычного сайта это лишнее.

Чего хотят пользователи

Принято считать, что с мобильных устройств в Интернет выходят только пользователи, которые куда-то спешат. Действительно, некоторые мобильные пользователи всего лишь быстро просматривают страницу, стремясь найти конкретные сведения. Например, они могут проверить онлайновый список продуктов, за которыми пошли в магазин, уточнить ингредиенты для овощной запеканки или найти самый лучший итальянский ресторан, до которого можно дойти за 5 минут.

Конечно, пользователь вряд ли заинтересуется тем, какова корпоративная организация компании, владеющей данным рестораном. Но если такая информация ему все-таки понадобится, то мы должны учитывать, что кто-то из пользователей попробует ознакомиться с ней с того же самого мобильного устройства. Может быть, нам кажется, что такой сложный поиск информации лучше делать на ПК, но в настоящее время все больше пользователей выходят в Интернет только с мобильных устройств.

Возможно, среднестатистический пользователь, просматривающий сайт на бегу с мобильного телефона, действительно всего лишь захочет посмотреть адрес, номер телефона и обновление статуса. Он отнюдь не собирается удалять, реорганизовывать, редактировать информацию на iPhone или заниматься долгими поисками. Но вообще-то такая возможность не исключена. Может быть, у пользователя нет ПК, а есть только мобильное устройство. Поэтому следует гарантировать, что самая важная информация всегда будет доступна в любой версии сайта и пользователь теоретически сможет выполнить на мобильном устройстве те же задачи, которые обычно решает на широкоэкранном мониторе.

Позаботьтесь также об удобстве использования. На устройствах с сенсорным экраном основным инструментом ввода является палец, а не мышь. Палец значительно менее точен, чем курсор. На устройствах с сенсорным экраном интерактивные элементы должны быть крупными и разделяться широкими отступами. Об особенностях пользовательского интерфейса на устройствах с сенсорным экраном мы поговорим в главе 13.

В мобильной версии разметки не должно быть изображений, которые не являются презентационными. Как правило, изображения оптимизируются именно для мониторов ПК, а не для мобильных устройств. Они занимают место, которое в условиях дефицита экранного пространства лучше отдать под контент. Полоса передачи данных также может быть очень дорогостоящим ресурсом, а из-за изображений такая передача слишком замедлится. Да, нужно включать изображения, если они важны в конкретном контексте. Но если они являются декоративными, то лучше обойтись фоновыми картинками.

О чем эта книга

В главе 1 рассматривается настройка нашей среды разработки и вкратце обсуждаются примеры, изучаемые в других главах книги.

В главах 2–6 мы поговорим о том, что нового появилось в HTML5. Обсудим наилучшие методы написания семантической разметки, совместимой со всеми современными браузерами как на ПК, так и в мобильном пространстве. Поговорим о новых семантических элементах HTML5, технологии Web Forms 2.0, а также о некоторых API HTML5 и других связанных с ними API — в частности, об обеспечении геолокации. Коснемся вопросов, связанных с масштабируемой векторной графикой (SVG), холстом, веб-формами, видео, аудио, кэшем приложений AppCache, базой данных и веб-работниками.

В главах 7–11 мы познакомимся со всеми новинками, которые уже присутствуют в CSS3 или готовятся к выходу. Это новые цвета, тени, граничные изображения, скругленные углы, анимация. У вас в арсенале будут все необходимые инструменты для создания красивых веб-приложений, которые отлично отображаются и в мобильных, и в настольных браузерах. В главе 11 особый акцент делается на возможностях адаптивного веб-дизайна.

В главах 12–14 обсуждаются характерные черты мобильных платформ: события касания, особенности пользовательского восприятия, вопросы, связанные с производительностью мобильных устройств. Приемы, изучаемые в этих главах, гарантируют высокую производительность сайта, качественное пользовательское восприятие и надежность отображения веб-страниц на любых платформах.

Итак, наша основная цель — научиться разрабатывать сногсшибательные сайты для мобильных устройств. Первым делом потребуется создать классный сайт, а уже потом адаптировать его для мобильной среды. Конечно, для простоты вы должны разрабатывать сайт на ПК. Но при этом всегда следует продумывать, как сайт будет использоваться на смартфоне, планшете или другом подобном устройстве. Мы будем создавать приложения, корректно работающие на смартфонах в любых современных браузерах.

Условные сокращения, используемые в книге

В данной книге применяются следующие условные обозначения.

Шрифт для названий

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

`Шрифт для команд`

Применяется для обозначения программных элементов: переменных, названий функций, типов данных, переменных окружения, операторов, ключевых слов и т. д.

`Шрифт для листингов`

Используется в листингах программного кода.

`Полужирный шрифт для листингов`

Указывает команды и другой текст, который должен воспроизводиться пользователем буквально.

`Курсивный шрифт для листингов`

Обозначает текст, который должен быть заменен значениями, сообщаемыми пользователем или определяемыми в контексте.

ПРИМЕЧАНИЕ

Здесь помещен совет, замечание практического характера или общее замечание.

ВНИМАНИЕ

Это предостережение.

Работа с примерами кода

Ресурсы к книге доступны по адресу http://www.standardista.com/mobile. Там вы можете найти ссылки на полезные сайты, примеры кода, а также все ссылки, упоминаемые в этой книге.

Эта книга написана для того, чтобы помочь вам в работе. В принципе, можете использовать код, содержащийся в этой книге, в своих программах и документации. Можете не связываться с нами и не спрашивать разрешения, если собираетесь воспользоваться небольшим фрагментом кода. Например, если вы пишете программу и кое-где вставляете в нее код из этой книги, разрешения на это не требуется. Однако, если вы запишете на диск примеры из книг издательства O'Reilly и начнете раздавать или продавать такие диски, на это необходимо получить разрешение. Если вы цитируете эту книгу, отвечая на вопрос, или воспроизводите код из нее в качестве примера, на это не требуется разрешения. Если вы включаете значительный фрагмент кода из этой книги в документацию по вашему продукту, на это требуется разрешение.

Как с нами связаться

Замечания и вопросы, связанные с этой книгой, направляйте по следующему адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-99-38 (в США или Канаде)
(707) 829-05-15 (международный или местный телефон)
(707) 829-01-04 (факс)

Благодарности

Благодарю Брюса Лоусона, Адама Лихтенштейна, Дженнифер Хейнен, Тима Кадлеца, Джеффа Бертофта, Томоми Имуру и Джастина Лоуэри.

Брюс Лоусон выступил соавтором первой книги по HTML5, которая называлась *Introducing HTML5*[1]. Он также является одним из создателей сайта www.HTML5Doctor.com и входил в состав рабочей группы консорциума W3C, занимавшейся подготовкой рекомендаций по веб-разработке для мобильных устройств. Он продвигает свободные веб-стандарты для Opera — старейшего производителя браузеров, чьи программы-обозреватели для мобильных устройств, ПК, телевидения и встроенных систем служат более чем 300 млн человек во всем мире (www.opera.com). Читайте Брюса в Twitter по адресу @brucel, а также на сайте www.brucelawson.co.uk.

Джастин Лоуэри разработал внешнее оформление для игры CubeeDoo. Он работает архитектором пользовательских взаимодействий в своей компании Cerebral Interactive, которая специализируется на проектировании и разработке веб-приложений и мобильных приложений. Он работал дизайнером в сфере полиграфии с 2001 года, а с 2006-го занимается веб-разработкой. Кроме того, он является IT-специалистом в области медицины, в настоящее время его целью являются революционные прорывы в сфере использования информационных технологий в медицинском образовании. Читайте Джастина в Twitter: @cerebralideas — и на сайте www.cix.io.

Адам Лихтенштейн — разработчик клиентского кода и горячий любитель технологий OOCSS (объектно-ориентированные каскадные таблицы стилей) и языка Sass. Он создатель программы FormFace, предназначенной для семантического построения и стилевого украшения форм HTML5. В настоящее время трудится клиентским разработчиком и дизайнером в компании Wufoo и пишет свою первую книгу по клиентской разработке. Если Адам не пишет код и не пишет о коде, то предается своему любимому развлечению — размышляет о коде. Читайте его в Twitter по адресу @seetroughtrees и на сайте http://seethroughtrees.github.io.

---

[1] Русское издание — *Лоусон Б., Шарп Р.* Изучаем HTML5. — СПб.: Питер, 2011. — *Примеч. пер.*

Дженнифер Хейнен — мобильный разработчик, дизайнер и фотограф, страстно желающий влюбить всех окружающих в мобильную разработку — так же, как любит ее сама. Мисс Хейнен разработала свой первый сайт в 1996 году для знакомой рок-группы. С 2000 года она занимается консультированием в области мобильной и веб-разработки, совмещая эту работу с постом адъюнкт-профессора по веб-дизайну и истории искусства. Читайте ее Twitter по адресу @msjen либо посетите ее сайт http://blackphoebe.com/msjen.

Томоми Имура — убежденная сторонница открытой Сети, разработчик клиентского кода, специализирующаяся в области мобильного программирования с тех давних пор, когда эта сфера еще не была всеобщим увлечением. Она занималась мобильной разработкой, платформенным дизайном взаимодействий и пользовательских интерфейсов, а также работой с фреймворками в Yahoo!, участвовала в разработке Mobile и webOS в Palm до перехода в Nokia, сотрудничала с W3C и занималась продвижением HTML5. Ее Twitter — @girlie_mac, сайт — http://girliemac.com.

Джеф Бертофт занимается продвижением HTML5 в компании Microsoft, активно участвует в работе сообщества программистов JavaScript/HTML5. Мистер Бертофт — убежденный сторонник веб-стандартов. Ему понравится любой язык программирования, если этот язык — JavaScript. Кроме того, он является автором книги *HTML5 Hacks*, вышедшей в издательстве O'Reilly, а также автором-основателем блога htmlhacks5.com. Он живет на юге Техаса с женой и тремя детьми. Читайте его в Twitter по адресу @boyofgreen.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты vinitski@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

1 Подготовка к изучению API для работы с мобильным HTML5, CSS3 и JavaScript

Если мы с вами похожи, то вы уже не первый год испытываете отвращение к старым версиям Internet Explorer. Эти браузеры — настоящие фабрики неисправностей. И они были переполнены такими неисправностями на протяжении всего своего существования[1]. Все мы знаем, что IE6 плохо работал. Как только мы научились решать проблемы IE6 с помощью полизаполнения, мы избавились сразу от массы проблем.

В области мобильных технологий также случаются неудачи, но это происходит по-новому, более разнообразно и изменчиво. Различные версии браузеров на разных устройствах могут поддерживать множество новых возможностей, но делать это совершенно по-разному. Либо браузер может поддерживать ту или иную возможность, но пользоваться ею будет неудобно. Например, некоторые современные устройства поддерживают объект `LocalStorage`, другие — нет. Некоторые устройства, поддерживающие `LocalStorage`, не позволяют записывать информацию в эту базу данных. Даже если браузер допускает считывание из `LocalStorage`, оно может протекать очень медленно и отрицательно сказываться на производительности. А если даже браузер разрешает вам записывать в `LocalStorage`, приложение может быстро исчерпать свободное место в хранилище.

Мы не можем рассмотреть в этой книге все нюансы всевозможных браузеров в разнообразных операционных системах и на разных устройствах. Даже если бы я знала все эти мелочи (а я знаю далеко не все), то мог бы получиться целый фолиант, который уже устарел бы к тому моменту, как я его дописала. Да и эта книга уже устарела. Технологии непрерывно меняются. Написать абсолютно актуальную книгу на темы, связанные с программированием, попросту невозможно, так как небольшие изменения возникают даже за время, требующееся на подготовку одной

---

[1] В 2001 году, когда вышел IE6, он считался ультрасовременным браузером. Он практически не имел конкурентов на браузерном рынке, поэтому совершенно не обновлялся. — *Примеч. авт.*

главы. Вполне вероятно, что некоторые упоминаемые здесь браузеры, возможности, модели смартфонов и интернет-ресурсы уже устарели. Вот с какой установкой следует читать эту книгу: если вы пользуетесь в работе только рекомендуемыми методами и пишете код в соответствии со стандартами, то ваш код будет работать как на современных устройствах, так и на устройствах, которые появятся в обозримом будущем.

Я описываю возможности, которые поддерживаются в том или ином браузере, но не останавливаюсь отдельно на возможностях, которые в нем не поддерживаются. Полагаю, что все браузеры будут развиваться и совершенствоваться. Если сегодня в браузере имеется определенная проблема, то уже завтра она может быть решена.

Поэтому при использовании той или иной возможности вы должны не только обнаружить ее в браузере, но и протестировать. Только так вы сможете быть уверены в том, что она хорошо поддерживается и ее можно задействовать.

Вся разметка, используемая в этой книге, равноценна для любых устройств, операционных систем и браузеров, а также не требует подключения каких-либо библиотек JavaScript. В этой книге я смогла обходиться без библиотек и писать код на базовом JavaScript (эту наиболее простую версию языка JavaScript еще называют ванильной). Базовый JavaScript нужен именно для того, чтобы вы научились сами писать код. Выбрав этот вариант языка, я постаралась избавиться от любой путаницы, вызывающей вопросы типа: «А это нативный метод или он взят из какого-то фреймворка?»

Это не означает, что вы вообще не должны пользоваться библиотеками. Скорее наоборот! Свободно распространяемые библиотеки — одни из лучших источников, в которых можно познакомиться с финтами разных браузеров. В свободных проектах с открытым кодом участвуют сотни, а иногда и тысячи энтузиастов. Эти люди рассматривают проблему с тысяч точек зрения, разрабатывают и тестируют код на множестве устройств, находят разнообразные странности, сообщают о них, а также оперативно исправляют библиотеки, устраняя те или иные причуды браузеров либо предлагая обходные маневры и полизаполнение. Все эти специалисты заняты также описанием ошибок, уведомлением производителей браузеров о найденных проблемах, например о том, что та или иная деталь не соответствует стандартам. Поэтому в будущих версиях браузеров многие ошибки удается исправить.

Популярные свободно распространяемые библиотеки, а также API JavaScript, имитирующие функции HTML5, применяемые в старых браузерах (такие решения на JavaScript и называются полизаполнениями), — это отличные ресурсы для быстрого обнаружения браузерных причуд, позволяющие быстро с ними справляться. Эти библиотеки входят в джентльменский набор каждого хорошего веб-программиста. Если вы даже их не используете, обязательно читайте исходный код, знакомьтесь с ошибками, попадающимися в мобильных браузерах и уже обнаруженными другими специалистами.

Чтобы наиболее полно изучить HTML5, CSS3 и связанные с ними API JavaScript, нужно писать код. Давайте этим займемся.

CubeeDoo: игра для мобильных браузеров на HTML5

Чтобы изучить HTML5 и CSS3, я в свое время решила написать веб-приложение для одного мобильного браузера, а потом поэкспериментировать с этой программой, постаравшись максимально расширить ее возможности. Первым экспериментом с CSS3 было веб-приложение Pickleview. Этот мэшап был тематически связан с соревнованиями взрослой бейсбольной лиги, а также включал в себя возможность взаимодействия с Twitter. Я написала его в 2007 году, в те самые выходные, когда на рынке появился iPhone. На тот момент самым совершенным браузером из присутствовавших на рынке был Safari для iPhone (может быть, с ним мог соперничать Opera). Поскольку я писала приложение для единственного браузера, я могла игнорировать сложности, связанные с поддержкой IE6, IE7 или Firefox 2 (браузер Chrome еще не появился). Да, такой была Сеть в 2007 году.

В 2010 году я вновь попробовала программировать в одном браузере, применяя самые современные на тот момент возможности HTML5 и CSS3. Вместе с несколькими друзьями мы создали мнемоническую игру с анимацией, хранилищем данных, офлайновыми функциями и практически всеми новыми возможностями, которые имелись в Chrome 12 на ПК, но отсутствовали в Safari 3.1 на смартфонах. Работая всего в одном браузере и максимально активно используя все доступные новые технологии, я научилась программировать новейшие модули на HTML5, CSS3 и JavaScript. Такие модули на тот момент еще не имели практического применения, так как требовалось поддерживать унаследованные браузеры. К 2010 году многие браузеры значительно усовершенствовались по сравнению с тем, какими они были в 2007 году. Другие (не будем показывать пальцем на IE) практически не изменились.

В 2013 году в большинстве браузеров поддерживаются HTML5 и CSS3. Мы как разработчики вынуждены сами себя ограничивать, поддерживая устаревшие браузеры для ПК, то есть IE9 и старше. В мобильном пространстве есть свой IE6. Нашу работу тормозят старые сотовые телефоны, а также некоторые смартфоны, на которых еще работает Android 2.3. Но даже браузеры для сотовых телефонов и операционной системы Android 2.3 поддерживают современные возможности.

Чтобы научиться программировать на HTML5, CSS3, а также применять современные API JavaScript, временно забудьте о старых браузерах. Вместе мы узнаем, какие новые возможности открываются перед нами благодаря этим новым технологиям. В примерах кода, приведенных в этой книге, я постаралась рассмотреть большинство возможностей, которые широко поддерживаются в современных браузерах.

На рис. 1.1 вы видите окно с игрой CubeeDoo. Это мнемоническая игра, написанная с применением только клиентского кода. В книге я буду пользоваться примерами кода из этой игры, а также из ее нативного аналога, написанного для iPhone. Разметка в игре выполнена с применением тегов HTML5. Некоторые темы, в частности совпадающие пиктограммы, созданы с использованием сгенерированного контента. В оформлении игры применялись CSS-трансформации, переходы и анимация, а также градиенты, скругленные углы и другие возможности CSS. В игре

также задействованы масштабируемая векторная графика, нотация JSON, нежелательная, но поддерживаемая в мобильной среде база данных webSQL, хранилища данных `LocalStorage` и `SessionStorage`, атрибуты данных, формы HTML5, медиазапросы, а также URI данных.

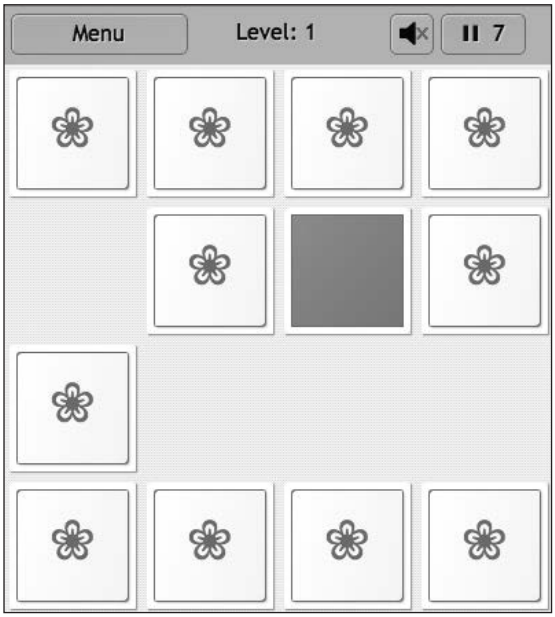


Рис. 1.1. Скриншот мнемонической игры CubeeDoo

Весь код к этой книге написан без применения каких-либо фреймворков. Как было указано ранее, все написано вручную на базовом (ванильном) JavaScript, HTML5 и CSS. Я хочу научить вас работать с реальными API, а не заниматься полизаполнениями. На боевых проектах вам, вероятно, придется прибегать и к полизаполнениям, но для того, чтобы с умом использовать такие решения, вы должны понимать, как именно они работают. Об этом и пойдет речь в книге, которую вы читаете.

В книге рассмотрены CSS3, HTML5 и связанные с ними API. Основное внимание уделяется технологиям, используемым в мобильной среде. Мы живем в мобильном мире, но отдельного «мобильного Интернета» не существует. Есть просто Интернет. Но если вы сосредоточитесь на веб-разработке только для ПК, то ваши творения могут оказаться непригодными для постоянно растущей аудитории пользователей, выходящих во Всемирную паутину только с мобильных устройств. Кроме того, если вы пишете сайты и приложения только для ПК, то вынуждены довольствоваться наименьшим общим знаменателем — тем набором возможностей, которые поддерживаются в старых версиях Internet Explorer.

Никогда не пускайте в работу приложение, нормально работающее лишь в одном браузере. Тем не менее если ваша цель — освоить технологии, которые только зарождаются, то, игнорируя старые браузеры, вы можете учиться, ставить себе более

высокую планку, в результате начнете мыслить нестандартно и преуспеете в своей профессии. Возьмите материал этой книги и попробуйте запрограммировать все рассмотренные в ней примеры в одном браузере. Проверьте, каковы истинные границы его возможностей. Экспериментируйте.

Все, что вам потребуется, — это браузер, интегрированная среда разработки (IDE) и некоторое время.

Инструменты разработки

Прежде чем приступать к разработке первого мобильного приложения, обязательно настройте среду разработки и оснастите ее самыми лучшими инструментами. Кстати, радуйтесь: эти инструменты у вас уже есть!

Чтобы проработать всю эту книгу, вам понадобится компьютер, на котором установлены как минимум текстовый редактор и браузер. Не обязательно даже иметь смартфон (впрочем, при чтении этой книги мобильное устройство вам весьма пригодится).

Текстовый редактор

Вы будете разрабатывать программы либо в обычном текстовом редакторе, либо в *интегрированной среде разработки* (IDE). Интегрированная среда разработки — это программный продукт, в состав которого, как правило, входят текстовый редактор и отладчик. Кроме того, IDE обладает другими возможностями (или плагинами). Например, в ней обычно поддерживается протокол передачи файлов (FTP), нужный для решения многих задач. У каждой IDE есть свои поклонники. Выберите ту, которая нравится вам больше всего. Я предпочитаю работать в Sublime Text, но вы можете воспользоваться TextMate, Dreamweaver, Eclipse, WebStorm или любой другой IDE. Конечно, можно обойтись и обычным текстовым редактором. Но, познакомившись с IDE, вы обнаружите, что она помогает организовать и ускорить процесс разработки. Рекомендую выбрать для работы одну IDE и как следует в ней разобраться. Хорошая IDE обычно оказывается очень мощным инструментом и позволяет сделать разработку программ максимально приятной.

Браузер

Вам понадобится браузер. Я предпочитаю заниматься разработкой в Chrome Canary — это бета-версия браузера Google Chrome. Этот браузер нравится мне потому, что в нем есть очень хороший отладчик. Во всех современных браузерах есть отладчики, но Canary обладает одним из лучших. Этот отладчик помогает разобраться со всеми новыми прибамбасами еще до того, как они войдут в очередной релиз браузера.

Если у вас нет компьютера Apple, вам будет довольно сложно разрабатывать нативные приложения для iPhone, iPad или iPod touch. Если у вас нет Windows 8, то

будет нелегко разрабатывать программы, которые условно объединяются под названием «приложения в стиле Metro». Но не волнуйтесь! Для работы с этой книгой вам понадобится всего лишь современный браузер. Конкретная операционная система или устройство не имеют значения. Вы можете протестировать все примеры из этой книги в Windows, UNIX, на смартфонах и планшетах Android, а также на Mac-компьютерах.

При мобильной веб-разработке вашими основными инструментами будут IDE и браузер для ПК. В процессе разработки мы будем предварительно просматривать и отлаживать ваше приложение в браузере. Некоторые важные черты в браузере для ПК сложно сымитировать. В частности, к ним относятся детализация изображения на мобильных устройствах, ограничения производительности JavaScript, рабочей памяти и ширины полосы доступа, а также доступность API. Тем не менее такие различия можно сгладить с помощью других инструментов, а также при непосредственном тестировании на реальных или виртуальных устройствах.

Вы можете с удовольствием заниматься разработкой в любимом браузере, но обязательно имейте в арсенале множество браузеров — для тестирования. Чтобы проверить, как приложение будет работать на Windows Phone, вам понадобится доступ к Internet Explorer. В браузерах Safari или Google Chrome можно протестировать работу приложения в операционных системах Android, Bada, BlackBerry и iOS. Для устройств с операционной системой Gecko понадобится браузер Firefox. В настоящее время для тестирования всех устройств, на которых используется движок визуализации Presto, нужен браузер Opera. Но поскольку Opera 14 построен на базе движка Chromium, а новейшие Opera и Chrome основаны на Blink[1], вам придется постоянно обновлять свои рабочие браузеры, чтобы они соответствовали тому мобильному ландшафту, в котором вы творите.

Если вы работаете на Mac, то скачайте Safari (если еще не сделали этого). Если пользуетесь Windows, скачайте новейший Internet Explorer. Если на вашем устройстве установлена операционная система Unix, скачайте также Chrome, Firefox и Opera. Также рекомендую обзавестись Chrome Canary, Aurora, Opera Next и сборками WebKit Nightly, чтобы тестировать код в готовящихся релизах основных браузеров. На момент написания этой книги именно перечисленные браузеры являются самыми важными, но не забывайте о том, что мобильный ландшафт постоянно меняется.

Инструменты отладки

Браузеры снабжены отладочными инструментами. Средства разработки, имеющиеся в браузере, — это встроенные инструменты, позволяющие проверять и отлаживать исходный код. С помощью этих инструментов можно управлять объектной моделью документа (DOM), редактировать и отлаживать код JavaScript, редактировать и отлаживать CSS, анализировать запросы ресурсов, а также проверять качество веб-контента и веб-приложений на «живом» материале.

---

[1] Blink — это ответвление компонента WebCore в составе WebKit, появившееся в версии 1 47503. Оно используется как браузерный движок в Chrome, начиная с версии 28, в Opera — с версии 15, а также во всех новых браузерах, работающих на базе движка Chromium.

Как правило, инструменты разработки скрыты, поскольку большинство пользователей, не занимающихся веб-программированием, не используют эти браузерные возможности. В мобильных браузерах также существуют некоторые отладочные возможности. Обычно доступ к этим инструментам с весьма ограниченным функционалом можно получить через интерфейс настроек устройства. Хотя и можно выполнять отладку на уровне устройства, намного удобнее отлаживать приложения с применением гораздо более надежных инструментов, которые есть в браузерах для ПК.

Отладчики для работы на ПК

Если вы разрабатываете сайты уже довольно долго, то вам, вероятно, знакомы названия Firebug[1], F12, Web Inspector и/или DragonFly. Firebug — это расширение для браузера Mozilla. F12, Web Inspector и DragonFly поставляются соответственно вместе с Internet Explorer, Chrome/Safari и Opera. Эти инструменты разработки обеспечивают отладку, редактирование и мониторинг HTML, CSS, DOM и JavaScript, присутствующих на сайте, а также позволяют анализировать некоторые виды информации, например HTTP-запросы, состояние локального хранилища данных, а также потребление памяти.

Firebug можно скачать на сайте getFirebug.com. Инструменты разработки Safari находятся в меню Develop (Разработка). Но доступ к ним нужно открывать отдельно, через команды Preferences ▸ Advanced (Настройки ▸ Расширенные) либо отметив опцию Show develop menu in menu bar (Отображать панель разработки на панели меню). В Chrome инструменты разработки открываются так: Инструменты ▸ Инструменты разработчика.

Отладчики Chrome, Safari, Firebug и Opera также можно открывать с клавиатуры, пользуясь сочетанием клавиш Command+Option+I или Control+I. Чтобы открыть Firebug и F12, достаточно нажать на клавиатуре F12. Перечисленные браузерные инструменты лучше всего подходят для отладки CSS, JavaScript и HTML.

Стоит также поближе познакомиться с инструментами Web Inspector, Error Console (консоль ошибок) и User Agent switcher. Эти отладчики позволяют проверять на веб-странице ее CSS, HTML JavaScript, DOM, а также заголовки. Независимо от того, чем именно вы пользуетесь — Web Inspector, Firebug, DragonFly, F12 или панелью Developer Tools (Инструменты разработчика) из Chrome либо какой-то комбинацией этих инструментов, — вы должны знать ваши отладочные инструменты очень хорошо. Отладчик вам пригодится не раз.

Вероятно, вы уже не один год пользуетесь браузерными инструментами отладки при разработке приложений для ПК, поэтому я не буду подробно останавливаться на этих инструментах. Однако, даже если вы работаете с ними уже лет пять, не исключено, что вы лишь слегка копнули возможности отладчика, которые

[1] В браузере Firefox есть определенный набор инструментов разработки, но большинство веб-программистов предпочитают использовать Firebug — это одно из расширений Firefox.

иной раз просто ошеломляют. Рекомендую вам самостоятельно получше познакомиться с этими инструментами, щелкая кнопкой мыши на каждой детали. О вкладке Timeline (Хронология) из инструментов разработки мы подробнее поговорим в главе 14.

Область просмотра на экране мобильного устройства. Чтобы сымитировать небольшой экран мобильного устройства, можно просто уменьшить размер окна браузера на ПК. Так у вас получится миниатюрная область просмотра, такая же как на экране того мобильного устройства, для которого вы хотите протестировать ваш ресурс. В браузере для ПК размер области просмотра равен размеру окна браузера. Область просмотра на экране мобильного устройства — это то, что вы видите, но при этом вы вполне можете видеть не всю мобильную веб-страницу, а только ее часть. Так или иначе, для большинства проверок, которые мы собираемся провести при тестировании, достаточно просто уменьшить размер окна браузера на ПК.

Когда вы изменяете размер окна браузера вручную, можете задавать для него случайные размеры. На панели Overrides (Приоритетные значения) окна Settings (Настройки) из Developer Tools (Инструменты разработчика) в Chrome (рис. 1.2) предоставляется несколько предустановленных размеров устройства. Чтобы получить доступ к настройкам Web Inspector, щелкните на значке шестеренки в углу окна Инструменты разработчика.

Когда вы выбираете устройство из меню User Agent (Пользовательский агент), Chrome переключает пользовательский агент в режим выбранного устройства, а затем создает область просмотра внутри окна браузера. Размер этой области соответствует размеру области просмотра, которой обладает выбранное устройство.

Если вы не найдете в списке нужного вам устройства, просто укажите значения высоты и ширины экрана устройства в двух предназначенных для этого полях (они находятся под параметрами устройства). Можно переключаться между параметрами, соответствующими книжной и альбомной ориентации устройства. Этот переключатель находится справа от параметров устройства. Познакомьтесь также с сайтом ScreenQueri.es — там предоставляется возможность предварительного просмотра для экранных параметров, которые вам точно известны. Кроме того, можно эмулировать сенсорный ввод либо использовать `thumbs.js` — это полизаполнение, функционально аналогичное TouchEvent.

На панели Developer Tools (Инструменты разработчика) в браузере Chrome вы можете также переопределять геолокационные данные и эмулировать интересующие вас координаты широты и долготы. Если же у вас на ноутбуке установлен гироскоп, то есть возможность даже эмулировать изменение ориентации устройства.

После того как вы закончите первый этап разработки приложения (он пройдет в браузере для ПК), понадобится протестировать готовый код на мобильном устройстве. Основное неудобство при тестировании на мобильном устройстве заключается в том, что у вас не будет доступа к мощным инспекторам, которыми вы привыкли пользоваться при разработке на ПК. Вот здесь вам и пригодятся удаленные инспекторы.

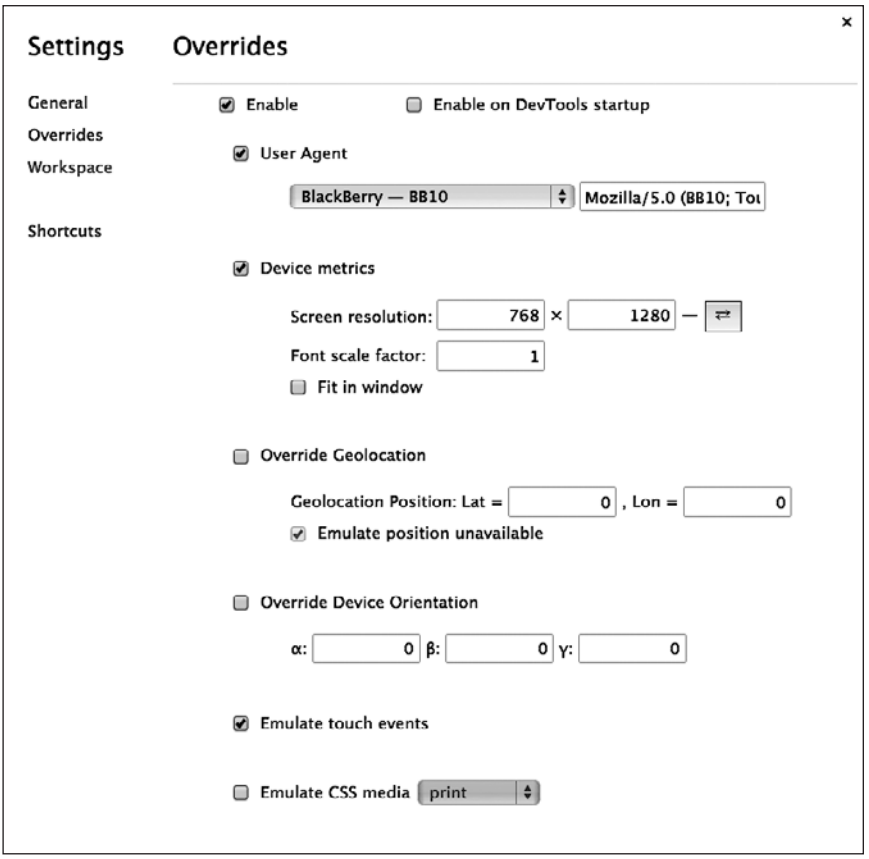


Рис. 1.2. Панель Overrides (Приоритетные значения) в разделе настроек (Settings) панели Developer Tools (Инструменты разработчика) браузера Chrome

Удаленная отладка

Существуют инструменты, предназначенные для удаленной отладки мобильного браузера через браузер для ПК. Удаленные отладчики позволяют браузеру ПК связываться с внешними устройствами, а затем удаленно выполнять на них код или перехватывать его. Удаленная отладка, как и традиционная, позволяет вам проверять HTML и CSS, манипулировать объектной моделью документа, заниматься редактированием в реальном времени и отлаживать сценарии.

Браузерный движок Opera в ближайшее время будет заменен. Конечно, сложно сказать, каким он станет в будущем, но уже с 2008 года компания Opera поддерживает удаленную отладку мобильного браузера Opera через отладчик DragonFly, который устанавливается в браузере Opera для ПК. Opera предоставляет возможности удаленной проверки HTML и CSS, обновления объектной модели документа, добавления контрольных точек, а также поддерживает любые другие операции, выполнимые в DragonFly на ПК.

WebKit обеспечивает поддержку удаленной отладки с подключением через USB-порт в Android, начиная с версии 4, и в iOS, начиная с версии 6. Чтобы использовать для удаленной отладки браузер Chrome, его нужно запускать не через ярлык на Рабочем столе, а из командной строки, со специальным флагом:

```
chrome.exe --remote-debugging-port=9222 --user-data-dir=remote-profile
```

или

```
/Applications/Chromium.app/Contents/MacOS/Chromium --remote-debugging-port=9222
```

Для отладки мобильного браузера Firefox добавьте к Firebug интерфейс прикладного программирования Debug (ранее это расширение называлось Crossfire).

Разумеется, текущая ситуация постоянно меняется в лучшую сторону. Если эта тема вам интересна, следите за ходом разработки протокола удаленной отладки (Remote Debugging Protocol), которой занимается рабочая группа по браузерному тестированию и инструментам.

Отладочный инструментарий Android

В инструментарии для разработки под Android (Android SDK) содержатся библиотеки API и инструменты для программирования, позволяющие собирать, тестировать и отлаживать приложения для Android. Вы можете заниматься отладкой веб-приложений непосредственно на устройстве или в эмуляторах, которые создаются средствами того же SDK (рис. 1.3).

Рис. 1.3. Эмулятор операционной системы Android 4.2.2, работающий в OS X

Вы можете скачать SDK по адресу http://developer.android.com/sdk/index.html. В этом комплекте вы найдете отладочный мост Android (ADB), другие отладочные возможности, а также средства для отслеживания консоли, создания и запуска эмуляторов.

В загруженных ресурсах найдите каталог tool и откройте android для доступа к ADB. В ADB предоставляются различные возможности управления устройствами. В частности, мост позволяет перемещать файлы в эмулятор и синхронизировать их, запускать на устройстве или на эмуляторе командную оболочку UNIX. Также здесь вы найдете средства общего назначения для обмена информацией с подключенными эмуляторами и устройствами.

В качестве альтернативы можете попробовать ADB-плагин. Это расширение для браузера Chrome, запускающее ADB-демон и обеспечивающее удаленную отладку кода на мобильных устройствах. Скачивать SDK при этом не требуется.

В вышеупомянутом каталоге tools можете открыть `Monitor`, обеспечивающий доступ к отладочному монитору Android (Android Debug Monitor). В этом инструменте есть консоль, через которую вы можете отлаживать свои приложения, в частности, просматривать любые методы `console.log()`, которые могут присутствовать на вашем сайте. Отлаживаемые устройства перечисляются на панели устройств, которая на рис. 1.4 находится слева. Журнал консоли находится внизу.

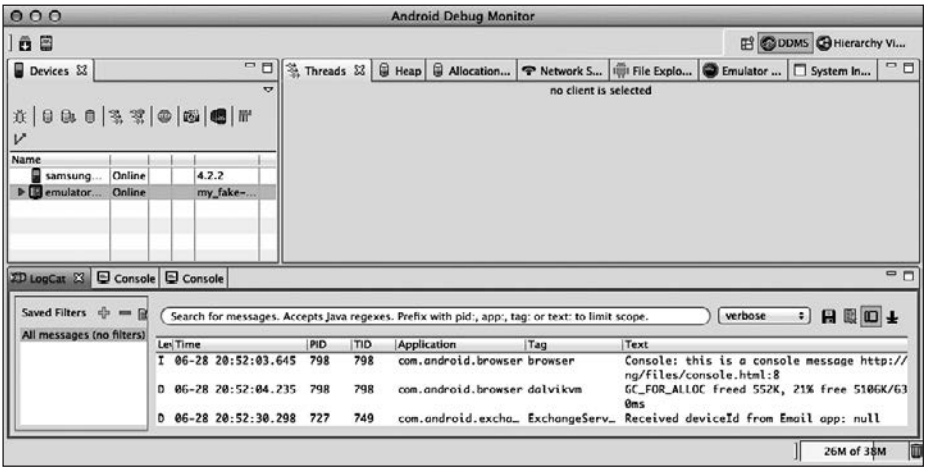


Рис. 1.4. Отладочный монитор Android

Открыв монитор, вы найдете в меню Window (Окно) диспетчер виртуальных устройств Android (Android Virtual Device Manager) (рис. 1.5). В этом окне вы можете создавать новые устройства-эмуляторы для тестирования и запускать их, как показано на рис. 1.3.

weinre

Это отладочный инструмент, полное название которого — web inspector remote — удаленный веб-инспектор. weinre — мощный отладочный инструмент, позволяющий проверять и исправлять код JavaScript, HTML и CSS. Он разрабатывается в рамках

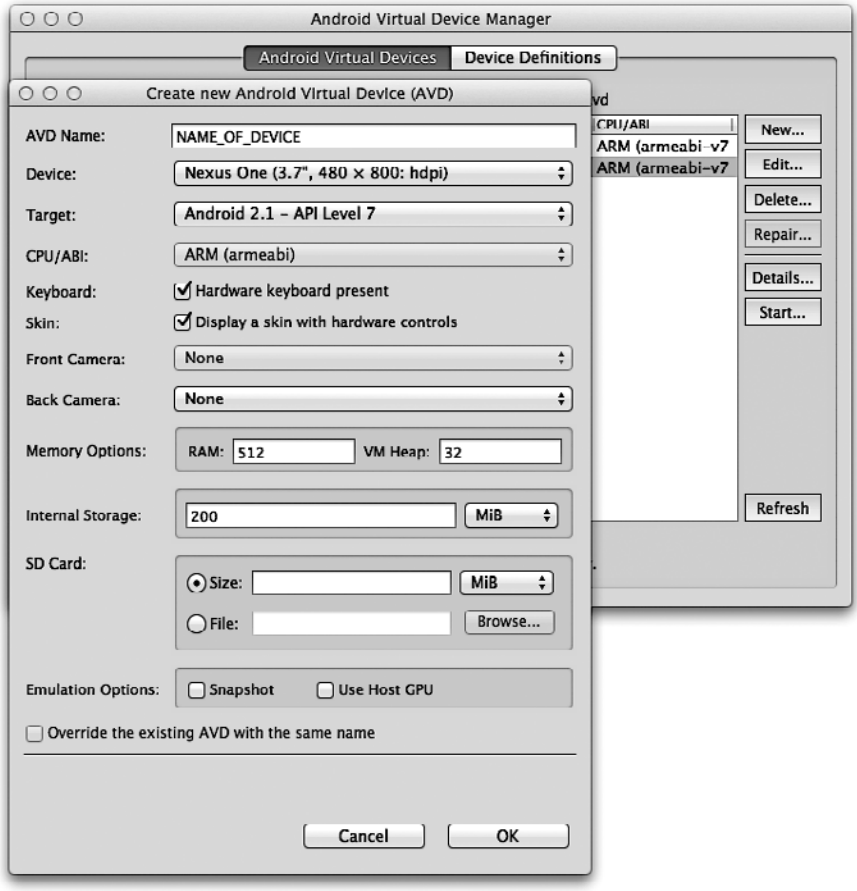


Рис. 1.5. Диспетчер виртуальных устройств Android обеспечивает создание эмуляторов, имитирующих работу ограниченного количества доступных для выбора устройств. Кроме того, он позволяет задавать практически неограниченное количество независимо определяемых конфигураций

проекта PhoneGap, вы можете использовать его на локальной машине либо непосредственно на сайте debug.phonegap.com. Кроме того, на основе weinre работает инструмент Adobe Edge Inspect, рассмотренный в одном из следующих разделов.

weinre — это удаленный отладчик, позволяющий подключать окно мобильного браузера, с которым вы сейчас работаете, к облегченной версии удаленного инспектора WebKit. В настоящее время weinre активно использует технологии Node.js и WebSockets[1].

На момент написания этой книги weinre обладает довольно неширокими возможностями. Этот отладчик позволяет в реальном времени просматривать состояние объектной модели документа, а также обеспечивает доступ к консоли JavaScript,

[1] Изначально этот инструмент был основан на Java. До WebSockets в нем использовались технологии CORS, JSON и XHR.

но в нем отсутствуют контрольные точки и не прослеживаются стековые следы. Как и следовало ожидать, в консоли JavaScript составляется список ошибок, поэтому отладка получается довольно сложной, но осуществимой.

Использование weinre

Инструмент weinre можно установить с помощью кода Java или JavaScript. Чтобы сделать это с использованием JavaScript, сначала скачайте и установите Node.js, в частности npm, диспетчер пакетов Node. Для установки введите в командной строке:

```
npm -g install weinre
```

Теперь можно запустить weinre, просто записав в командной строке:

```
weinre
```

По умолчанию сервер weinre работает на хосте `localhost:8080`, пока не будет остановлен командой Control+C, либо не будет перезагружен компьютер, либо серверный процесс не завершится каким-либо другим способом.

Чтобы начать отладку приложения, добавьте в него сценарий weinre:

```
<script src="http://localhost:8080/target/target-script-min.js#anonymous">
</script>
```

В любом браузере WebKit на ПК можно ввести адрес http://localhost:8080/client/#anonymous, чтобы получить доступ к отладчику. Инспектор отобразит полноэкранное окно браузера, очень напоминающее панель Инструменты разработчика в Chrome, однако с меньшим количеством вкладок и ограниченным функционалом.

На вкладке Remote (Удаленная работа) вы найдете актуальный список окон тех мобильных браузеров, которые работают в той же сети, что и ваш сценарий weinre. Вкладки Elements (Элементы), Resources (Ресурсы), Network (Сеть), Timeline (Хронология) и Console (Консоль) (рис. 1.6) напоминают вкладки веб-инспектора для ПК. Как видите, в этом облегченном инструменте отсутствуют вкладки Sources (Исходники), Profiles (Профили) и Audit (Аудит), но в будущем могут появиться и они.

Adobe Edge Inspect и Ghostlab

Отладка в Adobe Edge Inspect строится примерно по тому же принципу, что и в weinre. Это делается, чтобы упростить процесс ранней отладки (на этапе проектирования), практически полностью автоматизировав все предшествующие ему шаги. Adobe Edge Inspect построен именно на основе weinre. Такое упрощение достигается благодаря выполнению следующих задач: запуска сервера, ввода URL в строку браузера и добавления сценариев в разметку.

Adobe Edge Inspect потребуется установить на всех удаленных устройствах, где вы собираетесь заниматься отладкой, а также на ПК — как расширение браузера Chrome. Когда ПК и тестируемое устройство находятся в одной сети, вы можете создать соединение между компьютером и устройством.

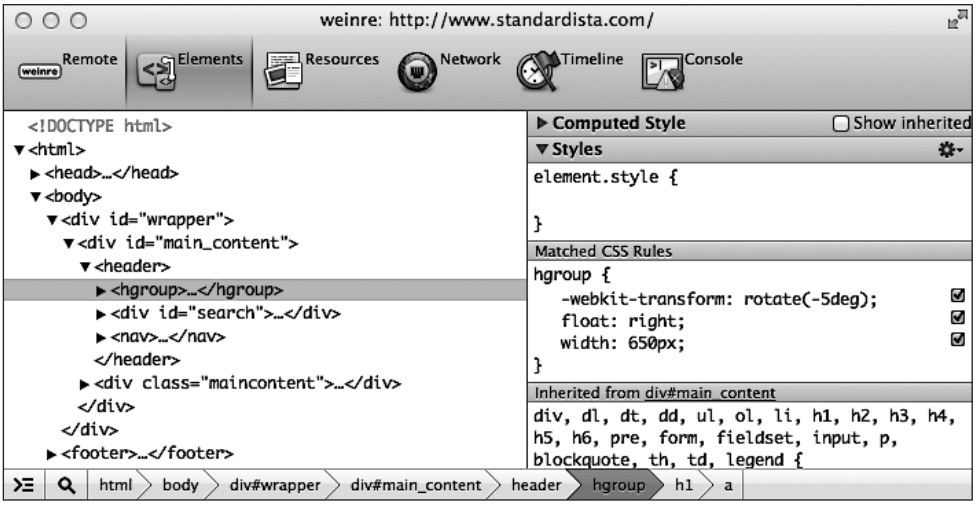


Рис. 1.6. Отладчик weinre

При открытии Edge на мобильном устройстве отладчик затребует у вас пароль к этому устройству. Пароль необходимо ввести на ПК в браузерное расширение Edge. Чтобы включить Edge в браузере на ПК, сначала необходимо открыть приложение и войти под своими учетными данными в Adobe.

Выполнив вход, щелкните кнопкой мыши на ярлыке браузерного расширения Edge (рис. 1.7). Так вы прикажете браузеру начать поиск устройств в сети. Когда он найдет ваше устройство, введите пароль к этому устройству в окно Edge.

Пароль гарантирует, что вы даете разрешение на обмен информацией между ПК и мобильным устройством. Вы не допускаете, чтобы посторонние компьютеры могли получить контроль над вашим устройством, а ваш компьютер, в свою очередь, не может получить контроль над телефонами других людей.

Как только будет установлено соединение между вашим компьютером и одним или несколькими устройствами, вы сможете одновременно отслеживать загрузку веб-страниц во всех их мобильных браузерах. Вкладка отладчика, которая в данный момент открыта в Chrome, будет получена и отображена на всех мобильных устройствах, подключенных к ПК через Edge Inspect.

Чтобы выполнить отладку веб-страницы с устройства, перейдите на интересующую вас страницу в браузере Chrome на ПК либо откройте ее на устройстве. Сначала щелкните на меню расширения Chrome Adobe Edge Inspect, а потом на символе <> рядом с обозначением устройства, отладкой которого хотите заняться. Отладчик weinre запустится на вашей локальной машине, а устройство и веб-страница будут доступны по активной ссылке на вкладке Remote (Удаленная работа) в weinre. На рис. 1.6 эта вкладка находится рядом с левым краем.

В бесплатной версии Adobe Edge Inspect можно выполнять отладку всего на одном, а не на нескольких устройствах. Чтобы одновременно контролировать все ваши устройства, достаточно оформить месячную подписку на эту программу. Кроме того, платная версия позволяет делать скриншоты.

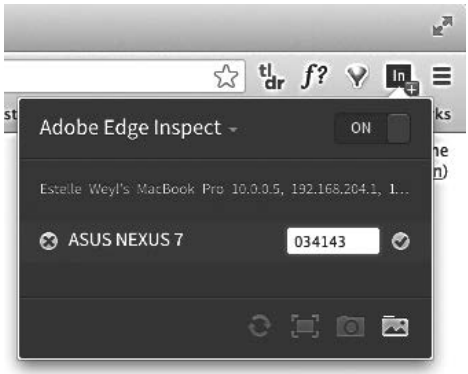


Рис. 1.7. Adobe Edge Inspect связывает Nexus 7 и Google Chrome для отладки

Если вы работаете с Mac и хотите протестировать несколько устройств, то можете получить аналогичный функционал, приобретя программу Ghostlab. Выбирая, какую из двух программ установить, выгоднее остановиться на Ghostlab: в этой программе требуется единовременный взнос, а ежемесячная абонентская плата за Adobe Edge может обойтись дороже.

Отладка JavaScript с применением Aardwolf

Если вас в первую очередь интересует отладка JavaScript, то можете попробовать Aardwolf. Это инструмент для удаленной отладки кода JavaScript, позволяющий выполнять и перехватывать код на этом языке. Aardwolf работает следующим образом: он переписывает ваш код на сервере и добавляет отладочные ловушки. Подобно weinre (как вы помните, в качестве машинного интерфейса weinre использует `Node.js`), Aardwolf применяет синхронные вызовы XHR для останова в контрольных точках. Aardwolf можно задействовать для удаленной пошаговой проработки кода. Инструмент поддерживает отслеживающие объекты, контрольные точки и стеки вызовов.

Отладчик BlackBerry 10

Конечно, weinre очень впечатляет, но в BlackBerry 10 вы найдете еще более мощный отладчик.

Обозреватель BlackBerry (BlackBerry Browser), подобно weinre, использует для предоставляения функций веб-инспектора клиент-серверную архитектуру. Но BlackBerry Browser значительно отличается от weinre по причине того, что этот инструмент действует как веб-сервер. Он выдает веб-страницы по протоколу HTTP, используя для этого USB-порт или соединение Wi-Fi. Вы удаленно просматриваете содержимое в браузере на ПК. Если в одной сети Wi-Fi работают несколько браузеров на базе WebKit, то в любом из них вы можете перейти на IP-адрес или на тот порт, на котором работает обозреватель BlackBerry (для перехода на порт нужно указать его номер). После этого можно приступать к проверке кода.

Для использования инспектора нужно активизировать отладку в параметрах обозревателя BlackBerry. Когда веб-инспектор включен, браузер или приложение отображает IP-адрес и номер порта, на который будет отсылаться содержимое.

Чтобы включить веб-инспектор в BlackBerry 10 из браузерного приложения, сделайте жест смахивания в направлении от верхнего края экрана. В результате отобразится панель меню браузера. Щелкните на значке настроек, а затем выберите Developer Tools (Инструменты разработчика). Там вы сможете включить веб-инспектор. Если работаете с планшетом, то найдете эту функцию в разделе Options ▸ Privacy & Secrity (Параметры ▸ Личные данные и безопасность). В браузере отобразятся IP-адрес и номер порта. Эта информация нужна для установки связи между устройством и вашим браузером на ПК. Когда откроется строка приглашения, введите в нее пароль к вашему устройству, чтобы завершить процесс активизации. Нажмите Back (Назад), чтобы сохранить сделанные изменения и вернуться в окно браузера. Теперь вы можете открыть соединение с обозревателем BlackBerry и удаленно отслеживать процесс отображения страниц.

Инструменты тестирования

При тестировании лучше всего испытывать ваши мобильные ресурсы на реальных устройствах. Но таких устройств уже тысячи, и попросту невозможно протестировать сайт на всех этих гаджетах, не говоря уже о том, что список таких устройств постоянно пополняется. Поэтому рекомендуется выполнять тестирование на репрезентативной выборке устройств. В этой группе должны быть представлены устройства с различными сочетаниями операционных систем и браузеров. Аппараты должны иметь разные размеры и разные функциональные возможности, такие как разрешение экрана, доступная память и ширина полосы сетевого соединения.

Тестирование ресурса на реальных устройствах может потребовать немало времени и вообще является дорогостоящим процессом. Поэтому, кроме отладочных инструментов, рассмотренных в предыдущем разделе, в нашем распоряжении есть и ряд программ, помогающих обеспечить максимально эффективное тестирование.

Эмуляторы и симуляторы

Эмулятор — это программа, которая копирует (эмулирует) функции мобильного устройства (или нескольких устройств) на ПК. Эмулируемое поведение с высокой точностью воспроизводит функционирование реального устройства. Разница между эмуляцией и симуляцией заключается в том, что эмулятор максимально точно воспроизводит работу устройства. При симуляции создается абстрактная модель имитируемой мобильной операционной системы.

Эмуляторы позволяют использовать на ПК программы, написанные для мобильных устройств. В эмуляторе можно запускать и отлаживать код, даже если у вас нет нужного устройства. Но даже если вы тестируете код в симуляторах и эмуляторах, невозможно охватить таким образом все разнообразные устройства. Эмуляторы

и симуляторы просто помогают начать и ускорить и работу и отладку. Все равно придется тестировать программы на самых разных мобильных устройствах.

Когда вы запускаете сайт в симуляторе, он работает в специальном приложении для ПК. Некоторые симуляторы написаны для имитации конкретных устройств, другие позволяют выбрать, какое устройство вы собираетесь имитировать. Например, симулятор iOS способен имитировать работу как iPhone, так и iPad. С помощью специальных меню можно менять ориентацию с книжной на альбомную и обратно. В симуляторе есть виртуальные кнопки, имитирующие кнопки устройства. А если устройство не оборудовано сенсорным экраном, то события касания можно имитировать с помощью мыши.

Симулятор неточно воспроизводит работу аппаратного обеспечения устройства, поэтому нет гарантии, что ваше приложение будет работать на устройстве точно так же, как и на симуляторе. Есть определенные библиотеки, которые будут отлично компилироваться и связываться при работе на симуляторе (поскольку на самом деле все происходит на ПК), а вот на устройстве компилироваться не станут.

Как правило, в состав эмуляторов и симуляторов входит полнофункциональный SDK для тестирования приложений в сымитированной нативной среде. Для тестирования нашего кода понадобятся такие эмуляторы и симуляторы, в которых есть браузер — а он есть в любом таком инструменте. Вероятно, потребуется загрузить ваш сайт на следующие эмуляторы и симуляторы и протестировать его в этих инструментах.

- **Эмулятор Android.** Бесплатный эмулятор Android для операционных систем Windows, Mac OS X и Linux. К эмулятору прилагается SDK. Скачать эмулятор и SDK можно по адресу http://developer.android.com. Как описано в разделе «Отладочный инструментарий Android», сначала скачайте базовый SDK, а затем отдельно каждую версию операционной системы Android. В загруженных материалах вы найдете командную строку терминала Android для работы в Mac/Linux и приложение Setup.exe для установки SDK в Windows.

  Эмулятор Android позволяет ограничить память виртуального устройства для более точного воспроизведения характеристик смартфона. В диспетчере виртуальных устройств Android (Android Virtual Device Manager) выберите интересующее вас устройство и нажмите Edit (Редактировать) (см. рис. 1.5). Чтобы сымитировать аппаратное обеспечение, щелкните New (Создать) и выберите объем оперативной памяти устройства (Device RAM) из раскрывающегося меню Property (Свойства).

- **Симулятор iOS.** Доступен только в операционной системе Mac OS X. Это бесплатная имитационная среда, включающая, в частности, браузер Mobile Safari. Однако учтите, что размер iPhone SDK составляет около 2 Гбайт, поэтому на его скачивание может уйти немало времени.

  Это именно симулятор, а не эмулятор. В нем нет никакой эмуляции аппаратного обеспечения или индикаторов производительности. Эта программа позволяет оценить, как работает ваш код и как отображается сайт, но в целом не подходит для измерения производительности сайта.

Если вы просто хотите посмотреть, как выглядит плод вашего труда, без эмуляции и симуляции, обратите внимание на инструменты iPhoney и iPadPeek и им подобные. На этих ресурсах можно открыть сайт в браузере, изображение получится как в одной из старых моделей устройства.

- **Симулятор BlackBerry.** В состав симуляторов BlackBerry для операционной системы Windows входят прокси-сервер, плагины для Eclipse и Visual Studio (они понадобятся веб-разработчикам) и, собственно, сами симуляторы.
- **Эмулятор Windows Phone.** Доступен только на компьютерах с операционной системой Windows. Эмулятор Windows Phone — это приложение для ПК, эмулирующее работу устройства с Windows Phone. Вы можете скачать комплект для разработки в Windows Phone (Windows Phone SDK) по адресу http://dev.windowsphone.com/en-us/downloadsdk. Наиболее актуальная версия и информация об установке находятся по адресу http://www.microsoft.com/en-us/download/details.aspx?id=35471.

  В настоящее время в Visual Studio по умолчанию применяется эмулятор WVGA 512 Мбайт. Он воспроизводит работу смартфона с Windows Phone 8, память которого весьма ограниченна.
- **Симулятор Firefox OS.** Является расширением для браузера Firefox. Он имитирует окружение, типичное для Firefox OS, его использование напоминает работу со смартфоном. Установив расширение, перейдите в браузере Firefox для ПК в раздел Web Developer ▸ Firefox OS Simulator (Веб-разработка ▸ Симулятор Firefox OS).
- **Эмулятор Opera Mobile.** Эмулятор Opera Mobile для операционных систем Windows, Mac и Linux можно скачать по адресу http://www.opera.com/ru/developer.
- **Симулятор Opera Mini.** Полнофункциональное приложение Opera Mini, соответствующее актуальной версии этого браузера, является апплетом Java и доступно по адресу http://www.opera.com/ru/mobile.

Это наиболее распространенные операционные системы. Для большинства мобильных операционных систем, в частности Symbian и WebOS, существуют такие комплекты SDK, которые вы можете скачать на ПК. В них можно имитировать нужную мобильную среду. В зависимости от целевого рынка вы должны тестировать свой код во всех операционных системах, которые могут быть установлены на устройствах ваших пользователей. Более подробный список эмуляторов приведен на сайте http://www.mobilexweb.com/emulators.

Онлайн-инструменты

Чтобы быстро оценить важнейшие статистические показатели вашего устройства, влияющие на основные медиазапросы, откройте в браузере устройства сайт http://www.quirksmode.org/m/tests/widthtest.html.

Инструмент W3C mobileOK Checker проверяет, насколько полно при программировании вашего сайта учитывались рекомендуемые методы. Здесь же вы найдете

информацию о том, как более качественно адаптировать ваш сайт для работы на мобильных устройствах, а также полезные ссылки. MobiReady — специальный онлайновый инструмент, оптимизирующий работу с W3C MobileOK Checker. Результаты в нем отображаются так, что сайт мотивирует вас действовать более эффективно и действительно приближать ваш сайт к проверенным мобильным стандартам.

Расширение для Firefox, называемое Modify Headers, пригодится для мобильной веб-разработки, HTTP-тестирования и обеспечения конфиденциальности. Этот инструмент позволяет изменять заголовки HTTP-запросов, отсылаемые на сервер, — добавлять в них информацию, заменять ее и фильтровать.

Смартфоны

Тестирование на реальных устройствах — важнейший этап в ходе разработки. Но приобрести целую кучу смартфонов будет накладно. Намеренно изменяя размер окна браузера и пользуясь эмуляторами, вы не сможете судить о производительности сайта на реальном устройстве, возможностях устройства, плотности пикселов, а также влиянии мобильной сети на работу смартфона.

Если вы пишете нативные приложения, то, несомненно, вам нужны устройства с теми операционными системами, для которых вы программируете. Но в этой книге речь идет не о нативной разработке, а о написании сайтов и приложений на языках HTML5, CSS3 и JavaScript. Поэтому наш код будет работать в браузерах на любых устройствах. Хотя мы и пишем код, который будет работать в браузере, нам не обойтись без тестирования на многих устройствах, а также в сетях нескольких мобильных операторов. Всегда тестируйте код на реальных устройствах с реалистичным соединением. Учитывайте, как код работает в точках доступа Wi-Fi, при соединении 3G, 4G и даже EDGE. Прокатитесь на автобусе или поезде, попытайтесь выйти на ваш мобильный сайт из различных точек, перемещаясь по городу, в пригороде и в сельской местности.

Браузерные лаборатории

Тестирование на реальных мобильных устройствах — это часть процесса разработки, которой нельзя пренебрегать. Существует немало браузерных лабораторий, попробуйте найти какую-нибудь поблизости от вас. Если таких лабораторий не окажется, соберитесь с коллегами и сами создайте ее.

Если вы предпочитаете работать в собственной лаборатории по тестированию устройств, то вам нужны аппараты с разными размерами экрана, операционными системами, наборами возможностей и, конечно, браузерами. Организовать собственную лабораторию по тестированию устройств, которая давала бы достаточно полное представление о технических возможностях современного мобильного рынка, не так уж дорого. Невозможно приобрести все виды устройств, но вы должны испытывать аппараты с разными размерами экранов, браузерами и операционными системами.

Существуют и виртуальные лаборатории для тестирования устройств — например, DeviceAnywhere и Nokia Remote Access. Там используются реальные устройства, к которым открыт удаленный доступ. Поскольку речь идет о реальных устрой-

ствах, естественно, их количество ограниченно. Если все они в данный момент заняты, вам придется дождаться своей очереди.

iOS

Если у вас еще нет устройства с операционной системой iOS, а целевая аудитория вашего приложения не сосредоточена в бедных странах третьего мира, не поскупитесь и приобретите такое устройство.

Еще лучше купить устройство с новейшей версией операционной системы iOS и довольно старой версией. Старое устройство можно приобрести на интернет-барахолке, например Craigslist или eBay, за символическую сумму. В настоящее время около 1,8% пользователей iOS (это соответствует 0,13% пользователей Интернета) используют операционную систему iOS 4.3 или ниже, а 12,5% пользователей iOS (0,93% пользователей Интернета) работают с iOS 5.

Приобретая устройство, мы руководствуемся только одним соображением: на нем должен быть рабочий браузер. Если бюджет очень ограничен, покупайте устройства с треснутым экраном — они стоят копейки. Обязательно нужен один телефон, желательно также приобрести другой телефон, iPad или iPod touch.

Итак, когда у вас будет устройство для тестирования, скачайте на него браузер Opera Mini. На iTunes он предоставляется бесплатно.

Если на всех ваших устройствах iOS установлены дисплеи с высоким разрешением, обязательно добудьте и устройство с обычным дисплеем. Кроме того, желательно не ограничиваться смартфонами и провести тестирование также на одном-двух планшетах.

Android

Android — самая популярная в мире операционная система, которая присутствует на рынке в виде самых разнообразных версий. Операционная система Android применяется на множестве устройств — как на смартфонах, так и на планшетах. Приобретите как минимум два (а лучше — больше) устройства с Android. В идеале это должны быть мощный довольно новый смартфон с одной из последних версий операционной системы и подержанный телефон с одной из старых версий операционной системы. На момент написания этой книги ОС Android 2.3 уже считалась архаичной, но она по-прежнему широко распространена на дешевых телефонах. Это самая популярная версия Android. В то время с ней работали 34% всех пользователей Android, что составляет 2,3% всех пользователей Интернета[1].

Приобретите для тестирования устройства не только с различными версиями операционной системы Android — постарайтесь, чтобы это были гаджеты разного размера, с разными процессорной мощностью и разрешением экрана, а также чтобы они были изготовлены разными производителями. На устройстве с Android можно устанавливать разные браузеры, в частности Chrome, Opera Mini, Opera Mobile, Firefox Mobile, Dolphin Mini и Dolphin HD.

---

[1] На момент перевода соотношение изменилось (см. http://developer.android.com/about/dashboards/index.html). — *Примеч. пер.*

Windows

Если вы хотите приобрести мобильное устройство с операционной системой Windows, то это должна быть операционная система как можно более новой версии. Windows Phone 7 так и не приобрела значительную популярность, но перспективы Windows Phone 8 кажутся более радужными. Обе эти операционные системы имеют пользовательский интерфейс в стиле Metro. Кроме тестирования вашего приложения (так вы проверите, правильно ли функционирует разметка), поэкспериментируйте с самим смартфоном от Windows. Практика взаимодействия пользователя с устройством существенно отличается от приемов, привычных по работе с Android и iOS. Возможно, вам захочется откорректировать некоторые взаимодействия в коде пользовательского интерфейса, чтобы они лучше соответствовали стандартному поведению, разработанному для устройств с Windows.

BlackBerry

Конечно, на устройстве с BlackBerry 10 имеется самый лучший отладчик из всех, что встречаются на мобильных смартфонах, но вот пользовательская аудитория этой операционной системы весьма невелика.

На рынке присутствует значительно больше старых устройств с BlackBerry, чем устройств, на которых установлена система BlackBerry 10. Рекомендую приобрести смартфон с BB6 или BB7. К счастью, эти старые устройства сравнительно недороги, а их экраны — несенсорные, ведь нам понадобится тестировать наши сайты и на таких экранах.

Браузер BlackBerry ниже версии 6 не был основан на WebKit. Этими старыми, очень старыми устройствами почти никто не пользуется. Если вы полагаете, что в вашем целевом сегменте рынка окажется существенное количество пользователей с BB5 и ниже, то можете приобрести и третье устройство с BlackBerry.

Nokia

В этом разделе речь пойдет об операционной системе Symbian, а не о Lumia Windows Phone.

В некоторых странах устройства с Symbian, Series 40, Samsung и в меньшей степени Sony Mobile и Motorolla распространены шире, чем Android, iOS, BlackBerry и Windows. Если бы я предложила купить одно из подобных устройств, то к моменту выхода книги в печать оно бы уже устарело. Просто необходимо учитывать, что в международных масштабах Nokia является сильным игроком мобильного рынка, ее операционная система Symbian довольно широко распространена. Рекомендую приобрести сотовый телефон Nokia с вводом через джойстик и небольшим экраном. Так вы получите реалистичное впечатление о том, в каком крошечном окошечке очень многие пользователи во всем мире будут просматривать ваш сайт.

Kindle

Не следует забывать и о Kindle. На этих устройствах установлен браузер Silk, основанный на WebKit.

WebOS

Устройства с операционной системой WebOS уже не производятся, но по-прежнему используются. Гаджет PalmPre или Pixi обойдется вам меньше чем в $30.

Автоматизированное тестирование

Инструменты для тестирования, перечисленные ранее, пригодятся лишь для ручного тестирования и визуального отслеживания результатов. Но чтобы по-настоящему полно протестировать сайт или приложение, требуется вращать, масштабировать, панорамировать экран, щелкать на нем кнопкой мыши и взвывать от досады. Проверяя внешний вид созданной страницы, вы должны просматривать ее на устройствах с разными размерами экрана, браузерами и операционными системами. Если мы тестируем только статический контент, нам может вполне хватить такого инструмента, как Adobe Edge.

Но при работе над веб-приложениями целесообразно применить автоматизированное тестирование. Необходимо непрерывно тестировать ваше приложение на протяжении довольно долгого времени, чтобы убедиться, что код действительно работает. Важно протестировать все потенциально возможные события и их результаты. Есть несколько специальных тестировочных библиотек JavaScript.

Jasmine — это фреймворк для разработки на JavaScript, работающий через реализацию поведений. PhantomJS — это консольный браузер, то есть не имеющий графического интерфейса, построенный на основе WebKit[1]. Таким образом, PhantomJS не является библиотекой для тестирования и обеспечивает нативную поддержку для различных веб-стандартов. В частности, он обеспечивает обработку DOM (объектной модели документа), работает с CSS-селекторами и нотацией JSON. На сайте PhantomJS вы можете скачать заготовленный двоичный файл этого браузера для любой операционной системы.

Чтобы максимально эффективно использовать PhantomJS для автоматизированных тестов, скачайте также утилиту CasperJS. Для имитации AJAX-вызовов можно воспользоваться Sinon.JS. На всех перечисленных сайтах предоставляется качественная документация, позволяющая в кратчайшие сроки научиться работать с этими библиотеками при тестировании в WebKit. Правда, наличие этих инструментов не отменяет необходимости тестирования на мобильных устройствах.

Существуют и онлайновые инструменты для тестирования. Некоторые ресурсы, например SauceLabs, позволяют тестировать сайты и приложения в сотнях сочетаний мобильных и настольных платформ, браузеров и операционных систем.

Выбирайте то, что оптимально подходит для ваших приложений, но всегда тестируйте.

А теперь перейдем к написанию кода, чтобы нам было что тестировать.

---

[1] Такие инструменты называют еще «безголовыми» браузерами. — *Примеч. пер.*

2 Как усовершенствоваться до HTML5

Чтобы изучить язык HTML, достаточно нескольких часов. На то, чтобы по-настоящему освоить его, требуются годы практики и участия в профессиональных дискуссиях. Да, практически все программисты, дизайнеры, а также очень многие студенты заявляют, что знают HTML, но они, скорее всего, знают лишь некоторые элементы (теги) HTML, а возможно, и используют их неправильно.

В этой главе будут рассмотрены многие секционирующие элементы HTML5. Дочитав эту главу, вы будете очень хорошо понимать семантику HTML5. Конечно, в трех главах я не смогу научить вас всем тонкостям HTML. Я и сама до сих пор продолжаю изучать HTML, и не только потому что спецификация HTML5 пока не завершена. Она далека от завершения, в ней наверняка будет сделан еще ряд изменений. Но и об этом не беспокойтесь. Вероятно, все функции HTML5, уже реализованные в браузерах, будут работать, как и сейчас, — может быть, изменятся какие-то нюансы.

Надеюсь, что в этой главе вы не только познакомитесь с различными элементами, их атрибутами, семантикой и назначением, но и осознаете, что возможности HTML гораздо шире, чем вам казалось ранее. Чем больше вы изучаете HTML, тем явственнее осознаете, как много еще можно выучить.

Мы вкратце рассмотрим элементы. О каждом элементе HTML можно написать целую главу, но в нескольких довольно объемных главах нам предстоит рассмотреть массу теоретического материала. Мы изучим каждый элемент настолько подробно, насколько требуется для уверенного его использования. Кроме того, вы будете представлять, что еще стоит выучить об этом элементе.

Первым делом давайте усвоим, что в названии этого языка нет пробела: именно HTML5, а не HTML 5.

Видите? Вы уже кое-что узнали! Первый шаг сделан, в путь!

Синтаксис HTML5

Язык HTML5 очень похож на HTML 4 и XHTML. Большинство элементов, поддерживаемых в этих версиях языка, поддерживаются и в HTML5. Удалены лишь нежелательные теги и атрибуты. Как правило, если ваш документ успешно проходил валидацию как строгий HTML 4 или XHTML, то он будет валиден

и в HTML5[1]. Синтаксис HTML и XHTML немного различается, но поддерживаются оба варианта. Просто измените тип документа (`doctype`), в котором есть указание на HTML 4.01 или XHTML, на `<!DOCTYPE html>` — и документ будет успешно проходить валидацию по правилам HTML5. Подробнее о типе документа мы поговорим в дальнейшем.

HTML5 по сравнению с HTML 4 и XHTML более совершенен. Он включает в себя многие старые элементы, но часть нежелательных элементов из него удалена, добавлены некоторые новые элементы, а какие-то элементы переопределены или немного доработаны.

Авторы спецификаций HTML5 внимательно рассмотрели, чем сейчас занимаются веб-разработчики: какие разделы веб-документа встречаются на большинстве сайтов, какие классы и ID присваиваются этим компонентам, какими сценариями наиболее активно пользуются авторы сайтов, какие библиотечные возможности распространились настолько, что уже считаются общими местами.

HTML5 стремится централизованно выполнять те функции, которые многие разработчики до недавних пор реализовывали самостоятельно. Создается стандарт, подробно описывается, как браузеры должны поддерживать его и как должны справляться с разметкой тех разработчиков, чей код, мягко говоря, не вполне соответствует стандарту. В спецификациях HTML5 подробно описано, как браузеры должны обрабатывать (интерпретировать) любой фрагмент корректного или некорректного кода. Такое внимание к мелочам обусловлено тем, что авторы HTML5 стремятся обеспечить правильную работу браузеров при любых возможных сценариях. Все браузеры должны выстраивать из конкретного образца разметки одну и ту же модель DOM, а разработчики не должны впустую растрачивать полосу доступа в Интернет, пытаясь справиться с межбраузерными различиями.

Лично мне хотелось бы, чтобы стандарты были строже. Я считаю, что дело не в браузерах, которые сегодня должны пусть и со скрипом, но интерпретировать плохой код. Дело в разработчиках, которым следует писать более качественный код. Вы читаете эту книгу, из чего я делаю вывод, что вы причисляете себя к «правильным» разработчикам. Хорошо, ведь этому мы и собираемся научиться.

Элементы

Веб-страница состоит из ряда элементов. Некоторые элементы пусты, другие содержат текст, а третьи включают в себя другие элементы (либо и элементы и текст). Большинство элементов могут содержать дочерние элементы либо текстовые узлы. Те элементы, которые не могут содержать дочерних (таковы, например, элементы с изображениями и `meta`-элементы), называются *пустыми*.

Элемент — это конструкт, состоящий из открывающего тега и каких-то опциональных (а иногда и обязательных) атрибутов (рис. 2.1). Как правило, элемент содержит какой-либо контент, а в конце его стоит закрывающий тег. Если вы

[1] Об изменениях в составе элементов и атрибутов см. http://www.w3.org/TR/html5-diff/. Здесь же перечислены устаревшие элементы и атрибуты.

пишете в XHTML-стиле, то элемент может сопровождаться необязательным прямым слешем. Слеш нужен для самозакрытия тега у пустых элементов, в частности <img> или <input>.

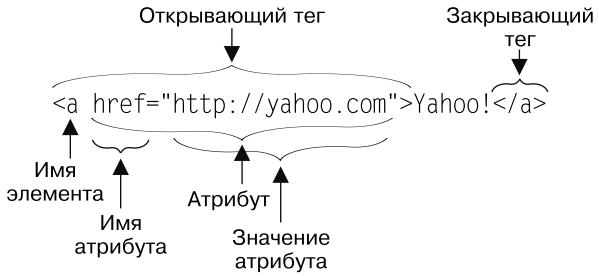


Рис. 2.1. Составные части элемента

В предыдущих версиях HTML строковые элементы могли содержать только другие строковые элементы и текст. Некоторые блочные элементы также могли содержать другие блочные элементы, строковые элементы и/или текст. Их вышестоящие элементы (предки) и нижестоящие элементы (потомки) состоят из элементов, атрибутов и текста.

ПРИМЕЧАНИЕ

С помощью CSS можно изменить внешний вид любого фразового элемента, чтобы он отображался как блок, либо принудительно оформить блочный или секционирующий элемент как строковый. (Обратите внимание: я по-прежнему использую термин «строковый». В HTML5 этот термин означает вариант представления элемента, но не его тип.)

В HTML5 мы распрощались с противопоставлением «строковые элементы — блочные элементы». Такое соглашение об именованиях было связано с представлением элементов на экране. В HTML5 элементы подразделяются на секционирующие, заголовочные, фразовые, встроенные, потоковые, элементы метаданных и интерактивные элементы. Но с семантической точки зрения некоторые идеи совершенно не изменились. Например, секционирующие элементы не должны располагаться внутри фразовых.

ПРИМЕЧАНИЕ

При выборе элемента опирайтесь в первую очередь на его семантику, а не на то, как этот элемент по умолчанию отображается в браузере. Вы действительно можете использовать любой элемент для любых оформительских целей, но так делать не следует. Каждый элемент обладает собственной семантикой.

Атрибуты

Все элементы могут иметь атрибуты. У некоторых элементов есть обязательные атрибуты. В качестве примера атрибута рассмотрим href. Он сопровождает элемент <a> (см. рис. 2.1). Как правило, атрибуты представляют собой пары «имя/значение»,

причем у логических атрибутов значение может отсутствовать[1]. Атрибуты предоставляют графическим движкам дополнительную информацию об элементе. Они ставятся у открывающего, а не у закрывающего тега.

Существует ряд атрибутов, которые являются глобальными практически для всех HTML-элементов. Во-первых, это собственно глобальные (основные) атрибуты, во-вторых — атрибуты интернационализации, о которых мы поговорим в следующем разделе. Другие атрибуты являются более специфичными для отдельных элементов. В главе 3 мы обсудим эти атрибуты, а также поговорим об элементах, с которыми они применяются.

Глобальные атрибуты и атрибуты интернационализации

В HTML5 появилось несколько основных атрибутов и атрибутов интернационализации, которые могут сопровождать практически любой элемент. Атрибуты `id`, `class`, `title`, `style`, `lang` и `dir` по-прежнему поддерживаются со всеми элементами. В HTML5 добавились атрибуты `accesskey`, `hidden` и `tabindex`. Также предлагается добавить к числу основных еще пять интерактивных атрибутов: `contenteditable`, `contextmenu`, `spellcheck`, `draggable` и `dropzone`. Эти атрибуты рассмотрены далее.

Наряду с глобальными атрибутами все элементы могут иметь атрибуты микроданных, роли WIA-ARIA, `aria`-атрибуты, а также специальные атрибуты данных. К последней категории относятся атрибуты, которые задаем мы сами. Они записываются как `data-*`, астериск (*) в данном случае означает, что этот атрибут создали мы сами. Префикс `data-` введен в HTML5 для того, чтобы разработчики могли сами создавать атрибуты и эти атрибуты не вызывали конфликтов при работе в последующих версиях HTML. Атрибутам `data-*` посвящен раздел «Специальные атрибуты данных с применением data-*» данной главы. Микроданные и атрибуты доступности (ARIA) описаны в главе 6.

id

Атрибут `id` — это уникальный идентификатор. В любом документе никакие два элемента не могут иметь одинаковый атрибут `id`, и у каждого элемента может быть только один атрибут `id`. В HTML5 значения атрибутов `id` должны состоять как минимум из одного символа и не содержать пробелов. В предыдущих версиях действовали другие правила. Атрибут `id` должен был начинаться с буквы в диапазоне A–Z или a–z, за этой буквой могли следовать другие буквы (A–Z, a–z), цифры (0–9), дефисы (-), нижние подчеркивания (_), двоеточия (:) или точки (.).

---

[1] Логическими называются такие атрибуты, которые могут находиться в одном из двух состояний: «истина» или «ложь». В XHTML примерами таких атрибутов являются readonly="readonly", checked="checked" и disabled="disabled". В HTML5 эти атрибуты можно (и нужно) записывать так: readonly, checked, disabled.

ПРИМЕЧАНИЕ

Рекомендую использовать в названиях идентификаторов только буквы и цифры. Однако названия атрибутов должны быть согласованными и соответствовать действующему соглашению об именованиях.

Можно и вовсе обойтись без атрибута `id`. Но он все же необходим при создании внутренних закладок на странице, а также реализации меток, явно предназначенных для элементов, находящихся внутри формы.

Эти метки задаются с помощью тега `<label>`, а их связь с элементами формы осуществляется посредством атрибута `for` этого тега и атрибута `id` элемента формы. Обратите внимание: атрибуты `id` могут отсутствовать у тех элементов форм, которые вложены в *неявную метку*. Мы подробнее поговорим о метках и об атрибуте `id` в главе 4.

Конечно, атрибут `id` полезен и в качестве якоря для работы с JavaScript, хотя им в таком качестве немного злоупотребляют. При включении в документ атрибутов `id` их также можно использовать с таблицами стилей CSS для точного указания нужных элементов. Однако атрибуты `id` — дорогой ресурс, так как каждый такой атрибут может использоваться в документе лишь единожды. Поэтому в каскадных таблицах стилей `id`-значения следует использовать экономно, хотя селекторы с указанием `id` действительно очень удобны в работе. Обо всем этом мы поговорим в главе 6. Пока что смею вас заверить: дочитав эту книгу, вы научитесь указывать на любой элемент на странице с помощью селекторов CSS3, совершенно не прибегая к `id`-селекторам.

class

`class` — это название классифицирующего признака либо нескольких таких признаков, которые свойственны определенному элементу. Атрибут `class` обозначает, что сопровождаемый им элемент относится к одному или нескольким классам. В отличие от атрибута `id`, один и тот же класс (или классы) можно присваивать сразу нескольким элементам. Элемент может иметь несколько атрибутов-классов, такие атрибуты разделяются пробелами.

В каскадной таблице стилей на языке CSS порядок следования имен классов в атрибуте `class` не имеет значения. Но их порядок в таблице стилей важен (об этом мы также поговорим в главе 6):

```
<a href="http://google.com" class="external popup search">Текст ссылки</a>
```

title

В атрибуте `title` представляется удобное для прочтения человеком описание любого элемента, к которому применяется этот атрибут. В браузерах с графическим интерфейсом атрибут `title` часто реализуется как всплывающая подсказка, но в других браузерах, особенно мобильных, атрибут `title` не отображается. Экранные дикторы могут поддерживать атрибут `title`, но в большинстве таких программ подобная поддержка по умолчанию отсутствует. Поэтому нельзя полагаться на атрибут `title` как на средство оптимизации доступности.

ПРИМЕЧАНИЕ

Существуют элементы, для которых атрибут title является обязательным. Таков, например, элемент <abbr>, у него значение атрибута title расшифровывает аббревиатуру. В большинстве других случаев атрибут title является опциональным.

Атрибут `title` очень полезен при работе со ссылками, с изображениями, фреймами и мультимедийными элементами. Он также довольно удобен для представления небольших фрагментов скрытой информации, которые выводятся на экран в ходе взаимодействия пользователя с сайтом. Например, с помощью CSS можно извлечь содержимое атрибута `title` и создать сгенерированный контент для всплывающих подсказок. При этом значение атрибута `title` указывается в качестве сгенерированного контента для псевдоэлемента `::before` или `::after`.

Конечно, можно пользоваться значениями атрибута `title` для реализации интересных эффектов. Но такие эффекты неудобны с точки зрения доступности. Поэтому, чтобы обеспечить последовательное улучшение сайта, лучше не полагаться на этот атрибут при сообщении важной информации. Поскольку пользователь может прочитать содержимое атрибута `title`, это содержимое всегда должно быть уместным и полезным. Если вы добавляете в `title` содержимое для облегчения веб-аналитики или записываете там код на JavaScript — больше так не делайте! Сегодня уже вполне можно отказаться от таких хитрых злоупотреблений атрибутами `rel` и `title` (да и давно стоило с этим покончить). Для подобных целей нужно использовать специальные атрибуты данных, которые описаны в подразделе «Специальные атрибуты данных с применением data-*» раздела «Что нового в HTML5: глобальная доступность и интерактивные атрибуты» данной главы.

style

Атрибут `style` позволяет внутри строки указывать правила оформления (стили) для конкретного экземпляра элемента. Этот элемент очень удобен при быстром прототипировании. В других случаях, не связанных с прототипированием, этот атрибут использовать не следует, так как веб-стандарты требуют отделять контент от представления!

Еще одно замечание: при просмотре кода посредством инспектора в таких программах, как Web Inspector для Safari или Chrome, Firebug для Firefox, DragonFly для Opera или F12 для IE, те стили, которые добавляются с помощью JavaScript или через интерфейс отладчика, будут отображаться внутри строки как значения атрибута `style`. Такой код генерируется динамически. Браузер может заниматься такой работой, а вам не следует.

lang

Атрибут `lang` — это один из двух глобальных атрибутов интернационализации (второй называется `dir`). С помощью атрибута `lang`, относящегося к элементу `<html>`, можно задавать основной язык веб-страницы. Для этой цели также можно использовать HTTP-заголовок `Content-Language` либо атрибут `httpequiv="language"`. Атрибут

lang используется для интернационализации или определения таких разделов содержимого, в которых применяется язык, отличающийся от заданного по умолчанию. Этот атрибут указывает язык, на котором будут записываться значения атрибутов элемента и его содержимое, в частности все элементы-потомки, для которых не указан собственный атрибут lang.

Атрибут lang позволяет поисковикам индексировать содержимое по языковому признаку. Кроме того, на основании этого атрибута в экранных дикторах применяются международные правила произношения.

Атрибут lang помогает также оформлять текст в зависимости от того языка, на котором он написан. Элемент <q> предназначен для отображения именно таких кавычек, которые соответствуют языку, указанному в атрибуте lang. Но эта возможность поддерживается не очень хорошо.

dir

Этот атрибут зачастую используется вместе с атрибутом lang. Он применяется для изменения направления текста на арабском, иврите и других языках, на которых текст записывается справа налево. По умолчанию атрибут dir имеет значение ltr (слева направо). Если содержимое вашей веб-страницы записано преимущественно на одном из этих ближневосточных языков, задайте направление текста в атрибуте dir элемента <html>.

Если в основном тексте (теле) страницы встречается текст, направление которого отличается от заданного по умолчанию, то направление текста можно изменить с помощью атрибута dir. Хотя это прямо и не требуется, я рекомендую сопровождать элемент атрибутами title и lang всякий раз, когда вы задаете для элемента атрибут dir. Как правило, атрибут dir применяется для отображения фрагментов текста справа налево (на соответствующих языках). Атрибут lang нужен для оптимизации работы поисковиков, экранных дикторов и других вспомогательных технологий. Так мы информируем программу о том, что язык изменился. Атрибут title помогает вам добавить перевод того или иного содержимого на основной язык, используемый на сайте. Эти атрибуты обеспечивают доступность вашего веб-контента не только для людей с ограниченными возможностями, но и для главного «подслеповатого» посетителя — Google!

Обратите внимание: в HTML5 атрибут dir немного изменился — у него появилось третье значение, auto. Таким образом, теперь этот атрибут может иметь значения rtl, ltr и auto.

Атрибуты HTML 4, вошедшие в число основных в HTML5

Вышеперечисленные атрибуты были глобальными (основными) в предыдущих версиях (X)HTML и остаются таковыми. Существуют еще два атрибута доступности, поддерживаемых с интерактивными элементами и в HTML5 ставших глобальными. Они рассмотрены в следующих разделах.

tabindex

Атрибут tabindex присутствовал в предыдущих спецификациях интерактивных элементов, например ссылок и элементов форм. Таким образом разработчик мог задавать порядок, в котором элементы этих типов получают фокус. В HTML5 сфера применения атрибута tabindex расширилась — теперь он может сопровождать любой HTML-элемент.

Многие пользователи переходят по страницам сайта в основном с помощью мыши. Они щелкают на ссылках и формах, чтобы взаимодействовать с интерактивными элементами. Другие, напротив, используют для навигации клавиатуру. Можно нажимать клавишу Tab, перемещая фокус с одного интерактивного элемента на другой. При работе с несенсорными устройствами пользователь может переходить между фокусируемыми элементами в четырех направлениях с помощью джойстика (крестовины). На смартфонах большинство пользователей просто трогают ссылки, которые хотят открыть, либо элементы форм, которые собираются заполнить. Как только ввод информации в форму завершен, на многих динамических клавиатурах появляется виртуальная кнопка Next (Далее) для перехода к следующему элементу формы. По умолчанию этот метод позволяет перемещать фокус только между формами или ссылками. Переводить фокус с актуального элемента на следующий можно с помощью правой кнопки джойстика, кнопки Next (Далее) либо клавиши Tab — так элементы упорядочиваются в исходном коде, если нативная последовательность элементов не переопределена в атрибуте tabindex.

В HTML5 в силу глобальности атрибута tabindex все элементы приобретают свойство *фокусируемости* — теперь оно распространяется не только на элементы форм и ссылки. В качестве значения этот атрибут получает целое число. Когда пользователь выполняет табуляцию, элементы с атрибутом tabindex будут получать фокус по порядку, в зависимости от числового значения каждого конкретного атрибута tabindex. Это касается атрибутов tabindex, имеющих положительное значение.

Не используйте tabindex с положительными значениями, если не можете предоставить по атрибуту tabindex для *каждого* интерактивного элемента на странице. Кроме того, если применен такой атрибут, должны быть веские причины, чтобы переупорядочить элементы. Наконец, вы должны быть уверены в том, что сможете поддерживать правильный порядок элементов на протяжении всего существования приложения. В противном случае разметка элементов на странице должна соответствовать стандартному порядку. Стандартный порядок элементов, задаваемый по умолчанию, соответствует их порядку в исходном коде. Изменив этот порядок, вы можете сильно запутать пользователей и значительно осложнить работу с сайтом. Целесообразно размечать страницу в наиболее логичном порядке и по возможности не использовать атрибут tabindex. В таком случае пользователь оптимально воспримет работу с сайтом.

Итак, если не рекомендуется менять порядок следования элементов на странице, почему же tabindex стала глобальной переменной? Это было сделано для того, чтобы можно было программно задавать фокус для любых элементов на странице,

в том числе неинтерактивных. Как правило, это делается с помощью JavaScript и/или клавиатурного фокуса.

Поскольку мы не собираемся изменять порядок получения фокуса в ходе табуляции по сравнению с тем порядком, что указан в исходном коде, атрибут `tabindex` должен получать одно из двух значений: 0 и –1. Значение `tabindex="-1"` (в данном случае подойдет любое отрицательное число, но принято использовать –1) можно применять для программируемого фокуса, а `tabindex="0"` — только если вы хотите предоставить доступ с клавиатуры не к ссылке и не к элементу формы, а к какому-то другому элементу, но не менять при этом порядок следования элементов на странице.

ПРИМЕЧАНИЕ

Может возникнуть вопрос: а какое отношение атрибут tabindex вообще имеет к мобильному Интернету, ведь табуляция здесь почти не применяется? Дело в том, что, когда вы заполняете элементы форм, на виртуальной клавиатуре, которая выводится на экране некоторых устройств, может присутствовать кнопка Next (Далее). Кроме того, может потребоваться обеспечить на мобильном устройстве такую фокусировку, как и при включении фокуса на ПК с помощью JavaScript. Наконец, далеко не у всех пользователей мобильных устройств есть специальный инструмент-указатель. Среди пользователей iPhone немало слабовидящих, и атрибут tabindex повышает для них доступность вашего ресурса.

accesskey

Атрибут `accesskey` напоминает `tabindex`. Отличие заключается в том, что в данном случае навигация по странице происходит путем перехода фокуса не к элементу с атрибутом `tabindex`, имеющим следующее по возрастанию значение, а прямо к элементу, с которым ассоциирован активизированный при этой операции ключ доступа. Таким образом, этот атрибут можно сравнить с горячей клавишей.

Горячие клавиши присваиваются тем или иным операциям именно на основании значения атрибута `accesskey`. Например, код `<input accesskey="s" name="search" type="text"/>` создает поле для поисковых запросов, атрибут `accesskey` которых равен `s`. Когда пользователь нажимает клавишу **s**, фокус перемещается в поисковое поле.

Значение атрибута `accesskey` представляет собой один или более символов, разделяемых пробелами. Как правило, в таких случаях используется всего один символ, но спецификации допускают присвоение одному элементу нескольких горячих клавиш. Синтаксис значения `accesskey` напоминает синтаксис атрибута `class` тем, что значение этого атрибута представляет собой упорядоченное множество маркеров, разделенных пробелами. Тем не менее в данном случае порядок важен: все маркеры после первого считаются резервными вариантами для тех пользовательских агентов, которые не смогут поддерживать исходное значение.

В свое время атрибуты `tabindex` и `accesskey` высоко ценились как средства для решения потенциальных проблем с доступностью, но они не являются идеальным решением такого рода. Как было указано ранее, `tabindex` может существенно ухудшить пользовательское восприятие, так как меняет ожидаемую последовательность фокусировки на элементах страницы. Аналогично атрибут `accesskey` может противоречить поведениям и горячим клавишам, заданным по умолчанию в клиентском браузере.

В момент написания книги я не знаю способов практического использования accesskey на смартфонах. Но, поскольку мы изучаем HTML5, я останавливаюсь здесь и на этом атрибуте. Атрибут accesskey был очень полезен до повсеместного распространения смартфонов, когда навигация в крошечном окошке в мобильном браузере обычного сотового телефона давалась с большим трудом.

Что нового в HTML5: глобальная доступность и интерактивные атрибуты

В HTML5 появилось несколько новых атрибутов, в частности новые глобальные атрибуты, о которых мы и поговорим в этом разделе. В следующих главах обсудим некоторые новые атрибуты, специфичные для определенных элементов, и их значения, когда будем рассматривать такие элементы.

hidden

Когда элемент сопровождается атрибутом hidden, это означает, что элемент уже не релевантен или еще не релевантен. Браузеры, поддерживающие данный атрибут, не отображают элементы, сопровождаемые этим атрибутом. Для этого в таблицу стилей пользовательского агента включается свойство display:none;. Лучше не применять этот атрибут, если требуется просто *спрятать* элементы от пользователя, так как он имеет семантическое значение. Данный атрибут указывает на нерелевантность содержимого этого элемента — например, возможно, контент элемента устарел.

contenteditable

Атрибут contenteditable указывает, поддается элемент редактированию или нет. При применении данного атрибута те изменения, которые пользователь вносит в ваш контент, не сохраняются. Но они сказываются на объектной модели документа, поэтому вы можете захватить все сделанные изменения и отправить их на сервер для сохранения. Атрибут contenteditable поддерживается во всех браузерах для ПК и во всех мобильных браузерах, кроме Opera Mini, на мобильных устройствах, Android 3.0 и выше и iOS 5 и выше.

Когда для элемента задан атрибут contenteditable, на экране мобильного устройства должна всплывать виртуальная клавиатура, обеспечивающая редактирование.

contextmenu

Атрибут contextmenu обеспечивает связывание элемента с элементом <menu>, сообщающим дополнительную контекстную информацию, либо с элементом <command>. В качестве значения этот атрибут принимает id того элемента <menu>, который вы хотите с ним ассоциировать. Этот атрибут пока не поддерживается практически ни в одном браузере, кроме Chrome (где его поддержка выполняется в качестве эксперимента), поэтому он не рассматривается в данной книге.

draggable

Атрибут `draggable` указывает, является ли элемент перетаскиваемым. Возможно, вы обращали внимание на то, что в большинстве браузеров для ПК перетаскивать изображения можно, а устанавливать их на новое место — нельзя. Именно так по умолчанию работает перетаскивание. Чтобы атрибут `draggable` был полезен, его нужно использовать вместе с обработчиками событий JavaScript — к их числу относятся `dragstart`, `drag`, `dragenter`, `dragleave`, `dragover`, `drop` и `dragend`. В мобильных браузерах, за исключением IE10, перетаскивание не поддерживается, поэтому данный API не рассматривается в этой книге.

dropzone

Перетащить элемент на экране — это полдела, а что с ним делать потом? В языке HTML5 имеется атрибут `dropzone`, указывающий, какие типы содержимого можно перетаскивать в данный элемент. Можно перемещать и копировать перенесенный таким образом контент либо создавать на него ссылку. Это делается с помощью значений данного атрибута `move`, `copy` и `link` соответственно. Поскольку операции перетаскивания не очень хорошо поддерживаются в мобильных браузерах, мы не будем рассматривать данный атрибут подробнее.

spellcheck

Атрибут `spellcheck` указывает, будет ли в данном элементе проверяться грамматика и правописание. По умолчанию на большинстве смартфонов и планшетов ошибки во вводимом вами тексте автоматически исправляются, но этот механизм работает не слишком хорошо. Есть даже несколько сайтов, на которых высмеиваются особенно неудачные «исправления». Дело в том, что такие устройства поддерживают автоматическое исправление, но при этом не поддерживают атрибут `spellcheck`.

Интересно отметить, что, хотя в iOS по умолчанию выполняется автоматическое исправление текста, при добавлении атрибута `autocorrect` к текстовому полю правописание не проверяется. Автоматическое исправление работает лишь при отсутствии этого атрибута.

Атрибуты доступности ARIA

В языке HTML5 поддерживаются атрибуты модуля доступности ARIA (стандарт обеспечения доступности активных интернет-приложений) `role` и `aria-*`. ARIA — это отдельный модуль, не входящий в состав спецификаций HTML5. В стандарте ARIA применяются живые области (live regions), роли, состояния и свойства ARIA. С помощью всех этих возможностей WAI-ARIA помогает повысить доступность динамически обновляемого контента и взломанных элементов.

Допустим, при взаимодействии с насыщенным интернет-приложением (в терминологии ARIA такие приложения именуются активными) пользователь не просматривает страницу, а прослушивает ее с использованием экранного диктора. При этом программа зачитывает вслух одну часть страницы, а тем временем другая ее часть динамически обновляется. Живые области ARIA подсказывают пользовате-

лю, что обновилась часть страницы — та, которая в данный момент находится не в фокусе. Когда атрибут `arialive` имеет одно из следующих значений: `assertive`, `polite` или `off`, задаваемое по умолчанию, ARIA предоставляет автору сайта возможность перебить голос экранного диктора и уведомить пользователя о том, что часть страницы обновилась. К числу ассоциированных атрибутов относятся `aria-atomic`, `aria-busy` и `aria-relevant`.

ARIA-атрибут `role` обеспечивает создание семантической структуры на базе переопределенных элементов. Например, этим атрибутом могут сопровождаться элементы, которые переопределены в качестве `grid`, `listbox`, `menu`, `menubar`, `tablist`, `toolbar`, `tree` или `treegrid`. Достаточно использовать этот атрибут — и с виду несемантическая разметка становится доступной, удобной и хорошо взаимодействует со вспомогательными технологиями. Хотя при полной поддержке новых элементов HTML5 в экранных дикторах некоторые структурные роли ARIA могут стать нерелевантными (см. главу 3), добавление ролевых значений `article`, `application`, `banner`, `complementary`, `contentinfo`, `document`, `form`, `heading`, `main`, `navigation` и `search` помогает работать с такими экранными дикторами, которые уже полностью поддерживают ARIA, а HTML5 — не полностью.

Два замечания о ролях. Во-первых, когда роль уже задана, `role` нельзя изменить динамически, поскольку это запутало бы вспомогательную технологию. Во-вторых, роли имеют приоритет над тем семантическим значением элемента, которое он имеет по умолчанию.

Наряду с атрибутом `role` и его многочисленными значениями ARIA включает в себя также атрибуты состояния и свойств. Существуют атрибуты состояния `aria-disabled`, `aria-busy`, `aria-expanded`, `aria-hidden` и атрибуты свойств, в частности `aria-describedby`, `aria-haspopup` и `aria-labelledby`, предоставляющие дополнительную информацию о переопределенных элементах. На практике рекомендуется максимально полагаться на семантические элементы, но если вам непременно требуется использовать конкретный элемент (допустим, древовидное меню) «не по назначению» — прибегайте к ARIA-атрибутам.

Специальные атрибуты данных с применением data-*

В HTML5 допускается создание собственных атрибутов. Конечно, ранее вы также могли создавать такие атрибуты, но эта разметка не прошла бы валидацию. В HTML5 появились специальные атрибуты данных. При этом вы как автор можете сами назвать атрибут.

Ранее разработчики включали в разметку невалидные атрибуты и/или использовали не по назначению атрибуты `title` и `rel`, чтобы обеспечить интерактивность данных. Вместо того чтобы злоупотреблять атрибутами `title` и `rel`, просто создайте атрибут с префиксом `data-`, и ваш код будет нормально проходить валидацию.

Например, в игре CubeeDoo мы хотим сохранять положение и значение каждой карты. Таким образом, сравнивая карты, мы можем определить, совпадает ли одна перевернутая карта с другой. С помощью `LocalStorage` мы также можем сохранять состояние, когда ставим игру на паузу и уходим с экрана. Каждую карту можно было бы отслеживать как массив JavaScript, но вместо этого мы создаем

в разметке атрибуты data-position и data-value, динамически обновляя data-value для каждой новой раздачи:

```
<div id="board" class="level1">
  <div data-value="0" data-position="1">
    <div class="face"></div>
    <div class="back"></div>
  </div>
  <div data-value="0" data-position="2">
    <div class="face"></div>
    <div class="back"></div>
  </div>
  <div data-value="0" data-position="3">
    <div class="face"></div>
    <div class="back"></div>
  </div>
  <div data-value="0" data-position="4">
    <div class="face"></div>
    <div class="back"></div>
  </div>
  ...
  <div data-value="0" data-position="24">
    <div class="face"></div>
    <div class="back"></div>
  </div>
</div>
```

Когда пользователь выбирает две карты, сравниваются их значения data-value. Если они равны, то мы имеем совпадение. Атрибут data-position позволяет отслеживать местоположение каждой карты и устанавливать в 0 значения атрибутов data-value двух карт, если эти карты совпадают. Мы также используем значение атрибута data-value для оформления лицевой стороны карты с помощью селекторов атрибутов (об этом мы поговорим в главе 7).

Когда атрибуты data-* еще не появились, мы структурировали карты с помощью или других подобных приемов. Хотя title и проходил валидацию, он был практически бесполезен и даже позволял игрокам жульничать, стоило навести указатель мыши на элемент (в этой игре по-прежнему можно плутовать с помощью инспектора элементов, но если в данном случае показывать всплывающую подсказку, игра слишком упростится). Для этой цели был зарезервирован префикс data-, который в будущем должен позволить избежать конфликтов с новыми версиями HTML. Специальные атрибуты данных предназначены для хранения специальных данных, приватных для страницы или приложения. Единственное требование, предъявляемое к этим атрибутам, — их нельзя использовать в качестве расширений пользовательских агентов, в частности -moz- или -webkit-.

API Dataset. Специальные атрибуты данных применяются с API dataset. Задействуя этот API, можно собирать пары «атрибут/значение», даже если имя специального атрибута данных генерируется динамически (то есть вы не знаете, какое имя атрибута будет стоять после дефиса):

```
1  // Получаем значения и позиции всех карт
2  // Используем dataset для получения значений всех карт
3  currCards = document.querySelectorAll('#board > div');
4  for (i = 0; i < qbdoo.cards; i++) {
5    cardinfo.push(currCards[i].dataset);
6  }
7  currentState.cardPositions = JSON.stringify(cardinfo);
```

Хотя нам и известно, какие атрибуты установлены, мы используем API `dataset`, а не `getAttribute()`. Так мы подсказываем API, что необходимо приостановить игру на время, пока мы извлекаем значения. В фрагменте кода из метода `qbdoo.pauseGame` используется селектор запроса для захвата всех карт (строка 3). Затем карты перебираются с помощью API `dataset`, в результате чего мы представляем в виде значения-массива все пары «ключ/значение» тех атрибутов множества данных, которые имелись в словаре `DOMStringMap`. В последней строке листинга (строка 7) собранные нами пары «ключ/значение» преобразуются в строку JSON. Можно было поступить и иначе, просто перебрав всю колоду:

```
1  for (i = 0; i < qbdoo.cards; i++) {
2    for (key in currCards[i].dataset) {
3      deck[key] = currCards[i].dataset[key];
4    }
5    cardinfo[i] = deck;
6  }
```

itemid, itemprop, itemref, itemscope и itemtype

Существует еще пять глобальных атрибутов, относящихся к *микроданным*. Эти атрибуты: `itemid`, `itemprop`, `itemref`, `itemscope` и `itemtype` — были исключены из основной части спецификации HTML5 и в настоящее время описываются в спецификациях микроданных. Я упомянул их здесь, чтобы в данном разделе были перечислены все глобальные атрибуты. Подробнее эти атрибуты будут рассмотрены в разделе «API микроданных» главы 6.

Синтаксис HTML-элементов и атрибутов

Итак, мы обсудили элементы и атрибуты, но еще не говорили о том, как включать их в код. Синтаксис очень важен, так что давайте им займемся.

Для включения элемента в код веб-страницы применяется открывающий и закрывающий теги. Открывающий тег начинается с левой угловой скобки (похожа на знак «меньше», <), затем идет имя элемента и, наконец, правая угловая скобка (похожа на знак «больше», >).

Правильно:

```
<a>
<p>
<div>
```

Неправильно:

```
<m> <!-- Элемент 'm' не существует -->
< div> <!-- Перед именем элемента не должно быть пробела -->
```

Если элемент сопровождается атрибутами, то они записываются в открывающем теге после имени элемента и разделяются пробелами. Все атрибуты представляют собой пары «имя/значение». В HTML5, в отличие от XHTML, не требуется явно объявлять логические значения атрибутов. Если такой атрибут присутствует, то браузер по умолчанию присваивает ему значение `true`, но это значение опускается.

Хотя в HTML5 и нет такого требования, ради обеспечения удобочитаемости и соответствия проверенным методам работы следует атрибут записывать в нижнем регистре, а его значения заключать в кавычки. В зависимости от типа атрибута его значение может быть чувствительным к регистру[1].

Атрибут может быть указан в каждом открывающем теге у любого элемента всего один раз. Вероятно, вы это знаете и не указываете в теге одни и те же атрибуты специально. Но это одна из самых распространенных ошибок, встречающихся при валидации, поэтому считайте, что я вас дружески предупредила.

Правильно:

```
<a href="http://www.standardista.com">
<p class="racket-tailed drongo">
<div id="content">
```

Пройдет валидацию, но неаккуратно:

```
<a href=http://www.yahoo.com>
<!-- Все значения атрибутов лучше заключать в кавычки -->
```

Не пройдет валидацию:

```
<p class="racket-tailed" class="drongo">
  <!-- Дублирование атрибутов не допускается. -->
<p class=Racket-Tailed Drongo>
  <!-- Хотя HTML5 и не требует заключать в кавычки все атрибуты, это рекомендуется
делать. Если в значении атрибута присутствует пробел, то кавычки необходимы, чтобы
недвусмысленно обозначить начало и конец значения атрибута! -->
```

Чтобы завершить (закрыть) элемент, ставится левая угловая скобка и прямой слеш, за которым следует имя элемента — то самое, которое уже было записано в открывающем теге. Затем идет правая угловая скобка. Если элемент является пустым (см. раздел «Самозакрывающиеся элементы» данной главы), то его можно завершить опциональным прямым слешем, идущим перед правой угловой скобкой открывающего тега:

```
<a href="http://standardista.com/mobile/ch2">Файлы для этой главы</a>
<p class="racket-tailed drongo">Экзотические птицы</p>
<div id="content">. . .</div>
```

---

[1] Значения, определяемые в спецификациях, как правило, нечувствительны к регистру. Исключение составляют определяемые вами строки, например ID и имена классов.

Между открывающим и закрывающим тегами помещается содержимое элемента (контент). Контент может включать в себя другие элементы и/или текстовые узлы. Соблюдайте правила вложения элементов! Если вы включаете один элемент в другой в качестве дочернего, то дочерний элемент должен быть как открыт, так и закрыт до окончания родительского элемента:

```
<div id="content">
  <p class="files">
    Примеры главы <a href="http://standardista.com/mobile">
    В Интернете</a>
  </p>
</div>
```

В данном примере в элементе содержится вся информация от первого открывающего < до последнего закрывающего >, в частности элементы-потомки «абзац» и «якорь». Обратите внимание: элемент-якорь <a> и открывается и закрывается в пределах элемента <p>. Элемент <p>, в свою очередь, полностью находится внутри элемента .

Самозакрывающиеся элементы

Закрывающие теги есть у всех элементов, кроме тех, которые называются *самозакрывающимися* (синоним — пустые элементы). По синтаксическим правилам XHTML такие элементы самозакрываются с помощью ведущего обратного слеша.

Пустые элементы не могут содержать вложенных элементов или текста. Будучи самозакрывающимися, они не имеют конечного (закрывающего) тега. При желании можно поставить слеш перед правой угловой скобкой открывающего тега. Хотя в HTML5 и не требуется обязательно закрывать элементы, в XHTML такое требование имеется — для закрытия элемента применяется ведущий слеш. К самозакрывающимся (пустым) элементам относятся:

- <img/> — изображение;
-
 — переход на новую строку;
- <meta/> — метаданные;
- <hr/> — горизонтальная разделительная линия;
- <base/> — базовый URL и стандартная цель для ресурсов и ссылок;
- <link/> — ссылка;
- <keygen/> — элемент формы для генерирования пары криптографических ключей;
- <area/> — область карты-изображения;
- <col/> — столбец таблицы;
- <command/> — команда меню;
- <embed/> — плагин (подключаемый модуль);
- <input/> — элемент формы;

- <param/> — параметр объекта;
- <source/> — медиаисточник (аудио или видео);
- <track/> — синхронизированное отслеживание медиа;
- <wbr/> — возможность разрыва строки.

Рекомендуемые методы

В языке XHTML действуют некоторые правила написания кода, которые в HTML являются опциональными или вообще не поддерживаются. HTML5 поддерживает оба варианта написания кода, но при написании разметки рекомендуется придерживаться следующих правил.

- **Пишите разметку в нижнем регистре.** В XHTML все теги элементов, а также все атрибуты должны записываться в нижнем регистре. Хотя в HTML5 и поддерживается «верблюжий регистр», а также другие возможности, пожалуйста, пишите всю разметку только в нижнем регистре. В документации W3C нигде прямо не указано, что названия атрибутов должны записываться в нижнем регистре, однако некоторые из них, например id, чувствительны к регистру, поэтому придерживайтесь этого правила.
- **Заключайте все атрибуты в кавычки.** В XHTML все значения атрибутов должны быть заключены в одиночные или двойные кавычки. В HTML5 такое требование предъявляется лишь к тем значениям атрибутов, которые содержат пробелы или специальные символы. Сделайте одолжение: пожалуйста, заключайте в кавычки все названия атрибутов.
- **Закрывайте все элементы.** В XHTML каждому открывающему тегу должен соответствовать закрывающий тег. Пустые элементы, в частности <img/> и
, должны быть самозакрывающимися. В HTML5 некоторые теги можно не закрывать. Конечно, опуская закрывающий элемент, мы сокращаем количество символов на странице, но такой код обычно сложнее читать, а значит, и поддерживать. В некоторых лекциях и блог-постах рекомендуется опускать закрывающие теги именно для сокращения количества символов и уменьшения файлов для мобильной среды. Однако несколько сэкономленных байтов информации не стоят сопряженного с этим риска. Лучше сокращать количество элементов объектной модели документа — так вы скорее сделаете файлы гораздо компактнее, чем состригая парочку символов. Сделайте одолжение: пожалуйста, закрывайте все элементы.

  Что касается использования ведущих слешей для закрытия элементов, их можно как применять, так и не применять. Однако выбрав один из двух вариантов, придерживайтесь его.
- **Все элементы должны быть вложены.** Все элементы должны быть правильно вложены. Если вы начинаете тег <a>, а затем начинаете тег (то есть вкладываете его в <a>), то сначала потребуется закрыть тег , а потом — тег . Аккуратно вкладывайте теги друг в друга: только при этом условии

разметка станет отображаться как следует, будет проще устранять ошибки, а вся разметка окажется валидной (к тому же вы обрадуете меня).

- **Указывайте значения для всех атрибутов, не являющихся логическими.** По правилам XHTML все атрибуты должны записываться в коде как пары «атрибут/значение», это касается даже логических значений. В XHTML опция, действующая по умолчанию, должна записываться как `selected="selected"`. В HTML5 такой же код можно упрощенно записать `selected`. В HTML5 можно и не указывать значения для логических атрибутов, поскольку при включении атрибутивного свойства логическое значение становится равно `true`, даже если в большинстве браузеров значение соответствующего атрибута равно `false`. Если вы решили включать в код значения логических атрибутов, всегда включайте их, если решили пропускать — всегда пропускайте.

  Не указывайте пустую строку `=""` для логического или даже для другого атрибута — напротив, всегда указывайте значение.

ВНИМАНИЕ

Обратите внимание: если вы включите в код логическое значение, оно будет равно true, даже если в качестве его значения будет задана пустая строка. Если вы хотите, чтобы это значение было равно false, пользуйтесь removeAttribute(attributeName), а не setAttribute(attributeName, ''), так как при применении пустой строки логический атрибут будет равен true.

Учтите, что использование пустых строк может повлечь за собой незапланированные последствия. Как будет показано в главе 3, эта проблема особенно актуальна для атрибута `form`.

- **Подбирайте для решения задачи такой элемент, чья семантика наиболее точно соответствует этой задаче.** В языке XHTML разметка должна создаваться по семантическому принципу. Таблицы и формы нельзя включать в абзацы. Элементы форм являются фразовыми, поэтому они должны содержаться в семантическом блочном элементе, например абзаце или ячейке таблицы. Разумеется, практически все содержимое вашей страницы может находиться в элементах `span` и `div`. Но если заголовок — это заголовок, то лучше воспользоваться тегом `<h1-6>`.

Необходимые компоненты

Элементы разметки — это кирпичики, из которых состоит Сеть. Если на сайте нет содержимого, до использования CSS или JavaScript просто не дойдет дело. Но на самом деле, чтобы веб-страница была написана на валидном XHTML, она должна содержать всего пять обязательных компонентов:

- объявление типа документа (DTD);
- корневой элемент HTML — `<html></html>`;
- заголовок документа, прямой потомок `html` — `<head></head>`;
- заглавие документа, находящееся в заголовке, — `<title></title>`;
- тело документа, прямой потомок `html` — `<body></body>`.

Иными словами, вот минимальный код, который должен присутствовать в документе HTML5, использующем синтаксис XHTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Пустой документ</title>
  </head>
  <body>
  </body>
</html>
```

Я включила в этот код элемент `charset` `<meta>`, но он вам не потребуется, если на сервере уже есть правильно установленные HTTP-заголовки. Если вы не управляете своим сервером, то вам также целесообразно включать этот элемент.

На самом деле для валидного документа требуется еще меньше кода, чем показано ранее. Если вы пропустите элементы `<html>`, `<head>` или `<body>`, то браузер неявно их включит. Фактически кратчайший валидный код на языке HTML будет таким:

```
<!DOCTYPE html>
<title>blank document</title>
```

Обратите внимание: на рис. 2.2 браузер самостоятельно включает недостающие теги. Если вы пропустите `<html>`, `<head>`, `<body>` или закрывающие теги, то браузер отобразит документ правильно, добавив в DOM нужные узлы и дописав закрывающие теги.

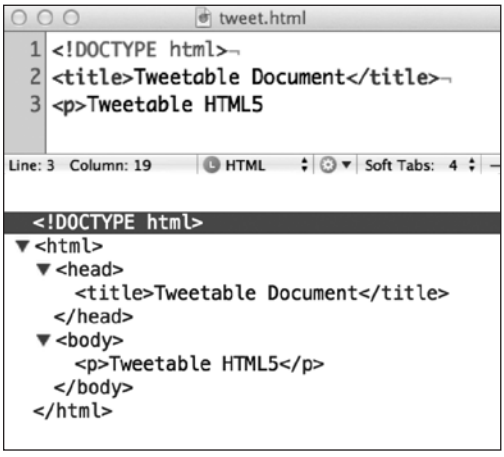


Рис. 2.2. Если пропустить теги <html>, <head> или <body>, то браузер все равно отобразит документ правильно, добавив соответствующие узлы в DOM (*вверху* — HTML-код, *внизу* — отображенный документ)

Действительно, целый HTML-документ можно записать с помощью менее чем 140 символов. Но хотя вы и можете опустить три из пяти обязательных элементов,

это еще не значит, что так и следует делать. Чтобы повысить удобочитаемость и упростить поддержку документа, особенно если читать и поддерживать его будете не вы, придерживайтесь выбранного стиля написания кода. Предпочтительно писать код в стиле XHTML, где требуется включать все пять элементов.

Давайте подробнее рассмотрим все пять обязательных компонентов.

Объявление типа документа

Объявление типа документа (сокращенно — тип документа (doctype), или DTD) сообщает браузеру, какой синтаксис вы использовали при написании разметки, чтобы браузеру было понятно, как его обрабатывать.

Эта информация почти всегда должна отсылаться браузеру в первую очередь. Исключение составляет XML-пролог — в случаях, когда синтаксический разбор кода должен происходить по правилам XML. В предыдущем примере используется DTD по правилам HTML5. Во-первых, оно самое короткое, во-вторых, именно такие документы и являются темой этой книги. Но существуют и другие разновидности объявлений документа (табл. 2.1).

Таблица 2.1. Типы документа HTML 4, XHTML и HTML5

Тип страницы	Объявление типа документа (DTD)
HTML 4.01 Transitional	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/HTML 4/loose.dtd">
HTML 4.01 Strict	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/HTML 4/strict.dtd">
XHTML 1.0 Transitional	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
XHTML 1.0 Strict	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
XHTML 1.1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
HTML5	<!DOCTYPE html>[1]

Возможно, вы используете XHTML DTD уже на протяжении 10 лет. Но при этом можете просто вырезать такое объявление из своих старых документов и вставлять в новые. Вечно не хватает времени выучить это объявление наизусть. Да и мне тоже! Однажды записав объявление типа документа для HTML5, я больше не заглядываю в этот раздел кода.

Все современные мобильные браузеры поддерживают все разновидности объявлений типа документа для HTML и XHTML, в том числе новое, сокращенное объявление, применяемое в HTML5.

Возможно, вы беспокоитесь, как ваш старый код будет взаимодействовать с новым типом документа для HTML5. Не волнуйтесь[2]. Если ваша веб-страница нормально

[1] Как и другие компоненты HTML5, объявление типа документа нечувствительно к регистру.

[2] Объявление типа документа для HTML5 переводит старые версии IE для ПК в режим совместимости.

валидировалась как HTML 4 или XHTML Strict, то такая разметка будет валидна и в HTML5. Браузеры, поддерживающие HTML5, должны обеспечивать обратную совместимость со всеми предыдущими версиями HTML и XHTML, в том числе с устаревшими элементами. Однако если ваш браузер поддерживает нежелательные элементы, это совсем не означает, что ими следует пользоваться! Возможно, `<center>` и `<font>` будут отображаться нормально, но такой код определенно не является валидным и качественным![1] Чтобы ваш код был согласованным, чистым и расширяемым, рекомендую использовать синтаксис XHTML — в частности, заключать в кавычки значения всех атрибутов и закрывать все элементы. В HTML5 допускается практика «ленивого» кодирования, но вы не будьте ленивы!

<html>

Элемент `<html>` — это корневой элемент HTML-документа. В HTML5 он опционален, но в синтаксисе XHTML — обязателен. В переходном варианте HTML 4 этот элемент необязателен, как и в HTML5. Но мы собираемся писать хороший, чистый код, соответствующий стандартам. Поэтому фактически этот элемент является обязательным и должен записываться так, как показано в табл. 2.2.

В элемент `<html>` вложены два дочерних элемента, `<head>` и `<body>`. Рекомендуется (но не требуется) включать в элемент `<html>` атрибут `lang`.

Таблица 2.2. Элемент HTML и его оформление в зависимости от типа страницы

Тип страницы	Элемент HTML
HTML 4.01 Transitional HTML 4.01 Strict HTML5	<html> <html lang=»en»>
XHTML 1.0 Transitional XHTML 1.0 Strict XHTML 1.1	<html xmlns="http://www.w3.org/1999/xhtml"> <html lang="en" xmlns="http://www.w3.org/1999/xhtml">

В HTML5 предоставляется новый атрибут элемента `<html>` — атрибут `manifest`. Если такой элемент присутствует, он получает в качестве значения URL файла манифеста (описания). Кэш приложений, файл манифеста и офлайновые приложения рассмотрены в главе 6.

Некоторые сценарные инструменты HTML5, например modernizr, добавляют классы к открывающему элементу `<html>`. Это вполне допустимо. Когда я работаю с кэшем приложений, сценарием modernizr для обнаружения возможностей и с объявлением языка, открывающий тег `<html>` зачастую выглядит так:

```
<html lang="en" manifest="cache.appcache" class="no-js">
```

Атрибут `lang` был рассмотрен ранее. В главе 6 мы поговорим об атрибуте `manifest`, используемом при офлайновой работе и работе с хранилищем данных. Класс `no-js`

[1] Устаревшие функции перечислены по адресу http://bit.ly/16t5Z6L

следует включать, если вы используете modernizr[1] для тестирования нативных реализаций различных веб-технологий, рассматриваемых в этой книге.

<head>

Элемент <head> в документе содержит важную информацию. Но она, в отличие от содержимого элемента <title>, не отображается прямо в окне браузера[2]. Как правило, содержимое элемента <head> не выводится на экран — в большинстве браузеров пользователь видит лишь содержимое элемента <title>, обязательного в валидном документе HTML5. Это заглавие пользователь видит в названии вкладки или в другом элементе окантовки окна браузера. Прочее содержимое элемента <head> сообщает браузеру о том, как следует отображать страницу, а также информирует поисковых роботов о контенте страницы. Возвращаясь к различиям между HTML 4 и HTML5, следует сказать и об атрибуте profile элемента <head>. Ранее этот атрибут практически не использовался, поэтому он не включен и в спецификации HTML5.

Элемент <head> является родительским для обязательного <title>, а также для опциональных <style>, <script>, <link>, <meta> и <base>.

<title>

Элемент <title> является *обязательным* и должен сопровождаться закрывающим тегом </title>. Ваша страница может пройти валидацию без тегов <head>, <body> и даже <html>, но не без <title>. Более того, если пропустить закрывающий тег этого элемента, то синтаксический разбор страницы также не удастся выполнить. Тег <body> может быть пуст, если вы не собираетесь показывать пользователю какое-либо содержимое. Даже элемент <title> может быть пуст, но он обязательно должен присутствовать.

Контент тега <title> должен определять общее содержимое элемента. Может показаться, что элемент <title> не так уж важен для компоновки страницы, однако этот элемент имеет первостепенное значение для поисковых роботов.

Обратите внимание на то, как контент тега <title> выглядит в строке заголовка в браузере (рис. 2.3). Конечно, поскольку <title> важен для поисковой оптимизации (SEO), на мониторе ПК такое название может выглядеть вполне сносно, но на крошечном экране смартфона подобная надпись быстро начинает раздражать пользователя. Поэтому внимательно подходите к выбору названия страницы.

<body>

Итак, мы добавили на веб-страницу элемент <title>, но она по-прежнему пуста. Весь контент, который вы собираетесь отображать на странице, должен находиться в теле

---

[1] Modernizr — это библиотека JavaScript, обнаруживающая в пользовательском браузере поддержку HTML5 и CSS3.

[2] Если требуется показать пользователю содержимое элемента <head>, это можно сделать с помощью CSS.

Рис. 2.3. Содержимое элемента <title>, отображаемое в окне браузера Safari (iOS) и в браузере операционной системы Firefox OS

страницы, то есть его нужно заключить в элемент <body>. Элемент <body> — это второй и последний из непосредственных потомков корневого элемента <html>.

Существует ряд презентационных атрибутов элемента <body>, которые были признаны нежелательными в XHTML. HTML5 продолжает эту оригинальную традицию XHTML — он не содержит никаких презентационных атрибутов, присутствовавших в HTML 4, в частности `align` и `bgcolor`, а также средств для окрашивания фона и ссылок. Дело в том, что функции этих атрибутов лучше реализуются в CSS. Скорее всего, вы будете добавлять к элементу <body> лишь атрибуты `id` и `class`, а при необходимости также `lang` и `dir`.

Просматривая исходный код веб-страницы, вы нередко будете замечать в открывающем теге <body> обработчики событий, например `onload="doSomething();"`. Вообще к открывающему тегу <body> следует добавлять только глобальные атрибуты, например `class` или `id`. Обработчики событий должны находиться во внешнем файле JavaScript, оформление и стили — во внешнем файле CSS.

Наш первый каркасный документ на HTML5 должен иметь примерно следующий вид. Отличаться могут только выбранная кодировка (об этом в дальнейшем) и объявление типа документа:

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8"/>
  <head>
    <title>Моя первая веб-страница на HTML5</title>
  </head>
<body>
</body>
</html>
```

Из этих шести компонентов (объявление типа документа, корневой элемент <html>, элементы <head> и <meta>, кодировка, элемент <title> и элемент <body>) мы создали веб-страницу. Она совершенно пуста, тем не менее это настоящая веб-страница. А если мы правильно подобрали содержимое для элемента <title>, то поисковикам будет гораздо проще найти в Интернете эту страницу, чем многие существующие сайты.

Как было указано ранее, можно обойтись и меньшим количеством элементов:

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

```
<title>Моя первая веб-страница на HTML5</title>
<p>Привет, мир!
```

Это валидный HTML-документ, но написан он не совсем качественно. Если вы станете писать код более аккуратно, то ваши коллеги-разработчики, которым позже придется поддерживать этот код, а также вы сами будете лучше понимать его назначение.

Элементы, находящиеся в <head>

Элемент <head> может показаться наименее интересной частью разметки веб-страницы, так как по умолчанию пользователь его не видит. Но если этот элемент вам неинтересен, это еще не означает, что к нему можно относиться пренебрежительно. Именно здесь, в заголовке веб-документа, разработчик рассказывает браузеру, как должна отображаться веб-страница. Здесь можно поместить подсказки для принтера, поисковика, синтаксического анализатора, а также прочую информацию об обработке контента.

В элементе <head> вы всегда найдете <title>, но здесь же вам могут встретиться и другие теги: <meta>, <base>, <link>, <script>, <style>, <command> и <noscript>.

Вот как может выглядеть элемент-заголовок, чрезмерно насыщенный информацией:

```
<head>
<meta charset="UTF-8"/>
<title>Мобильный HTML5</title>
<meta name="author" content="Estelle Weyl"/>
<meta name="publisher" content="O'Reilly"/>
<meta name="copyright" content="Copyright 2013"/>
<meta http-equiv="date" content="Пон, 18 ноя 201316:15:30 GMT"/>
<meta http-equiv="date-modified"
      content="11/18/2013" scheme="MM/DD/YYYY"/>
<meta name="keywords" content="html5 css3 svg
      border-radius canvas audio iphone android ipad"/>
<meta name="description" content="Переход от настольных компьютеров
      к мобильным: изучаем CSS3 и HTML5."/>
<meta name="pagetopic" content="Internet"/>
<meta name="page-type" content="Instruction"/>
<meta name="audience" content="all"/>
<meta name="robots" content="index,follow"/>
<meta name="generator" content="Sublime"/>
<meta name="apple-mobile-web-app-capable" content="yes"/>
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<base href="http://www.standardista.com/"/>
<script src="/js/application.js"></script>
<link rel="apple-touch-icon" href="touch-icon-iphone.png"/>
<link rel="apple-touch-icon" sizes="72x72"
      href="touch-icon-ipad.png"/>
<link rel="apple-touch-icon" sizes="114x114"
      href="touch-icon-iphone4.png"/>
```

```
<link href="/css/prettification.css" media="all" rel="stylesheet"/>
<link href="/css/tinylittledevice.css" media="только экран и
      (max-device-width: 480px)" rel="stylesheet"/>
<link href="/css/print.css" media="print" rel="stylesheet"/>
<style>
    p {color: #333333;}
</style>
</head>
```

Это очень большой заголовок, вам *никогда не следует* делать подобных заголовков. Тем не менее в нем необходимо разобраться. Итак, что же все это значит? Посмотрим...

<meta>: добавляем метаданные

Элемент <meta> позволяет веб-разработчику записывать на страницах различные метаданные, указывая соответствующие свойства и значения. Есть четыре атрибута, специфичных для элемента <meta>: charset, http-equiv, content и name.

<meta charset="UTF-8">

Первый вариант элемента <meta>, который мы здесь рассмотрим, по-видимому, будет встречаться во всех создаваемых вами документах HTML5:

```
<meta charset="utf-8"/>
```

Вероятно, вы добавляете в свои документы код:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

уже много лет. Так вы сообщаете браузеру, что страница написана на языке HTML и требуется использовать кодировку UTF-8, если HTTP-заголовки на сервере не сконфигурированы для автоматической установки кодировки.

Предполагается, что элемент <title> должен идти первым после открывающего тега <head>. Однако для объявления кодировки делается исключение, поскольку мы хотим гарантировать, что агент рендеринга будет точно знать, в какой именно кодировке должны отображаться символы. Эта деталь появилась только в HTML5, но уже поддерживается во многих крупных браузерах, так как им и ранее приходилось справляться с неправильно записанными элементами meta, в которых не хватало кавычек:

```
<meta http-equiv=Content-Type content=text/html; charset=utf-8>
```

Обратите внимание: здесь тег <meta> записан неправильно, поскольку в нем отсутствуют кавычки и есть пробел в значении атрибута содержимого. Поэтому браузер интерпретирует charset как отдельный атрибут. Эта недавняя «ошибка» правильно обрабатывалась во всех браузерах, поскольку была распространена повсеместно. Теперь она внедрена в HTML5 уже в рамках спецификаций.

Вы можете (и должны) доставлять с сервера все ваши файлы в кодировке UTF-8. Если вы пользуетесь сервером Apache, добавьте в ваш файл .htaccess запись AddDefaultCharset UTF-8.

Если не считать charset, тип тега <meta> определяется по значению одного из атрибутов — name или http-equiv. Кроме charset, каждый тег <meta> должен содержать атрибут name *или* атрибут http-equiv, *а также* атрибут content.

В большинстве случаев, если вы только не собираетесь сгенерировать заголовок сообщения HTTP-отклика, тег <meta> имеет атрибуты name и content, но значения двух этих атрибутов, в принципе, могут быть любыми. Вы сами создаете значения для имени и для содержимого. Сначала мы рассмотрим повсеместно распространенные типы <meta>, а потом те типы, которые специфичны для мобильной среды.

Метатег description

Метатег может иметь одно из нескольких значений. Самое важное из этих значений, не являющееся специфичным для мобильных устройств, — *description* (описание). Содержимое тега "description" <meta> — как раз та информация, которую вы видите на странице поисковика, когда ваша страница попадает в результаты запроса. Поэтому убедитесь, что значение content вашего описания — это правильно оформленное информативное предложение, сообщающее основную важную информацию о странице и ключевые слова (рис. 2.4):

```
<meta name="description"
  content="CSS3, JavaScript and HTML5 as explained by Estelle Weyl">
```

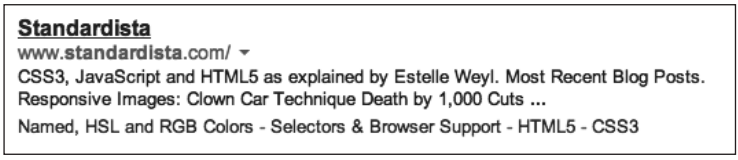


Рис. 2.4. Содержимое метатега с описанием, отображаемое в результатах поиска на сайте Google

Метатег keyword

Из всех метатегов тег "keyword" наиболее известен. Тем не менее, поскольку спамеры злоупотребляют этим тегом на протяжении всего текущего тысячелетия, поисковики не слишком высоко ценят содержащуюся в нем информацию. Здесь вы вполне можете указать метаданные в виде ключевых слов, но не рассчитывайте на то, что это значительно повысит ваши позиции в поисковиках:

```
<meta name="keyword" content="CSS3, HTML5, JavaScript">
```

<meta http-equiv="...">

Ранее мы говорили о том, что содержимое атрибута name элемента <meta> может быть фактически любым, но об атрибуте http-equiv такого не скажешь. Атрибут http-equiv, используемый вместо name, может взять на себя функцию сервера по созданию заголовков сообщений для HTTP-откликов. Значения атрибута http-equiv имитируют заголовки HTTP-откликов. На сайте http://www.standardista.com/html5/http-equiv-the-meta-attribute-explained/ я создала список значений атрибута http-equiv для тега <meta>. Если у вас есть доступ к файлу .htaccess с вашего сервера, задайте

заголовки прямо в этом файле. Установка заголовков в теге <meta> — это резервный, но не основной выход:

```
<meta http-equiv="cache-control" content="no-cache" />
```

Метатеги для мобильной среды

Существует ряд метатегов, специально приспособленных для использования на мобильных устройствах. К ним, в частности, относятся метатеги, которые приказывают окну браузера занять всю доступную область просмотра, отключить масштабирование и изменить цвет статусной панели (об этом мы поговорим в следующем разделе).

Метатег viewport. Если вы, работая на ПК, не сделаете окно браузера больше по ширине, чем размер монитора, размер области просмотра будет равен размеру окна браузера. На большинстве мобильных устройств масштаб страницы можно контролировать, но размер области просмотра не изменяется и остается равным размеру экрана[1].

Метатег "viewport" позволяет задавать логические размеры и масштабирование для окна браузера, соответствующего области просмотра на мобильном устройстве. В нашем приложении CubeeDoo мы воспользуемся следующим метатегом:

```
<meta name="viewport"
  content="user-scalable=no, width=device-width, initial-scale=1.0"/>
```

Метатег, описывающий область просмотра, поддерживается на всех смартфонах и других мобильных устройствах с операционными системами iOS, Android, webOS, Windows Phone и Opera Mobile. Устанавливая ширину области просмотра равной device-width, мы приказываем браузеру сделать ширину документа равной ширине экрана на устройстве. Элементарно!

Метатег области видимости поддерживает список команд, разделенных запятыми. С их помощью мы можем задавать ширину, масштаб и высоту области просмотра в браузере. Можно запретить браузеру применять масштабирование либо выполнять масштабирование в диапазоне от максимального до минимального значения, которое мы задали.

- width=<*num*>|device-width. Как правило, ключевой термин device-width приходится использовать в случаях, когда ширина области просмотра должна быть равна ширине экрана устройства, но числовые значения здесь также допустимы. Значение, задаваемое по умолчанию, в разных браузерах варьируется, но, как правило, оно близко к 980. Минимальное значение — 200, максимальное — 10000.
- height=<*num*>|device-height. Это значение, устанавливаемое в device-height или, например, равное 480 у iPhone 4, определяет высоту области просмотра. Как

[1] На самых миниатюрных устройствах браузер автоматически переключается в полноэкранный режим, и пользователь не может на это повлиять. На некоторых устройствах, например на планшете Slate, размер окна браузера можно корректировать.

правило, это значение опускается, задается только значение ширины. Для справки: минимальное значение в данном случае 223.

- user-scalable=no|yes. Определяет, может ли пользователь увеличивать или уменьшать размеры контента, то есть масштабировать область просмотра. Чтобы разрешить масштабирование, задайте здесь yes, чтобы запретить — no. По умолчанию действует значение yes. При этом значении данные не прокручиваются. Если пользовательское масштабирование допускается, его можно ограничить с помощью значений minimum-scale и maximum-scale, присваиваемых метатегу области просмотра.
- initial-scale=<*float*>. Задает исходный коэффициент (множитель) масштабирования, используемый при просмотре веб-страницы. По умолчанию задается значение 1.0, при котором веб-страница отображается без масштабирования.
- maximum-scale=<*float*>. Задает максимальный лимит пользовательского масштабирования, если параметр user-scalable не установлен в no. Максимальное значение — 10, но это может быть любое дробное значение от 0.25 и выше.
- minimum-scale=<*float*>. Задает минимальный лимит пользовательского масштабирования веб-страницы. Минимальное значение — 0.25.

В большинстве браузеров ширина области просмотра по умолчанию составляет 980 пикселов. Устанавливая здесь значение device-width, пользователю не приходится увеличивать загруженную страницу, поскольку мы выдали ему картинку шириной 980 пикселов, тогда как ширина экрана устройства — всего 320 пикселов. Мы могли бы задать для области просмотра значение 320, а не device-width, но в таком случае страница будет корректно отображаться лишь на устройствах именно с такой шириной экрана. Оптимальный вариант — приравнивать ширину окна к ширине экрана устройства, так как страница будет масштабироваться пропорционально устройству. При этом для автора сайта не имеет значения, с каким именно устройством работает пользователь.

Однако такая тактика оптимальна при работе не с любыми сайтами, а только с мобильными сайтами и веб-приложениями. Кроме того, когда активирована команда user-scalable, пользователь может увеличить страницу, чтобы ее легче было читать и людям с ослабленным зрением.

Мы создаем приложение CubeeDoo — мобильное приложение-игру. В ходе игры пользователь будет прикасаться к сенсорному экрану. Мы не хотим, чтобы пользователь случайно увеличил или уменьшил масштаб экрана, пытаясь перевернуть одну из карточек. Поэтому мы приказываем браузеру запускать игру в полноэкранном режиме и отключаем в ней масштабирование. Если бы мы программировали сайт, а не веб-приложение, то для повышения удобства использования разрешили бы пользователю выполнять масштабирование без ограничений. Мы делаем это в меню для выбора языка:

```
<meta name="viewport" content="user-scalable=yes,
      width=device-width, initial-scale=1.0"/>
```

Функция метатега, отвечающего за область просмотра, чисто презентационная и никогда не описывалась в каких-либо спецификациях. Эта возможность появилась под влиянием Apple. В настоящее время активно разрабатывается спецификация, описывающая реализацию этой функции не в HTML-разметке, а в CSS с помощью `@viewport`. Такая функциональность частично поддерживается в IE10.

Специфические значения в мобильной среде, определяемые поставщиками

В мобильной веб-разработке действует несколько значений, определяемых поставщиками. Например, Google и Apple создали собственные метапары «имя/значение» для интеграции программ с некоторыми из своих сервисов/API. В нашем проекте мы используем три таких тега — поговорим о них в следующих подразделах.

apple-mobile-web-app-capable

Этот метатег называется `"apple"`, но поддерживается и на Android. Он указывает, должно ли приложение работать в полноэкранном режиме. Если приложение работает в полноэкранном режиме, никакой браузерной окантовки не видно. Кроме окна браузера, на экране отображается только статусная панель телефона. Этот метатег влияет на веб-приложение лишь в том случае, если сайт попал в закладки. Мы включаем его в приложение именно на тот случай, когда пользователь открывает это приложение из закладок. Тогда программа займет на экране столько места, сколько сможет:

```
<meta name="apple-mobile-web-app-capable" content="yes"/>
```

Если атрибут `content` равен `yes`, то приложение работает в полноэкранном режиме, в противном случае этого не происходит.

Чтобы узнать, отображается ли страница в полноэкранном режиме, можно применить JavaScript, воспользовавшись логическим свойством `window.navigator.standalone`. Оно предоставляется только для чтения.

apple-mobile-web-app-status-bar-style

Как было указано ранее, даже при включении полноэкранного режима с помощью `apple-mobile-web-app-capable` на экране все равно остается статусная панель устройства. Это единственный элемент интерфейса, который разработчик не может убрать с экрана даже в нативных веб-приложениях. Но сравнительно немногие разработчики знают, что в некоторой степени вы все-таки можете управлять внешним видом статусной панели. В частности, можно влиять на ее цвет и прозрачность с помощью метатега `apple-mobile-web-app-status-bar-style`:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

Если ваше веб-приложение выдержано в темных тонах (в основном черное), то стоит оформить в такой гамме и статусную панель. В таком случае вы можете улучшить дизайн своего приложения и придать ему более нативный вид. В iOS для

этой цели применяются значения `default`, `black` и `black-translucent`. К сожалению (или к счастью, учитывая, на какую безвкусицу способны некоторые дизайнеры), это единственные варианты, доступные в настоящее время.

format-detection

Мобильный метатег `format-detection` включает или отключает автоматическое обнаружение телефонных номеров, которые могут присутствовать на веб-странице:

```
<meta name="format-detection" content="telephone=no"/>
```

По умолчанию некоторые устройства автоматически обнаруживают строки, отформатированные как телефонные номера. При этом на устройстве такие номера преобразуются в ссылки, нажимая на которые можно позвонить по конкретному номеру или как минимум открыть телефонное приложение, в котором уже введен этот номер. Указав `telephone="no"`, мы отключаем такую возможность. В приложении, которое мы собираемся написать, не будет ни телефонных номеров, ни каких-то других номеров такого рода, поэтому мы не используем в программе этот метатег.

Тег <base> веб-страницы

Тег `<base>` практически не используется, но оказывается весьма кстати, если требуется выполнить локальное тестирование. Элемент `<base>` дает базовый URL для разыменования относительных URL. Допустим, у вас в коде есть относительно расположенное изображение `<img src="image/drongo.gif"alt="Райский дронго"/>`. При применении ссылки `<base src="http://RacketTailedDrongo.com"/>` браузер перейдет в каталог с изображениями на сервере RacketTailedDrongo.com и найдет там рисунок `drongo.gif`[1].

Базовый URL можно переопределить с помощью HTTP-заголовка, но, как правило, этот URL удобно использовать при локальном тестировании. Попробуйте сохранить на жестком диске файл из Интернета. Поставьте указатель `base` на исходный сервер в элементе `<head>` вашего файла. Когда вы откроете страницу на локальном компьютере, она, скорее всего, отобразится правильно, хотя файл и не был на сервере, вы ничего ниоткуда не скачивали и не изменяли путь к вашему файлу в теле страницы. Синтаксис таков:

```
<base href="http://www.mydomain.com"/>
```

Элементы <link> применяются не только для работы с таблицами стилей

Элемент `<link>` не получил должного признания. Это мощный тег, которым любят пользоваться, но делают это не вполне рационально. Элемент `<link>` позволяет определять взаимосвязи между вашим HTML-документом и другими документами и ресурсами. Элемент `<link>` можно использовать для управления отображением

[1] Если вам уже любопытно, почему я все время ссылаюсь на райского дронго, подскажу: это та самая птичка, которая изображена на обложке книги.

документа при печати, связывания таблиц стилей и сценариев, определения фавиконов[1], представления альтернативных форм данного документа.

В нашем приложении будет четыре тега <link>:

```
<link rel="icon" href="/appleicon.png"/>[2]
<link rel="apple-touch-icon" href="/appleicon.png"/>
<link rel="apple-touch-startup-image" href="startup.png"/>
<link rel="stylesheet" href="styles.css"/>
```

Элемент <link> может сопровождаться несколькими атрибутами, в частности href, rel, type, sizes, hreflang, media и title. Атрибуты rev и charset элемента <link> в HTML5 были упразднены. Синтаксис таков:

```
<link href="url to resource" rel="type of relationship" title="title"/>
```

Атрибут rel указывает именованное отношение между актуальным документом и тем ресурсом, который указан в атрибуте href. Атрибуты rel и href являются *обязательными*.

Из четырех вышеупомянутых тегов <link> первые три предназначены для работы с изображениями, а четвертый должен быть вам наиболее хорошо знаком. Он обеспечивает связывание с таблицами стилей.

Добавление элементов <link> для связывания с таблицами стилей

Маленький да удаленький элемент <link> можно использовать для отправки различных таблиц стилей на смартфон, планшет или ПК. С его помощью можно отсылать различные таблицы стилей в зависимости от того, под каким углом в данный момент расположен дисплей устройства, либо в зависимости от ширины окна пользовательского браузера.

Здесь мы включили в приложение всего одну таблицу стилей, но могли включить и несколько. Каждая таблица стилей может быть нацелена на работу с разными или частично пересекающимися медиа, размерами браузера, разрешениями экрана и даже ориентациями окна браузера:

```
<link href="/css/styles.css" media="all" rel="stylesheet"/>
<link href="/css/tinylittledevice.css"
  media="only screen and (max-device-width: 480px)" rel="stylesheet"/>
<link href="/css/print.css" media="print" rel="stylesheet"/>
```

Эти примеры должны быть вам знакомы, за исключением, пожалуй, лишь двух аспектов. Обратите внимание: здесь отсутствует пара «атрибут/значение» type="text/css". В настоящее время не существует (и не предвидится) никакого иного языка таблиц стилей, поэтому HTML5 предполагает, что в данном случае имеет дело

---

[1] Фавикон — это значок сайта, отображаемый, например, в закладках браузера, см. http://ru.wikipedia.org/wiki/Favicon. — *Примеч. пер.*

[2] Код rel="shortcut icon" добавляется для IE. Этот браузер требует указывать термин shortcut, когда фавикон называется иначе, нежели favico.ico, либо не хранится в корневом каталоге сайта.

с информацией типа text/css и специально указывать это излишне. Кроме того, вы могли ранее не сталкиваться с кодом media="only screen and (max-device-width: 480px)". В следующих разделах мы рассмотрим и этот атрибут, и другие атрибуты тега <link>, а медиазапросы более подробно обсудим в главе 7.

Атрибуты тега <link>

Как и почти все другие элементы, тег <link> принимает все глобальные атрибуты. В табл. 2.3 перечислены другие атрибуты этого элемента.

Таблица 2.3. Атрибуты элемента <link>

Атрибут	Описание
href	Обязательный атрибут. Эта «гиперссылка» представляет собой URL и указывает на целевой файл <link>
media	Описывает, к какому типу медиа относится содержимое ссылки
rel	Указывает отношение <link> к актуальному документу
hreflang	Язык медиаинформации, связанной с документом
type	MIME-тип связанного медиа. Является опциональным, если связь именно такая, как предполагает отношение rel
sizes	Новый атрибут. Если медиа представляет собой пиктограмму, то этот атрибут определяет ее размер

Атрибут media

Атрибут media описывает, к какому типу медиа относится контент, к которому можно перейти по ссылке. Если никакое значение прямо не указано, то по умолчанию действует значение all. При таком значении всегда присутствует источник, описываемый href.

Набор значений этого атрибута всегда был очень ограничен, например: screen для ПК, print для принтеров и т. д. К числу значений для различных медиа относятся screen, tty, tv, projection, handheld, print, braille, aural и all. Теперь сюда можно включать и @media-запросы. С появлением CSS3 важность атрибута media значительно повысилась. Теперь мы можем выдавать разные таблицы стилей, основываясь на довольно непростых значениях свойства media. Например, изменив ориентацию мобильного устройства или размер окна браузера на ПК, мы одновременно изменяем соотношение сторон области просмотра. Вот как можно выдавать разные файлы CSS в зависимости того, в каком режиме — книжном или альбомном — работает ваше приложение:

```
<link rel="stylesheet" media="все и (orientation:portrait)" href="prtrt.css"/>
<link rel="stylesheet" media="все и (orientation:landscape)" href="lndscp.css"/>
```

Запросы @media — это одна из новинок CSS3, но такие значения вполне приемлемы для атрибута media. Вот наиболее важные значения этого атрибута, распространившиеся в последнее время:

- (min/max)-width — ширина области просмотра;
- (min/max)-height — высота области просмотра;
- (min/max)-device-width — ширина экрана;

- (min/max)-device-height — высота экрана;
- orientation — ориентация экрана, книжная (высота больше ширины) или альбомная (высота меньше ширины);
- (min/max)-aspect-ratio — соотношение ширины и высоты экрана;
- (min/max)-device-aspect-ratio — соотношение ширины и высоты устройства.

Некоторые атрибуты <link>, присутствовавшие в HTML 4, были исключены из спецификации HTML5, в частности атрибуты coords, shape, urn, target, charset, methods и rev. Атрибут title этого элемента также обладает необычной семантикой.

Атрибут rel

Атрибут rel описывает именованное отношение между актуальным документом и тем ресурсом, который указан в атрибуте href. Атрибут rel считается необязательным, но если не включить его, то браузер не сможет правильно связать ресурс с документом. Ссылка на таблицу стилей без rel="stylesheet" не сможет правильно отобразить ни один стиль; ваш браузер просто загрузит файл со стилями и подумает, что это информационный мусор (да, браузеры умеют думать, и к тому же постоянно смеются над нами — а вы не знали?). В табл. 2.4 перечислены и описаны некоторые значения атрибута rel элемента link.

Таблица 2.4. Атрибут rel

Значение	Описание
stylesheet	Наиболее распространенное значение атрибута rel, сообщающее браузеру, что связанный файл должен использоваться при рендеринге представления актуального документа. Если вы используете stylesheet, то пара «атрибут/значение» type="text/css" больше не является обязательной, поскольку text/css — это единственный существующий язык таблиц стилей и данная информация выводится логически
next	Ссылка указывает на следующий документ в пошаговом руководстве или в упорядоченной последовательности. В спецификации изначально предполагалось использовать этот атрибут для предварительной загрузки следующего файла и улучшения пользовательского восприятия ресурса
prev	Ссылка указывает на предыдущий документ в пошаговом руководстве или в упорядоченной последовательности
index	Индекс документа
help	Ссылка указывает на справочный документ (например, описывающий более широкий контекст или содержащий дополнительные ссылки на другие справочные документы). Атрибут предназначен для помощи пользователям, которые немного запутались
contents	Ссылка на документ, содержащий оглавление документа или сайта
alternate	Указывает альтернативную версию документа. При использовании вместе с атрибутом hreflang предполагается, что расположенный по ссылке файл — это перевод данного документа. При использовании вместе с атрибутом media предполагается, что по ссылке находится версия документа для конкретного носителя, например, для вывода на печать. В комбинации с stylesheet этот атрибут означает, что пользователь может выбрать для данной страницы альтернативную таблицу стилей
copyright	Ссылка на информацию об авторских правах, касающихся данного документа или сайта

Значение	Описание
glossary	Документ с определениями терминов, используемых в актуальном документе
icon	Фавикон страницы или сайта
apple-touch-icon	Определяет, какая пиктограмма будет отображаться на пользовательском экране, когда приложение добавляется в закладки и появляется на панели закладок на экране
apple-touch-startup-image	Загрузочная картинка, отображаемая при запуске веб-приложения. Она особенно полезна, когда ваше веб-приложение находится в режиме офлайн. Если такое изображение отсутствует, некоторые браузеры отображают скриншот веб-приложения по состоянию на момент последнего обращения к нему

Атрибуты `apple-touch-icon` и `apple-touch-startup-image` поддерживаются в операционных системах iOS и Android. В Windows Phone вместо пиктограмм используются плитки. Для включения такой информации нужен следующий код:

```
<meta name="msapplication-TileColor" content="#<color>"/>
<meta name="msapplication-TileImage" content="<image reference>"/>
```

<style>

Элемент `<style>` позволяет оформлять документ с помощью стилей. В отличие от стилей, импортируемых с помощью элемента `<link>`, те стили, которые включены в тег `<style>` в верхней части страницы, действуют только на этой странице. Поэтому к ним закрыт нативный доступ из кэша, и другие документы не могут использовать эти стили. В отличие от стилей, добавляемых к элементу с помощью атрибута `style` (такой стиль влияет лишь на тот элемент, к которому он применен, по крайней мере пока не начнет поддерживаться атрибут `scoped`), стили из элемента `<style>` применимы во всем документе к тем элементам, которым соответствуют указанные селекторы.

Ранее с элементом `<style>` требовалось обязательно использовать атрибут `type` (как правило, `type="text/css"`). В HTML5 такой атрибут можно опустить, но значение `"text/css"` подразумевается по умолчанию. Подобно `<link>`, тег `<style>` принимает атрибут `media`.

Атрибут `scoped` — новинка HTML5, на момент написания книги он не поддерживался ни в одном браузере. Добавляя к элементу `<style>` атрибут `scoped`, мы приказываем агенту рендеринга задействовать данную CSS только в указанной области применения (scope). Область применения — это тот раздел документа, где действует конкретный стиль. Этот атрибут будет полезен при создании виджетов на сайтах (возможно, вы не будете контролировать эти виджеты). Он будет гарантировать, что CSS вашего виджета не смогут случайно заменить собой CSS того сайта, на котором находится виджет.

Элемент <style> и производительность мобильных устройств: антипаттерн. Вот уже на протяжении 12 лет рекомендуется включать стили, действующие в пределах всего сайта, в элемент `<link>`, а не в элемент `<style>`. Включая CSS в элемент `<style>`, мы, возможно, снижаем количество HTTP-запросов, но не можем пользоваться кэшированием. Очевидно, что такой подход не является оптимальным.

Из-за проблем с задержками, обусловленными обработкой лишних HTTP-запросов, в мобильной среде возник специфический антипаттерн.

Для сокращения длительности задержек CSS внутристрочно встраиваются в основной отклик и заключаются в один или несколько тегов `<style>`. С помощью JavaScript содержимое блоков `style` извлекается и сохраняется в `LocalStorage`, а значения ключей добавляются к строке cookie. В дополнительных HTTP-запросах содержатся cookie и имена стилей (а также других ресурсов), хранимых в `LocalStorage`. На стороне сервера cookie считываются и проверяется, какие ресурсы по-прежнему нужны (и есть ли такие ресурсы вообще). Затем в `LocalStorage` отсылаются сразу все файлы, которые там пока отсутствуют, они внедряются в один общий отклик. В результате при первом запросе требуется загрузить значительное количество информации, а информационные порции в последующих запросах получаются гораздо меньше. В данном случае и первичная загрузка сайта, и все последующие операции его перезагрузки оказываются сосредоточены всего в одной паре «HTTP-запрос — HTTP-отклик».

Хотя описанный подход и является антипаттерном, снижение количества HTTP-запросов позволяет значительно повысить производительность. Такая оптимизация может даже компенсировать издержки, связанные с отправкой очень крупного файла (ведь весь его контент нам бы рано или поздно понадобился), а также затраты на обращение к данным, сохраненным в `LocalStorage`.

Добавление тега <script> на веб-страницу

Тег `<script>` позволяет включать в веб-страницу блоки кода JavaScript либо ссылаться на внешний файл, написанный на JavaScript или другом языке сценариев. Атрибут `type` считался обязательным в XHTML и практически всегда был равен `type="text/javascript"`. В HTML5 по умолчанию предполагается, что `type` равен `text/javascript`. Если ваш сценарий действительно написан на JavaScript, атрибут `type` следует опускать. Кроме того, атрибут `language` также считается вышедшим из употребления.

При включении `src` может показаться, что элемент пуст, так как между открывающим и закрывающим тегами нет никакого текста. Так или иначе, обязательно указывайте закрывающий тег и *не записывайте* никакого JavaScript-кода в этой паре тегов.

Советы по повышению производительности JavaScript. Хотя мы обсуждаем элементы, расположенные в теге `<head>` нашего документа, необходимо отметить, что элемент `<script>` встречается как в теге `<head>`, так и в теге `<body>`. Нередко бывает наиболее уместно расположить элемент `<script>` именно в конце элемента `<body>`.

Почему? Обычно синтаксический анализ JavaScript выполняется сразу же после загрузки страницы. В результате загрузка документа замедляется, так как сначала код JavaScript должен загрузиться, пройти синтаксический анализ и выполниться. В результате фактическое время загрузки страницы значительно увеличивается. Чтобы страница загружалась быстрее, рекомендуется располагать сценарии ближе к концу документа, а не в элементе `<head>`.

Считайте, что JavaScript на устройствах пользователей отключен до тех пор, пока не закончатся загрузка, синтаксический анализ и выполнение всех остальных компонентов вашей веб-страницы. Ведь гораздо разумнее предложить пользователям какой-либо контент, когда загрузка еще продолжается, а не заставлять их таращиться в пустой экран. Вот почему так важны производительность JavaScript и порядок расположения элементов в исходном коде.

Есть два атрибута, которые позволяют изменить последовательность выполнения JavaScript: `defer` и `async`. Оба они являются логическими, причем `async` появился только в HTML5. Атрибут `async` означает, что сценарий должен выполняться асинхронно, когда появляется возможность его выполнить. Атрибут `defer` указывает, что сценарий должен быть выполнен после того, как закончится синтаксический анализ документа. Если оба эти атрибута отсутствуют, то JavaScript подвергается синтаксическому разбору, как только браузер встретит такой код на странице. Оба атрибута могут использоваться лишь с внешними сценариями, но не со встроенными.

Как было указано ранее, синтаксический анализ JavaScript выполняется сразу же после того, как браузер встретит код на этом языке (кроме случаев, когда в коде есть атрибуты `async` и/или `defer` и эти атрибуты поддерживаются браузером). Браузер приостанавливает загрузку остальных элементов с сервера до тех пор, пока JavaScript не загрузится полностью, а также не пройдет синтаксический анализ и не будет выполнен. Если же включить JavaScript в конце документа, то *воспринимаемое* время загрузки значительно сокращается. Когда сценарий находится в элементе `<head>`, вся страница подвисает на время загрузки и выполнения JavaScript. Когда элемент `<script>` расположен в нижней части страницы, на его загрузку и выполнение тратится ровно столько же времени, как если бы он был сверху, но никаких видимых пробуксовок при загрузке не наблюдается.

В HTML5 появилось новое решение, помогающее справляться с зависанием пользовательского интерфейса из-за медленного кода JavaScript. Веб-работники, описанные в главе 6, обеспечивают параллельное выполнение JavaScript во множестве потоков. Другой прием, помогающий повысить производительность, — динамическое генерирование тега `<script>`.

Когда JavaScript отключен, отображается элемент <noscript>

Существует элемент `<noscript>`, содержимое которого отображается лишь в случае, когда в пользовательском браузере отключен JavaScript. Вообще рекомендуется постепенно совершенствовать статический функционал страницы, чтобы тег `<noscript>` рано или поздно окончательно устарел. Однако некоторые менеджеры проектов настаивают на том, что все детали сайта должны целиком зависеть от JavaScript. В таких случаях можно использовать в теле документа тег `<noscript>`, чтобы подсказать пользователю, для работы с какими функциями сайта нужно обязательно включить JavaScript. По умолчанию JavaScript работает во всех мобильных и других современных браузерах, в частности в мобильных браузерах WebKit, Blink, Opera Mobile, Windows и Firefox.

Тело документа (элемент <body>)

Тег <body> всегда является вторым и последним прямым потомком элемента <html>. Первый прямой потомок <html> — элемент <head>. Все, что пользователь видит в основном окне браузера, находится в элементе <body>. Рассматривая элемент <head>, мы узнали, что в нем содержатся все метаданные страницы. Элемент <body> включает в себя весь видимый (а иногда и невидимый) контент.

В следующей главе мы поговорим о тех элементах, которые отображаются на клиенте: о <body> и обо всех его потомках. Как уже упоминалось при обсуждении обязательных элементов, <body> не обладает никакими специфическими атрибутами, зато имеет множество обработчиков событий, в частности:

- onafterprint;
- onbeforeprint;
- onbeforeunload;
- onblur;
- onerror;
- onfocus;
- onhashchange;
- onload;
- onmessage;
- onoffline;
- ononline;
- onpopstate;
- onredo;
- onresize;
- onstorage;
- onundo;
- onunload.

Эти обработчики событий лучше помещать во внешнем файле JavaScript, *а не в самом* элементе <body>. Я упоминаю их здесь, просто чтобы вы знали об их наличии, но настоятельно рекомендую и убедительно прошу полностью разделять в коде контент, представление и поведение.

А теперь поговорим о контенте веб-страниц.

3 Новые элементы, появившиеся в HTML5

В HTML5 появилось немало новых элементов, большинство из которых являются семантическими. Назначение некоторых старых элементов было пересмотрено, другие элементы признаны устаревшими. Как мы знаем из главы 2, у нас есть корневой элемент <html>, метаданные документа, описываемые в разделе <head>, а также сценарные элементы. В HTML5 в нашем распоряжении есть следующие виды элементов: секционирующие элементы, заголовочные элементы, фразовые элементы, встроенные элементы и интерактивные элементы. Интерактивные элементы форм рассмотрены в главе 4. Встроенные элементы, относящиеся к работе с медиа, обсуждаются в главе 5. Мы не будем отдельно останавливаться на табличных элементах, так как они перешли в HTML5 практически без изменений. Остальные элементы рассмотрены в следующем разделе.

В предыдущих спецификациях элементы HTML относились к одной из двух категорий: *строковые*, передававшие семантику на уровне текста, или *блочные*, отвечавшие за группирование контента. В HTML5 термины «строковый» и «блочный» не применяются. Авторы HTML5 верно полагают, что представление элементов выполняется средствами CSS и все браузеры, в частности мобильные, работают с таблицами стилей, отвечающими за отображение элементов. Итак, HTML5 больше не классифицирует элементы как строковые или блочные. Но таблицы стилей, применяемые по умолчанию в пользовательских агентах, оформляют одни элементы как строковые, другие — как блочные, просто эта разница теперь не регламентируется в спецификации.

В HTML5 сохранилось большинство элементов из HTML 4, а также появились новые. Кроме того, в HTML5 добавились некоторые новые атрибуты, а также в основном были удалены презентационные элементы и атрибуты, чьи функции лучше реализуются на уровне CSS. В главе 2 были рассмотрены новые глобальные атрибуты. В этой главе поговорим о большинстве новых элементов HTML5, а также о некоторых элементах из предыдущих версий, которые претерпели значительные изменения. Другие элементы — в частности, элементы форм и встроенные элементы — будут рассмотрены в отдельных главах, посвященных именно им.

Секционирующие элементы в HTML5

Корневым секционирующим элементом является тег <body>. Другие секционирующие элементы появились еще в HTML 4 и включены в спецификацию HTML5. Таковы, например, редко используемый элемент <address> и разноуровневые заголовки от <h1> до <h6>. В HTML5 добавлено несколько новых секционирующих элементов, в частности[1]:

- section;
- article;
- nav;
- aside;
- header;
- footer.

Поддерживаются также следующие старые секционирующие элементы:

- body;
- h1–h6;
- address.

Новые секционирующие элементы заключают в себе контент, определяющий область применения заголовков и нижних колонтитулов страницы. Новые секционирующие элементы, в частности <footer>, <aside> и <nav>, не заменяют элемент , а предлагают для него семантические альтернативы. Итак, секционирующие элементы определяют область применения заголовков и нижних колонтитулов страницы, например, относится этот заголовок лишь к части страницы или ко всему документу. Каждый секционирующий элемент с контентом потенциально может иметь собственную структуру. В секционирующем элементе может располагаться запись блога, в которой выделяются собственная область заголовка и «подвал». Кроме того, в документе может быть и несколько заголовков, и несколько нижних колонтитулов. На самом деле каждый section и даже каждый blockquote в документе может иметь собственный footer. Поскольку в каждой секции есть своя область применения заголовков, вы можете использовать на странице несколько заголовков <h1>. Более того, ваши возможности не ограничены шестью заголовками (от <h1> до <h6>). Итак, рассмотрим новые секционирующие элементы.

Авторы HTML5 изучили миллиарды документов, учли каждое имя класса, чтобы узнать, как веб-разработчики обычно называют различные разделы (секции) страницы. Компания Opera выполнила аналогичное исследование, в котором были рассмотрены не только имена классов, но и ID. Из-за широкого распространения Dreamweaver и Microsoft оказались исключительно популярны названия

[1] Элемент hgroup, добавленный в первый вариант спецификации HTML5, уже считается устаревшим. Он пока присутствует в спецификации WHATWG, но уже удален из спецификаций W3C HTML5 и W3C HTML5.1.

style1 и MsoNormal. Отбросив программно сгенерированные классы и такие имена классов, которые явно подбирались по презентационным соображениям, в частности left и right, исследователи обнаружили, что разработчики активно пользуются названиями с «секционирующей» семантикой, например main, header, footer, content, sidebar, banner, search и nav. Эти названия встречались настолько часто, как будто были встроены в стандартный шаблон Dreamweaver (но их не было в этом шаблоне).

Поэтому на основании сложившейся практики разработки в HTML5 было добавлено более 25 новых элементов. Изначально отсутствовали элементы main или content. Почему? Дело в том, что вся информация, не относящаяся к области навигации, боковым панелям, заголовкам или нижним колонтитулам, относится к основному контенту. Элемент <main> появился в спецификации позже остальных, он будет описан в дальнейшем.

Новые элементы, в частности <header> и <footer>, заменяют более нейтральные и , точнее выражая соответствующую семантику. С их помощью можно более информативно описывать стандартную веб-разметку. Новые элементы (рис. 3.1) образуют типичный макет страницы, но при этом семантически его обогащают.

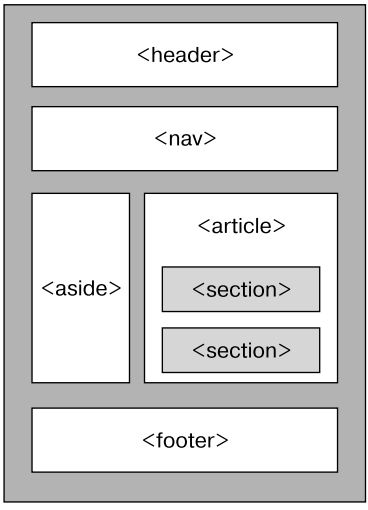


Рис. 3.1. Типичный макет веб-страницы, в котором используются секционирующие элементы HTML5

<section>

Элемент <section> может использоваться для тематического группирования контента — обычно с помощью заголовка. Элемент <section> представляет собой базовый документ или раздел приложения, но при этом не является базовым контейнерным элементом, так как имеет семантическое значение. Если вы группируете

элементы лишь в порядке оформления, пользуйтесь несемантическим элементом [1]:

```
<section>
  <header>
    <h1>Мобильные веб-приложения с помощью HTML5 и CSS3</h1>
  </header>
  <h2>HTML5</h2>
  <p>Немного о HTML5.</p>
  <h2>CSS3</h2>
  <p>Немного о CSS3.</p>
  <footer>Provided by Standardista.com</footer>
</section>
```

<article>

Элемент <article> похож на <section>, но, подобно новостной статье, может восприниматься независимо от сайта или документа, в котором находится. Элемент <article> — это компонент страницы документа, приложения или сайта, который обладает самодостаточной композицией и предназначен для независимого распространения или многократного использования — например, при синдикации:

```
<article>
  <header>
    <h1>Мобильные веб-приложения с помощью HTML5 и CSS3</h1>
    <p><time datetime="2013-11-11T12:31-08:00">11.11.13</time></p>
  </header>
  <h2>HTML5</h2>
  <p>Немного о HTML5.</p>
  <h2>CSS3</h2>
  <p>Немного о CSS3.</p>
  <footer>
    <p>Provided by Standardista.com</p>
  </footer>
</article>
```

Изначально в спецификациях HTML5 присутствовал логический атрибут pubdate, но позже решено было от него отказаться, так как для предоставления информации, соответствующей этому атрибуту, можно использовать словарь микроданных.

Сравнение <section> и <article>

В сообществе авторов спецификаций нередко поднимается вопрос о том, что два этих элемента слишком похожи. Но следует применять именно <article>, а не

[1] Поскольку этот элемент обладает семантическим значением, экранные дикторы указывают начало и конец каждого элемента <section>. Если контент вашей страницы не состоит из семантически значимых разделов, пользуйтесь несемантическим элементом , который никак не акцентируется в экранных дикторах.

<section>, если заключенный в нем контент отличается значительной самостоятельностью. Зачастую такое решение бывает субъективным. Вся разница между двумя фрагментами кода, приведенными в предыдущем разделе, заключается в том, что в примере с <article> я добавила опциональный элемент <time>.

Рассуждая о сходствах, различиях и функциональности двух этих элементов, проведем аналогию с воскресной газетой (если некоторые молодые читатели уже не помнят, что такое газета, то подскажу: это такая бумага с буквами, которой иногда разжигают камин). В воскресной газете есть несколько разделов: передовица, новости, информация о недвижимости, раздел объявлений, еженедельное приложение, кроссворд и т. д. В большинстве разделов есть статьи. У статей есть заголовки, а у некоторых материалов, в частности у сложных новостных репортажей, есть вложенные разделы. На сайте, как и в воскресной газете, статьи и разделы могут быть вложены внутрь других статей и разделов.

Например, в элементе <article> могут находиться запись для форума, газетная или журнальная статья, пост для блога, пользовательский комментарий, интерактивный виджет или гаджет, а также любой другой независимый элемент содержимого.

Следует руководствоваться таким правилом: элемент <section> уместен лишь в случаях, когда его содержимое четко вписывается в структуру документа — как подраздел «Сравнение <section> и <article>» вписывается в содержание этой книги. Так, если бы я опубликовала эту книгу онлайн, то все разделы о секционирующих элементах, которые вы сейчас читаете, находились бы в тегах <section>.

Кроме того, в виде секций были бы оформлены главы, различные страницы-закладки в окне с закладками либо нумерованные разделы диссертации. Домашнюю страницу сайта удобно разбивать на секции, в которых находятся введение, новости и контактная информация.

Рекомендуется использовать элемент <article>, а не <section>, если планируется использовать содержимое элемента для синдикации.

<nav>

Элемент <nav> используется для создания в документе больших навигационных блоков. Как правило, в нем находится фрагмент, содержащий гиперссылки на другие страницы сайта или разделы длинной страницы. При создании основной навигационной панели в такие элементы можно поместить раскрывающиеся меню или другие крупные группы ссылок. Если посетитель слушает информацию с вашего сайта (пользуется экранным диктором), то он, вероятно, захочет пропустить такие группы ссылок и перейти к основному содержимому страницы.

Небольшие группы ссылок, в частности содержащие юридическую информацию, можно поместить в нижнем колонтитуле страницы, заключив в элемент <footer>. Если у вас в элементе <footer> находится крупный навигационный раздел, то в этом элементе можно сделать вложенный <nav>.

Заключая навигационный раздел сайта в элемент <nav>, вы подсказываете слабовидящим посетителям (а также поисковым роботам, которые индексируют сайт в поисковых системах), что здесь находится «оглавление» вашего ресурса. Когда

элемент <nav> будет хорошо поддерживаться в экранных дикторах, мы сможем отказаться от ссылок «пропустить навигацию», которые до сих пор используются для обеспечения доступности. Если вы хотите оперативно принять дополнительные меры для повышения доступности, используйте атрибут WAI-ARIA role="navigation":

```
<nav role="navigation">
  <ul>
      <li><a href="/">Домой</a></li>
      <li><a href="/css/">CSS3</a></li>
      <li><a href="/html/">HTML5</a></li>
      <li><a href="/js/">JavaScript</a></li>
      <li><a href="/access/">Доступность</a></li>
  </ul>
</nav>
```

<aside>

Элемент <aside> содержит контент, который косвенно относится к основной информации. То есть связь с основной информацией достаточна для того, чтобы акцентировать содержимое <aside>, но в полной мере этот фрагмент нельзя считать частью основного контента. Фрагмент <aside> довольно четко отграничен от основного контента и может применяться для оформления типографских эффектов, в частности боковых колонок или цитат, для рекламных блоков, группирования навигационных элементов и подачи прочего содержимого, которое лишь косвенно касается основного контента страницы. Придерживайтесь такого правила: если <aside> можно убрать со страницы, а основной контент после этого воспринимается ничуть не хуже, то в разметке следует применить именно этот элемент.

Содержимое <aside> совсем не обязательно переносить на боковую панель. Например, если в нижней части вашего документа содержится не только типичный «подвальный» контент и в данном случае уместно говорить о *расширенном нижнем колонтитуле*, то <aside> можно расположить в нижней части страницы вместо <footer> или в дополнение к нему:

```
<section>
  <h1>......</h1>
  <!-- Основное содержимое страницы -->
<section>
<aside>
  <dl>
    <dt>HTML5</dt>
    <dd>Следующая серьезная переработка стандарта HTML.</dd>
  </dl>
</aside>
```

<header>

В элементе <header> обычно группируется вводная или вспомогательная навигационная информация, а также содержится заголовок раздела (один из элементов от

<h1> до <h6>). Элемент <header> может применяться также в качестве обертки для оглавления раздела, поисковой строки или любого логотипа — словом, содержать любые компоненты типичного заголовка.

На странице может присутствовать несколько элементов <header>. В первую очередь это основной <header>, представляющий собой область заголовка всего документа. Здесь находятся логотип, основная навигационная панель, названия вашего сайта и учетной записи. Кроме того, на странице могут присутствовать отдельные элементы <header> для каждого блока кода <section> и/или <article>. Например, в вашем блоге может быть общий заголовок документа, содержащий логотип, строку поиска, теглайн и основную навигационную панель. В свою очередь, для каждой записи в блоге можно создать отдельный элемент <header> и поместить в нем, к примеру, название записи, имя автора и дату публикации.

ПРИМЕЧАНИЕ

Считайте элемент <header> семантической заменой основного заголовка и множественных заголовков разделов , оставшихся в прошлом.

<footer>

Элемент <footer> обычно содержит второстепенную информацию о тех разделе или статье, к которым относится. Здесь могут быть указаны имя автора, ссылки на связанные документы, информация об авторских правах и т. д. Как и в случае с <header>, в документе может присутствовать несколько элементов <footer>. Один из них представляет собой общий нижний колонтитул, а остальные играют такую же роль в отдельных элементах <section> и/или <article>. Здесь могут находиться ссылки на социальные сети или ссылка для того, чтобы оставить комментарий под записью в блоге.

Элемент <footer> должен заключать в себе нижний колонтитул того секционирующего элемента, который является непосредственным предком <footer>. Аналогично <header> каждый <footer> ассоциирован со своим ближайшим предком — <article>, <section> или <body>. Если непосредственным секционирующим элементом-предком является <body>, то <footer> представляет собой область нижнего колонтитула для всего документа. Он заменяет распространенный ранее код , добавляя семантическое значение.

Контактные данные автора записываются в элементе <address> (описан чуть позже), который также может находиться внутри <footer>. Нижние колонтитулы не обязательно должны стоять в конце документа или статьи, но обычно они располагаются именно там. В области нижних колонтитулов можно помещать приложения, предметные указатели и им подобный контент:

```
<footer>
  <p>Copyright 2013
    <address>estelle@standardista.com</address>
  </p>
</footer>
```

Обратите внимание: секционирующие элементы <footer> и <aside> обладают одной особенностью — их заголовки не включаются в структуру документа.

Область заголовка и нижний колонтитул игры CubeeDoo

Программируя игры для мобильных устройств, мы стремимся освободить как можно больше места для игрового поля. Однако, если пользователь будет играть на ПК, нам потребуется чем-то заполнить обширное пустое пространство! Поэтому, если игра открыта на большом экране, в зависимости от размера окна браузера мы будем помещать над игровым полем область заголовка, а под ним — нижний колонтитул:

```
1  <article>
2    <header>
3      <h1>CubeeDoo</h1>
4        <h2>Мобильная комбинационная игра</h2>
5    </header>
6    <section id="game" class="colors">
7      <div id="board" class="level1">
8      <!-- Здесь находится игровое поле -->
9      </div>
10        <footer> <!-- Нижний колонтитул раздела -->
11          <div class="control scores">Счет</div>
12          <div class="control menu" id="menu">Меню</div>
13          <ul>
14            <li id="level">Уровень: <output>1</output></li>
15            <li id="timer">
16              <div id="seconds"></div>
17            </li>
18            <li><button id="mutebutton">Звук</button></li>
19            <li id="score">Счет: <output>0</output></li>
20          </ul>
21        </footer>
22    </section>
23        <footer><!-- Нижний колонтитул элемента article -->
24        <ul>
25          <li><a href="about/estelle.html">Об Эстель</a></li>
26          <li><a href="about/justin.html">О Джастине</a></li>
27        </ul>
28        </footer>
29  </article>
```

Мы расположили страницу с игрой в элементе `<article>` (строки 6–22), причем в элемент `<article>` вложен элемент `<section>`. У документа есть область заголовка `<header>` (строки 2–5) и нижний колонтитул `<footer>` (строки 23–28). В области заголовка «статьи» у нас есть ее название и подзаголовок, содержащиеся в элементах `<h1>` и `<h2>` соответственно. На этой странице два нижних колонтитула. Кроме основного колонтитула, который действует во всем документе и отображается на всех страницах нашего сайта, есть и нижний колонтитул игрового экрана (строки 10–21). Этот элемент `<footer>` заключен в элемент `<section>`.

Элемент <address>: появился давно, но используется нечасто

Элемент <address> — не новинка в HTML5. Он применяется уже довольно давно и неплохо поддерживается, в большинстве пользовательских агентов его содержимое отображается курсивом. Но разработчики его почти не используют, поэтому я и напоминаю вам здесь о его существовании. Кстати, элемент <address> не предназначен для контактной информации.

В отличие от семантики большинства элементов, значение элемента <address> не становится очевидным из его названия. Можно сказать, что в элементе <address> находится «контактная информация непосредственного элемента-предка типа <article> или <body>». Если речь идет об элементе <body>, то эта контактная информация касается документа в целом. В случае с элементом <article> контактная информация касается лишь этого элемента <article>. Но здесь не следует указывать адрес вашей организации или страницу с контактными данными, хоть это и кажется логичным.

Группирование контента: другие новые элементы HTML5

Большинство элементов, которые ранее относились к блочным, теперь разделены на две группы: секционирующие и группирующие. Группирующие элементы — это различные списки, а также <p>, <pre>, <blockquote> и . Появились три новых элемента: <main>, <figure> и <figcaption>. Элемент <hr> получил в HTML5 семантическое значение и теперь также включается в новую группирующую категорию, тогда как старый элемент <hr> не имел семантического значения и ничего не группировал (табл. 3.1).

Таблица 3.1. Группирующие элементы

Новые группирующие элементы	Старые группирующие элементы
main	p
figure	pre
figcaption	blockquote
hr (статус изменен)	ol
	li
	ul
	dl
	dd
	dt
	div

Здесь мы рассмотрим только новые группирующие элементы, а также изменения, произошедшие со старыми. Со старыми элементами, которые не претерпели никаких изменений, вы должны быть уже знакомы.

<main>

Элемент <main> определяет основной контент страницы. Поскольку это главная информация, на каждой странице может быть только один элемент <main> (он является уникальным). Элемент <main> не секционирует контент, как элементы, описанные ранее, поэтому не оказывает никакого влияния на структуру документа. Элемент <main> должен заключать в себе контент, уникальный для данной страницы, но в нем не должно быть деталей, располагаемых на всех страницах сайта, например заголовка, нижнего колонтитула и основной навигационной панели.

Поскольку элемент <main> включает в себя основной контент, он может быть предком, но не может быть потомком более мелких секционирующих элементов, содержащих статьи, отступления, заголовки, нижние колонтитулы и навигацию. Нет, я себе не противоречу. Вернемся к примеру с блогом. Элемент <main> на странице блога может содержать записи блога, а также заголовок и нижний колонтитул конкретной записи. Но в нем не будет заголовков, нижних колонтитулов, навигации и отступлений, отображаемых на всех страницах сайта.

<figure> и <figcaption>

Элемент <figure> заключает в себе потоковый контент, который может сопровождаться заглавием <figcaption>. Это заглавие обычно является самодостаточным, как правило, на него можно ссылаться из основного содержимого документа как на отдельно взятый компонент.

Согласна, предыдущий абзац написан на жаргоне W3C. Как это все переводится на человеческий язык? Помните, как сложно всегда было добавить заглавие к картинке? Теперь для этого есть семантический способ.

Если какой-то контент можно удалить со страницы, не исказив при этом смысла остальной информации, такой контент следует помещать в элемент <figure>. Например, в большинстве глав этой книги есть таблицы и рисунки с названиями. Если бы мы вырезали эти таблицы и рисунки, то текст книги, возможно, стал бы менее понятен, но его структура не претерпела бы изменений. Именно рисунки и таблицы в веб-документе особенно удобно помещать в элементе <figure>:

```
<figure>
  <img src="madeupstats.jpg" alt="Доли рынка для шоколада
    в США: 27% — темный шоколад, 70% — молочный шоколад
    и 3% — белый шоколад." />
  <figcaption>Статистика браузера по типу шоколада</figcaption>
</figure>
```

В предыдущем примере были использованы как <figure>, так и <figcaption>, а также выдуманная статистика.

Элемент <figcaption> может быть только первым или последним потомком элемента <figure>. В элементе <figcaption> может содержаться либо заглавие, либо легенда того элемента <figure>, в который вложен <figcaption>. Строковое содержимое <figcaption> является заглавием того элемента <figure>, с которым оно ассоциировано.

<hr>

Пустой элемент <hr> приобрел в HTML5 новое семантическое значение. Обычно его используют исключительно в качестве презентационного элемента — как горизонтальную линейку, он и семантически трактуется как «тематический разрыв на уровне абзаца» или «конец абзаца» (жесткий перенос). Элемент <hr> удобен для обозначения смены контекста (сюжета) или перехода к новой теме в разделе викидокумента.

Все атрибуты, которые были специфичны для элемента <hr/>, признаны нежелательными.

Изменения в атрибутах элементов <li> и

Кроме изменений глобальных атрибутов и атрибутов интернационализации, о которых шла речь в главе 2, в данной версии языка вновь появились два атрибута, некоторое время считавшиеся нежелательными: value для элементов списков и type для упорядоченных списков. Кроме того, для упорядоченных списков добавился логический атрибут reversed, обращающий порядок нумерации и обеспечивающий нумерацию списков в обратном порядке.

Новые текстовые семантические элементы, появившиеся в HTML5

В HTML5 присутствует более 20 семантических элементов, действующих на уровне текста. Некоторые из этих элементов являются новыми, другие переопределены, у третьих имеются новые атрибуты, четвертые сохранились без изменений, и лишь немногие, например <acronym>, удалены из спецификации. Семантические текстовые элементы перечислены в табл. 3.2.

Таблица 3.2. Текстовые семантические элементы

Новые	Изменившиеся	Неизменившиеся	Устаревшие
mark	a	q	acronym
time	small	samp	big
ruby	s cite	kbd	center
rt	i b u	sub	font
rp		sup	strike
bdi		bdo	tt
wbr		span	
data		br	
		em	
		strong	
		dfn	
		abbr	
		code	
		var	

<mark>

Элемент <mark> используется для того, чтобы помечать фрагмент текста в зависимости от его релевантности в контексте. «Чего?» — недоуменно спрашивают читатели. Да, <mark> применяется именно для того, чтобы помечать фрагмент текста, а не чтобы акцентировать его важность. Этот элемент удобен для выделения слов из поискового запроса в поисковой выдаче либо того текста в абзаце, на который вы ссылаетесь в следующем абзаце.

Вы когда-нибудь пользовались нативным поиском в браузере Safari для ПК? Помните, как его страница блекнет и сереет и на ней остаются подсвеченными лишь экземпляры искомого термина?

Создавая такой эффект либо эффект просмотра кэшированной страницы при поиске в Google, следует заключать все найденные поисковые результаты в тег <mark>, вот так:

```
mark {background-color: yellow;}
mark:focus {background-color: blue;}
```

Это код CSS. Он дает примерно такой эффект, как показано на рис. 3.2.

Рис. 3.2. При поиске строки в Opera Mobile на экране подсвечиваются экземпляры текста, введенного в поле поиска

Из этого объяснения можно сделать вывод, что этот элемент исключительно презентационный, однако обладает и семантическим значением. Текст в элементе <mark> становится семантическим (а благодаря CSS получает и соответствующее визуальное оформление). Таким образом привлекается внимание к фрагменту текста, который без специальной пометки может восприниматься как незначимый.

С помощью элемента <mark> можно подсвечивать такой фрагмент документа, который важен только в настоящий момент, но не всегда. Допустим, мы ввели поисковый запрос HTML5. Чтобы подсветить искомый термин на результирующей странице и сделать это семантически, нужно написать следующий код, в котором используются элемент <mark> и CSS:

```
<p><mark>HTML5</mark> находится в стадии разработки. Он станет
    следующим основным пересмотренным стандартом HTML. Как и его
    непосредственные предшественники, HTML 4.01 и XHTML 1.1,
    <mark>HTML5</mark> является стандартом для структурирования
    и представления содержимого в Сети.
```

Поисковый функционал можно обогащать, добавляя к нему сценарии. Таким образом можно заключать результаты поиска в теги <mark>, а затем оформлять эти фрагменты информации с помощью CSS, показывая, сколько поисковых результатов оказалось на странице и где они расположены.

<time>

Элемент <time> используется для обозначения конкретной даты или времени. Он дает точное машиночитаемое значение времени, которое может быть проанализировано пользовательскими агентами или повторно использоваться для других целей, в частности для записи в календарь. Мы программируем игру, в данном случае элементы <time> удобно использовать для показа даты и времени в таблице рекордов. Целесообразно также отображать в этом элементе длительность игры или время, оставшееся до конца раунда.

Атрибут datetime перечисляет дату. Если атрибут datetime присутствует, то его значение должно быть валидной строкой для представления даты[1]. Если атрибут datetime отсутствует, то текстовое содержимое элемента должно быть валидной машиночитаемой датой.

А вот фразу: «Я играю в CubeeDoo в субботу по утрам» — не следует включать в элемент <time>. Гораздо логичнее использовать этот элемент во фразе: «Давайте поиграем в следующую субботу в 11:00 (утром хотелось бы выспаться)», ведь здесь есть точное указание даты и времени:

```
<time datetime="2013-11-30T11:00-8:00">next Saturday at 11:00 a.m.</time>
```

В игре CubeeDoo можно было бы показывать в элементе <time> те значения времени, в которые были достигнуты рекорды, делая это при перечислении дат.

---

[1] Значения даты должны быть записаны в машиночитаемом формате. Строки дат описаны в документе http://www.w3.org/TR/NOTE-datetime

Пользователь видит человекочитаемое время, а устройство может узнать машиночитаемое время по атрибуту `datetime`.

<rp>, <rt> и <ruby>

Элемент <ruby>, или ruby-аннотация, позволяет сопровождать фрагменты текста ruby-аннотациями. Эти элементы не имеют никакого отношения к языку программирования Ruby. Ruby-аннотации — это записи или символы, подсказывающие правила произношения восточноазиатских иероглифов (рис. 3.3).

Элемент <rp>, или ruby-скобки, используется для заключения фрагмента ruby-текста в скобки. Такие скобки будут отображаться в браузерах, не поддерживающих ruby-аннотаций. Если же браузер поддерживает ruby-аннотации, то такие скобки будут скрыты. Элемент <rt>, или ruby-текст, помечает отрезок ruby-текста ruby-аннотацией.

Рис. 3.3. Элементы <ruby> и <rt>, применяемые при записи текста на японском языке. Перевод: «Конфуций сказал: "Всему есть предел"»[1]

Следует вместе использовать теги <rt> и/или <rp>: элемент <rp> описывает текст, содержащийся в элементе <ruby>, и/или подсказывает его произношение. Опциональный элемент <rp> определяет, как такое содержимое должно отображаться в браузерах, не поддерживающих ruby-элементов. Мы не будем пользоваться этим функционалом в игре CubeeDoo; кроме того, ruby-элементы лишь частично поддерживаются на мобильных устройствах.

<bdi>

Элемент <bdi> (в отличие от уже имеющегося элемента <bdo>, который переопределяет направление текста) изолирует конкретный фрагмент двунаправленного контента. Этот элемент необходим по причинам, связанным с организацией алгоритма Unicode для двунаправленного отображения текста и с тем, как этот алгоритм обращается со «слабыми» символами. Код <bdo dir="rtl"> инвертирует целую строку, даже если элемент заключает в себе лишь отрезок этой строки. Элемент <bdi> гарантирует, что в обратном направлении будет записан лишь тот текст, который находится между тегами <bdi> и </bdi>. В спецификации режимов записи CSS описаны некоторые новые свойства, в частности `text-combine-horizontal`, позволяющие вынести за пределы HTML презентационные элементы содержимого и реализовать их в CSS, где им и место.

<wbr>

Элемент <wbr> позволяет сделать разрыв строки в содержимом, в котором отсутствуют пробелы. Например, некоторые URL могут быть очень, очень длинными. Настолько длинными, что не уместятся по всей ширине столбца:

---

[1] Источник: https://dl.dropboxusercontent.com/u/1330446/tests/ruby.html

```
<p>
<a href="http://isCubeeDoo.partofhtml5.com/">Это<wbr/>CubeeDoo.<wbr/>
Часть<wbr/>HTML5.com</a>?
</p>
```

Чтобы и обеспечить заверстывание текста, и гарантировать его удобочитаемость, отдельные слова в URL разделяются с помощью элемента <wbr>. Добавьте элемент <wbr>, чтобы указать браузеру, где можно сделать разрыв строки. Элемент <wbr> является пустым и не имеет никаких специфических атрибутов.

Измененные семантические элементы, действующие на уровне текста

Некоторые элементы HTML в HTML5 были изменены. В частности, таковы элементы a, small, s, cite, i, b и u.

Элемент <a> содержит гиперссылку. Этот элемент ненов, но мы опишем его здесь, так как в HTML5 он претерпел определенные изменения. Кроме того, в мобильной среде этот элемент получает некоторые специальные значения, зависящие от атрибута href, сейчас являющегося опциональным[1].

Для начала нужно отметить, что некоторые атрибуты элемента <a> устарели, например name. Чтобы создать внутристраничный якорь (точку привязки), присвойте атрибут id тому элементу, который расположен в документе максимально близко к вашей цели. Затем создайте ссылку на этот элемент, воспользовавшись его атрибутом id. Например, <a href="#anchorid"> — это точка привязки гипертекстовой ссылки, указывающей на элемент с id, равным anchor id. Кроме того, устарели атрибуты shape, coords, charset, methods и rev.

Вновь используется атрибут target элемента <a>, считавшийся нежелательным в XHTML Strict. Появилось несколько новых атрибутов, в частности download, media и ping[2]. Атрибут download означает, что гиперссылка предназначена для скачивания файла. При использовании этого атрибута он получает в качестве значения имя той файловой системы, которая должна использоваться при скачивании информации. Атрибут media принимает в качестве значения список медиазапросов, на которые рассчитано место назначения гиперссылки. Атрибут ping принимает разделенный пробелами список URL, которые должны откликаться (пинговаться) при переходе по гиперссылке, информируя третью сторону о том, что было совершено действие, и не перенаправляя пользователя с этого сайта.

Еще одно изменение, которое элемент <a> претерпел в HTML5, заключается в том, что он может содержать как строковый, так и блочный контент либо, в терминологии HTML5, как секционирующие, так и фразовые элементы. Например, следующая гиперссылка в HTML5 является правильно оформленной:

---

1 Если атрибут href не задан, то элемент <a> является заполнителем гиперссылки.

2 Атрибуты media, ping и download пока обсуждаются, но ожидается, что они будут включены в спецификации.

```
<a href="index.html" rel="next" target="_blank">
  <header>
    <h1>Это мой заголовок</h1>
    <p>Это мой подзаголовок</p>
  </header>
</a>
```

Мобильно-специфичная обработка ссылок

На мобильных устройствах различается несколько типов гиперссылок, которые требуют специальной обработки при отображении в браузере или в почтовом клиенте мобильного устройства[1].

Вероятно, вам приходилось иметь дело со ссылками типа `mailto:`. При выборе такой ссылки на компьютере или смартфоне открывается приложение для работы с электронной почтой, получающее в качестве адреса цель, на которую указывает ссылка. Кроме того, для электронного сообщения можно задать *тему* и *содержимое*.

Ссылки вида `tel:` открываются в телефонных приложениях смартфонов, после чего устройство набирает номер, указанный в качестве цели такой ссылки. В iOS на экране появляется диалоговое окно подтверждения, из которого система перенаправляет вас в телефонное приложение, а там уже автоматически набирает заданный номер. Если ссылка `tel:` нажата в Android, то пользователь переходит прямо в телефонное приложение, где уже введен соответствующий телефонный номер, но набор номера автоматически не начинается. Аналогичным образом обрабатываются ссылки вида `sms:`, используемые для обмена короткими сообщениями.

Возможно, вы не сталкивались с `sms:`-ссылками. Их синтаксис таков:

```
sms:<phone_number>[,<phone-number>]*[?body=<message_body>]
```

Щелкнув на следующей гиперссылке, вы откроете окно с предупреждением. В этом окне система предложит вам выбор: позвонить по номеру, оформленному в виде ссылки, либо отменить эту операцию в iOS. В Android в аналогичной ситуации на экране появится виртуальная клавиатура, на которой уже будет набран номер. Ссылка с SMS открывает приложение для обмена SMS. Не забывайте, что не на всех устройствах предусмотрена возможность обмена SMS:

```
<a href="tel:16505551212">1 (650) 555-1212</a>
<a href="sms:16505551212">Текст</a>
```

Определение телефонного номера по умолчанию выполняется на большинстве устройств. Например, браузер Safari в iPhone автоматически преобразует любой номер, записанный в формате телефонного номера некоторых стран, в гиперссылку, даже если он гиперссылкой не является. Чтобы такого не происходило, вы должны специально приказать устройству этого не делать (см. подраздел «Специфические значения в мобильной среде, определяемые поставщиками» раздела «Синтаксис HTML-элементов и атрибутов» главы 2).

---

[1] Skype также можно запускать из браузера. Подробнее об этом — на http://developer.skype.com/skype-uris

Кроме того, по-своему обрабатываются ссылки Google Maps, YouTube, iTunes и Google Play. Когда на веб-странице или в теле электронного сообщения оказывается обычная ссылка на Google Maps, некоторые устройства при нажатии на эту ссылку открывают картографическое приложение, но не открывают карту в текущем или в новом окне браузера. Распознав такую ссылку, телефон запустит приложение Maps:

```
<a href="http://maps.google.com/maps?q=san+francisco,+ca">Карта Сан-Франциско</a>
```

В iOS при нажатии ссылок на YouTube и на магазин iTunes открываются виджеты YouTube и iTunes соответственно. Ссылки в операционной системе Android на соответствующих устройствах открывают всплывающее окно, в котором система предлагает выбор: перейти по данной ссылке или открыть ссылку в Google Play.

Изменения текстовых элементов по сравнению с HTML 4

Многие из нас были уверены, что презентационные элементы `<i>`, `<b>`, `<s>`, `<u>` и `<small>` обречены на переход в категорию нежелательных. Но все сложилось иначе: получив семантическое значение, они буквально родились заново.

Элемент `<i>` следует использовать вместо `<span>`, чтобы обеспечивать отступ текста от окружающего контента, не делая акцента на таком тексте. Например, так можно оформлять технический термин, идиоматическое выражение из другого языка, мысль персонажа или, например, название корабля.

Элемент `<b>` представляет фрагмент текста, который должен визуально выделяться из окружающей информации, но не иметь при этом никакого дополнительного акцента. Например, таковы ключевые слова в аннотации документа, названия товаров в обзоре или другие текстовые фрагменты, которые принято выделять жирным шрифтом.

Элемент `<s>` заключает в себе контент, который уже неактуален или неточен и поэтому «вычеркнут» из документа. Элемент `<s>` употреблялся уже довольно давно, но до HTML5 имел только презентационное значение. В HTML5 элемент `<s>` имеет семантическое значение.

Элемент `<u>` также получил семантическое значение. Он означает выделение фрагмента текста в окружающем контенте без придания этому фрагменту какой-либо особой важности и без акцента на нем. В типографском тексте такая информация подчеркивается. В частности, обычно подчеркиваются слова, написанные с ошибками, или китайские имена собственные.

Элемент `<small>` обычно используется для представления тех частей документа, которые записываются мелким шрифтом. Текст, находящийся в этом элементе, не обязательно делать более мелким, чем остальной текст в документе, но в смысловом отношении он должен передавать именно такой материал, который традиционно записывается крошечными буквами. Например, это может быть юридическая информация на лотерейном билете или информация о противопоказаниях к лекарственному препарату (хотя, если призадуматься, эту информацию стоило бы писать огромными буквами).

А вот элемент <cite>, на мой взгляд, совсем не подавал признаков выхода из употребления, но в HTML5 он также получил новое значение. Теперь в элементе <cite> записывается только один вид информации: цитируемое название произведения. Например, это может быть заголовок книги, название песни, фильма, телепередачи, судебного иска и т. п. В предыдущих версиях HTML элемент <cite> можно было использовать для обозначения имени собственного. Такое использование этого элемента больше не допускается.

Элементы, которые не изменились

Вероятно, вы знаете значения всех текстовых элементов, которые существовали до HTML5 и не претерпели семантических изменений в новой версии. Однако некоторые элементы используются довольно редко. В табл. 3.3 кратко обобщена информация о существующих элементах.

Таблица 3.3. Неизменившиеся элементы

Элемент	Описание
em	Текст с экспрессивным значением
strong	Текст повышенной важности
q	Текст, процитированный из другого источника
dfn	Термин или определение при первом его появлении
abbr	Аббревиатура или акроним. Обратите внимание: элемент acronym устарел. Полное название должно сопровождаться атрибутом title
code	Фрагмент компьютерного кода
var	Математическая или программная переменная, а также любой заполнитель, который должен быть заменен другим значением
samp	Образец вывода из программы или вычислительной системы
kbd	Представление пользовательского ввода, сделанного с клавиатуры
sub	Нижний индекс
sup	Верхний индекс
bdo	Форматирование направления текста; данный элемент позволяет переориентировать поток текста на странице в противоположную сторону
span	Универсальная несемантическая обертка для фразового контента
br	Разрыв строки

Существуют также элементы <ins> и <del>. Они нужны для редактирования и представляют операции вставки и удаления соответственно.

Встраиваемые элементы

К числу встраиваемых элементов относятся шесть старых и шесть новых. Вот новые элементы[1]:

[1] В настоящее время рассматривается также специальный элемент <picture> для работы с адаптивными изображениями. Черновой вариант спецификации этого элемента приведен по адресу: http://www.w3.org/TR/html-picture-element/

- embed;
- video;
- audio;
- source;
- track;
- canvas.

А вот уже имеющиеся:

- img;
- iframe;
- object;
- param;
- map;
- area.

К числу встраиваемых элементов относятся также медиаэлементы, в частности <video>, <audio>, <source>, <track> и <canvas>, о которых мы поговорим в главе 5. Еще один «новый» элемент — <embed>. Он был впервые реализован уже много лет назад, но никогда не входил в спецификации HTML 4 и XHTML. Мы обсудим и этот элемент, и те, которые существовали ранее, поскольку некоторые их атрибуты могли устареть.

<iframe>

Элемент <iframe> — не новинка HTML5, но набор его атрибутов изменился. Элемент <iframe> лишился атрибутов longdesc, frameborder, marginwidth, marginheight, scrolling и align, зато приобрел srcdoc, sandbox и seamless.

Значением атрибута srcdoc является код HTML, который используется для создания документа, отображаемого в элементе <iframe>. Теоретически любой элемент, применяемый в теге <body>, может быть значением атрибута srcdoc. Внутри значения srcdoc следует экранировать все кавычки с помощью ", либо это значение преждевременно оборвется. Если браузер поддерживает атрибут srcdoc, то содержимое srcdoc будет использоваться. Если браузер не поддерживает атрибут srcdoc, то вместо него будет отображаться содержимое файла, указанного в атрибуте src:

```
<iframe srcdoc="<p>Learn more about the
<a href=&quot;http://developers.whatwg.org/the-iframe-element.html
#attr-iframe-srcdoc&quot;>srcdoc</a> атрибут."
src="http://developers.whatwg.org/the-iframe-element.html
#attr-iframe-srcdoc"></iframe>
```

Атрибут sandbox позволяет задействовать набор дополнительных переопределяемых ограничений, налагаемых на содержимое элемента <iframe>. Применяя этот атрибут, мы встраиваем контент из внешнего источника, как если бы он подавался

с того же домена, но со значительно более строгими ограничениями. Плагины, формы, сценарии и ссылки на другой браузерный контент в элементе <iframe> с таким атрибутом отключаются. Считается, что содержимое <iframe> получено из отдельного уникального источника, не допускает обхода дерева DOM или считывания данных cookie. Чтобы переопределить эти значения, необходимо задать для атрибута sandbox такое значение, которое соответствует определенному синтаксису.

Ключевое слово allow-same-origin позволяет обращаться с контентом, как будто он получен из того же источника, что и весь контент сайта, а не из отдельного источника. Ключевое слово allow-top-navigation позволяет открывать содержимое документа в вышестоящем браузерном контексте. Ключевые слова allow-forms, allow-pointer-lock, allow-popups и allow-scripts вновь активизируют формы, API pointer-lock для работы с мышью, всплывающие окна и сценарии соответственно. Укажите от нуля и более значений, разделенных пробелами, в зависимости от того, какие именно ограничения вам требуются. Но учтите, что каждое новое значение привносит дополнительный риск для безопасности:

```
<iframe sandbox="allow-same-origin allow-forms allow-scripts"
    src="http://maps.google.com" seamless></iframe>
```

Атрибут seamless обеспечивает «бесшовное» отображение элемента <iframe>, как будто он является нативной частью родительского документа. При поддержке этого атрибута предполагается, что он обеспечит каскадирование правил CSS от родительского элемента к содержимому <iframe>, активизирует работу ссылок для перехода в родительский элемент. Кроме того, при необходимости размер <iframe> должен уменьшаться или увеличиваться, подстраиваясь под параметры родительского элемента, но неизменно сохраняя одно и то же начало координат.

Пустой элемент <img> утратил атрибуты border, vspace, hspace, align, longdesc и name. Обсуждается возможность добавления нового атрибута srcset, который позволил бы использовать в этом элементе альтернативные изображения в зависимости от требуемых высоты, ширины и пиксельной плотности.

Кроме тех случаев, когда изображение является частью контента и необходимо в данном контексте, стоит ограничиваться использованием фоновых картинок. О том, как работать с фоновыми изображениями и предоставлять разные варианты фонового изображения в зависимости от размера устройства и разрешения экрана, мы поговорим в главе 9.

Если вы хотите поддерживать адаптивные изображения переднего плана еще до того, как в браузерах широко распространится поддержка атрибута srcset, элемента <picture> или клиентских подсказок, воспользуйтесь тегом <object>. Применяя его по принципу «автомобиль клоуна»[1], можно отсылать на клиент

[1] Подробное описание этого метода на русском языке приводится в статье http://bot.kz/2013/06/23519. — *Примеч. пер.*

файлы масштабируемой векторной графики (SVG), о которых пойдет речь в главе 5, образующие единое растровое изображение и используемые на основе медиазапросов.

<object>

Элемент <object> требует атрибутов data и type. Некоторые атрибуты, в частности align, hspace, vspace и border, были признаны устаревшими, а их функции теперь реализуются в CSS. Кроме того, признаны устаревшими атрибуты archive, classid, code, codebase и codetype элемента <object>, которые теперь должны использоваться с элементом <param>. Вместо применения старого атрибута declare следует применять элемент <object> с каждым интересующим вас информационным фрагментом. Вместо использования атрибута standby, который уже устарел, оптимизируйте ресурс, чтобы он загружался быстрее, а в случаях, когда это допустимо, — даже постепенно, небольшими частями. Элемент <object> используется нечасто, но поддерживается широко.

<param>

Пустой элемент <param> утратил атрибуты type и valuetype, зато приобрел атрибуты name и value.

<area>

Пустой элемент <area> утратил атрибут nohref, но приобрел атрибуты rel, ping (см. раздел о теге <a>), media и hreflang.

<embed>

Вероятно, элемент <embed/> вам знаком. Но в спецификациях он появился только недавно. Элемент <embed> — это точка для интеграции с контентом, который будет отображаться с применением стороннего плагина, например Adobe Flash Player, а не в нативном браузерном элементе управления, таком как <video> или <audio>. Этот элемент пустой, как и <img>, он должен быть самозакрывающимся. Чтобы добавить URL встраиваемого источника, пользуйтесь атрибутом src. MIME-тип источника задается с помощью атрибута type.

Интерактивные элементы

К числу интерактивных элементов в настоящее время относятся элементы форм, изменившийся элемент <menu>, а также новые элементы <detail>, <summary> и <command>.

<details> и <summary>

Вам доводилось когда-либо создавать такой узел, при щелчке на котором открывается более подробная информация о его содержимом, а при повторном щелчке эти подробности скрываются? Если в браузере поддерживаются элемент HTML5 <details> и его дочерний элемент <details>, то такая функция реализуется нативно, без всякого применения JavaScript (рис. 3.4).

Рис. 3.4. Элемент <summary> остается видимым всегда. Щелкая на нем, можно отображать и скрывать элемент <details>

В элементе <details> удобно заключать поясняющий виджет. В этом виджете пользователь может найти дополнительную информацию или получить новые элементы управления, например контент, который уместно помещать в сносках, концевых сносках или всплывающих подсказках. У элемента <details> есть атрибут open, также появившийся только в HTML5. Если присутствует атрибут open, то содержимое элемента <details> изначально будет видимым. При отсутствии этого атрибута дета-

ли будут скрыты, пока пользователь сам не потребует их отобразить. Элемент <summary> в документе должен быть дочерним для <details>. В текстовом узле должна содержаться следующая информация: аннотация, название или легенда остального содержимого элемента <details>, который является непосредственным предком <summary>. Содержимое элемента <summary> отображается по умолчанию независимо от того, установлен ли атрибут open. Щелкая на <summary>, пользователь может выводить на экран или скрывать остальное содержимое элемента <details>. Такое интерактивное поведение по умолчанию свойственно комбинации элементов <details>/<summary>. Если они поддерживаются, то такие задачи выполняются без применения JavaScript:

```
<details>
  <summary>5 из 5 звезд в трех обзорах</summary>
  <ul>
    <li>5 звезд от Amazon</li>
    <li>5 звезд от Costco</li>
    <li>5 звезд от Barns &amp;amp; Noble</li>
  </ul>
</details>
```

Если эти элементы поддерживаются, все содержимое <details> (кроме <summary>) остается скрытым при отсутствии опционального атрибута open (это происходит по умолчанию). Остальное содержимое должно отображаться только после щелчка на <summary>. Не путайте элемент <summary> и атрибут summary элемента <table>. Элемент <summary>, являющийся потомком <details>, отвечает за переключение остального содержимого элемента <details>, делая его то видимым, то невидимым, как если бы было установлено правило display: none;. Элемент <summary> содержит аннотацию, название или легенду остального содержимого элемента <details>, который является непосредственным предком <summary>.

Поскольку элемент <summary> предназначен для управления открытием/закрытием остального содержимого элемента <details>, он всегда остается видимым (независимо от того, открыт или закрыт элемент <details>). Пока два этих элемента поддерживаются не во всех браузерах, аналогичного эффекта можно достичь с помощью JavaScript. Просто снабдите элемент <summary> слушателем событий, который добавляет к родительскому элементу <details> атрибут open или удаляет его, а также добавьте в CSS следующие стили:

```
details * {display: none;}
details summary {display: auto;}
details[open] * {display: auto;}
```

Если последняя строка вам пока непонятна — не волнуйтесь! Мы поговорим о селекторах атрибутов в главе 8.

<menu> и <menuitem>

Возродился элемент <menu>, считавшийся нежелательным в HTML 4.01 и XHTML. Изначально он определялся как одностолбцовый список меню. В HTML5 элемент

<menu> был переопределен. Теперь он содержит список команд или элементов управления.

В HTML5 в элементе <menu> перечисляются элементы форм. Значение атрибута id элемента <menu> может включаться в код как значение атрибута menu элемента <button> или значение атрибута menuitem элемента <input>. Таким образом создается меню или контекстное меню для элемента формы. Меню может содержать элементы <menuitem>, каждый из которых вызывает срабатывание определенного действия. Если при применении меню оно должно имитировать панель инструментов, то атрибут type устанавливается в значение toolbar. Если нужно создать контекстное меню, то в данном случае применяется атрибут context. Значение атрибута label определяет надпись, которой сопровождается меню. Такие элементы могут быть вложены друг в друга, в результате вы получите многоуровневое меню.

<menuitem>. Элемент <menuitem>, который может находиться только внутри <menu>, определяет командную кнопку или запись в контекстном меню. Тип команды зависит от атрибута type. В качестве значения этот атрибут может иметь radiobutton (выбор одного элемента из списка), checkbox (включение/отключение варианта) или command — для создания кнопки, позволяющей совершать определенную операцию. Хотя может сложиться впечатление, что это типичный элемент формы, он не предназначен для отправки информации на сервер. <menuitem> — это интерактивный элемент, обеспечивающий интерактивные операции с актуальным содержимым страницы.

Элемент <menuitem> является пустым и не имеет закрывающего тега. Обязательно сопровождайте его атрибутом label, значение которого представляет собой надпись, отображаемую пользователю. Другие возможные атрибуты этого элемента — icon, disabled, checked, radiogroup, default и command. Значение атрибута command — это определение команды. Если у элемента присутствует атрибут title, он должен описывать команду.

Задействовав эти теги и атрибуты, вы сможете создавать примерно такие элементы управления, которые открываются при щелчке правой кнопкой мыши, и отображать эти элементы в окружении Windows. Пока на персональных компьютерах поддержка таких возможностей еще остается экспериментальной, а на мобильных устройствах они не поддерживаются.

Весь XHTML — это HTML5, кроме...

Почти все элементы из XHTML доступны и валидны в HTML5. К числу устаревших элементов относятся следующие:

- basefont;
- big;
- center;
- font;
- strike (пользуйтесь <del>);

- tt;
- frame (<iframe> по-прежнему валиден);
- frameset;
- noframes;
- acronym (пользуйтесь <abbr>);
- applet (пользуйтесь <object>);
- isindex;
- dir.

Как было указано ранее, некоторые элементы не устарели, а приобрели более семантическое значение. Так, элементы <b>, <hr>, <i>, <u> и <small> были в предыдущих спецификациях исключительно презентационными, но теперь имеют семантическое значение и определяются независимо от внешнего вида. Элемент <menu> оказался очень удобным для создания панелей инструментов и контекстных меню. Элемент теперь означает «важно», а не «логическое ударение». Элемент <a> может содержать не только строчный, но и блоковый контент. Кроме того, он теперь может обходиться без атрибута href.

Некоторые атрибуты устарели, другие были добавлены. Большинство изменений связаны с элементами веб-форм, которые мы очень подробно обсудим в главе 4. Кроме того, хотелось бы обратить ваше внимание еще на некоторые детали, не упомянутые ранее.

- Элемент <table> больше не имеет атрибутов width, border, frame, rules, cellspacing и cellpadding.
- Элемент <ol> вновь приобрел атрибуты reverse и start.
- Элементы <col> и <colspan> утратили все свои специфические атрибуты, кроме span.
- Набор атрибутов у элементов <td> и <th> уменьшился. Теперь у них есть только атрибуты headers, rowspan и colspan, зато нет abbr, axis, width, align и valign. Область применения теперь отсутствует у <td>, но остается у <th>.
- Элементы <tr> и <thead> не имеют никаких атрибутов, кроме глобальных.

Резюме

Спецификация HTML5 *огромна*. В этом разделе я познакомила вас с синтаксисом и семантикой HTML5, а также некоторыми новыми элементами. Глава задумывалась как быстрый (или не очень быстрый) экскурс по новым элементам HTML5.

Разговор об HTML5 далеко не окончен. В главе 4 поговорим о некоторых замечательных свойствах веб-форм и узнаем, как эти возможности помогают разрабатывать сказочные пользовательские интерфейсы с минимальным применением JavaScript.

4 Веб-формы в HTML5

Если вы, как и я, причисляете себя к фанатам веб-разработки, то самые классные новые возможности, которые вам может предложить HTML5, связаны, пожалуй, с обработкой форм. Да, Canvas сказочный. SVG чудесна. API JavaScript обеспечивают широкие возможности нацеливания на узлы DOM с помощью селекторов. Поддерживается сопоставление с медиазапросами. Не составляет труда добавлять, удалять и переключать имена классов, и для этого совершенно не требуется задействовать JS-фреймворк (если не верите мне, наберитесь терпения — все эти темы мы рассмотрим в дальнейшем). CSS3 обеспечивает быстрое прототипирование любых дизайнерских фантазий. Почему же я отдаю пальму первенства именно формам HTML5? Дело в том, что эти формы значительно повышают удобство работы с ресурсом и снижают зависимость от валидации с применением JavaScript.

В HTML5 мы можем выполнять валидацию форм, а также некоторые другие операции более декларативным способом. HTML5 позволяет разработчикам реализовывать улучшенные возможности работы с формами — например, валидацию и выдачу сообщений об ошибках — без привлечения JavaScript. Поскольку зависимость от JavaScript снижается, сокращается и время, необходимое на разработку и поддержку программы. Жить становится гораздо легче.

Например, чтобы поместить элемент формы в фокус, можно не добавлять метод `setFocus()` к событию `onload`. Теперь у нас есть атрибут `autofocus` (пользоваться которым, однако, не следует[1]). Теперь не приходится программировать множество функций на JavaScript для запрашивания, валидации адресов электронной почты и помещения этих адресов в фокус при ошибке. HTML5 позволяет делать надписи на элементах форм по мере необходимости, обеспечивает нативную валидацию множества типов ввода. Речь идет как о стандартных форматах, таких как формат адресов электронной почты, так и о специальных, которые разработчик может за-

[1] Когда элемент формы оказывается в фокусе, страница «перепрыгивает» к этому полю ввода формы, оставляя за пределами видимости надпись, сопровождающую поле. Это плохо сказывается на доступности сайта, а также на пользовательском восприятии. Проблема особенно остра на небольших устройствах, где надпись может обрезаться. Оба вышеупомянутых метода вызывают определенные проблемы с доступностью, поэтому старайтесь не пользоваться ни одним из них.

давать с помощью регулярных выражений. Кроме того, HTML5 предоставляет пользователям понятные сообщения об ошибках, подробно описывающие суть каждой из них.

Ранее приходилось добавлять к элементам форм множество атрибутов и обработчиков событий. Когда те значения и поведения, которые задаются по умолчанию в HTML5, будут повсеместно поддерживаться и использоваться, мы сможем просто писать `<form>`!

Ранее мы могли задавать в формах лишь очень ограниченное количество типов данных. Атрибут `type` элемента `<input>` мог принимать совсем немного значений. Для большинства полей с данными действовало значение `'text'` независимо от того, какой тип данных ожидался. На мобильных устройствах для ввода таких данных появлялась виртуальная клавиатура QWERTY. Работая с формами HTML5, мы можем указать браузеру, какие типы данных придется принимать, обозначать приемлемые шаблоны вводимой информации, а также давать дополнительные подсказки пользователю. Мобильные браузеры характеризуются улучшенной поддержкой веб-форм HTML5. В частности, на экран выводится такая виртуальная клавиатура, которая содержит символы именно для предполагаемого типа ввода (например, если предполагается вводить телефонный номер, то выводится телефонная клавиатура с цифрами). В составе пользовательских интерфейсов появилось множество продвинутых возможностей — в частности, календари и цветовые палитры.

До появления HTML5 разработчики использовали CSS для описания представления ресурса, JavaScript — для валидации и код базы данных — для обеспечения правильного заполнения того или иного обязательного элемента. Хотя вам непременно требуется и далее разделять три этих зоны ответственности и *всегда* валидировать пользовательский ввод на интерфейсе базы данных, HTML5 позволяет в итоге избавиться от JavaScript в пользовательском интерфейсе или как минимум значительно упростить этот код.

С помощью HTML5 браузер может проверять, использован ли для заполнения обязательных полей правильный тип данных, укладываются ли данные в допустимый диапазон, следуют ли правильному синтаксису и т. д. Если в форму введены неподходящие данные, она не будет отправлена. Эти возможности невероятно круты и обеспечивают практически полную поддержку форм на уровне браузера. В этой главе мы обсудим все эти новые возможности и остановимся на том, поддерживаются ли они уже, или такая поддержка только планируется.

Мобильные устройства в последнее время уже поддерживают некоторые возможности веб-форм HTML5, поддержка других возможностей только начинается. На большинстве сенсорных устройств с виртуальными клавиатурами браузер выводит для пользователя подходящую для ввода клавиатуру с минимальным набором символов. Причем этот набор символов максимально соответствует тому типу ввода, для которого предназначена данная веб-форма. Другие мобильные браузеры обрабатывают большинство новых возможностей веб-форм, содержащихся в пользовательском интерфейсе, а также обеспечивают нативную валидацию. Независимо от того, насколько широко будут поддерживаться веб-формы

на мобильных устройствах к тому моменту, как вы станете писать свой код, вам определенно следует задействовать все возможности работы с веб-формами, поддерживаемые в HTML5. Дело в том, что все эти возможности постепенно совершенствуются. Конечно, в новых браузерах набор возможностей будет шире, но даже самые старые мобильные браузеры, появившиеся в середине 1990-х, отображают на экране элементы HTML5 с довольно высокой степенью доступности.

Чтобы обозначить, что элемент формы рассчитан на получение конкретного типа ввода, этот тип можно указать в атрибуте `type`. Ранее мы были вынуждены работать с очень ограниченным количеством типов полей ввода и преодолевать различные препятствия для валидации их на клиенте перед отправкой информации на сервер. В HTML5 мы не только располагаем гораздо более разнообразными типами полей ввода, но и получаем нативную валидацию многих типов данных. Вскоре устареют тысячи валидационных сценариев JavaScript, которые мы уже успели написать, поскольку вся валидация форм станет нативной и будет происходить в браузере. Пока мы еще не располагаем полной нативной поддержкой всех возможностей, которые будут описаны в следующем разделе, но можем имитировать эти функции с помощью минимального кода на JavaScript. При этом мы можем в полной мере воспользоваться новыми атрибутами и типами полей ввода, продолжая применять селекторы пользовательского интерфейса (см. раздел «Оформление для повышения удобства использования» далее), селекторы атрибутов (см. главу 7), а также зависящие от ввода динамические клавиатуры. Таким образом мы можем самостоятельно улучшить удобство использования во всех современных мобильных браузерах, а поддержка веб-форм HTML5 тем временем будет только совершенствоваться.

Атрибуты <input> (и другие элементы форм)

Прежде чем подробно познакомиться со старыми и новыми типами полей ввода, поговорим о некоторых новых и старых атрибутах элемента `<input>`.

type

Ранее мы говорили о том, что у элемента `<input>` есть только один обязательный атрибут — `type`. Правда, если его опустить, то код все равно будет работать, так как примет значение, задаваемое по умолчанию: `type="text"`.

```
<label>Phone: <input type="tel" name="phone"></label>
<label>Website: <input type="url" name="website"></label>
```

В HTML5 существует 23 возможных значения атрибута `type`. Все они будут рассмотрены в разделе «Типы и атрибуты элемента <input>». Обратите внимание: если браузер не поддерживает новый тип поля ввода, то атрибут `type` по умолчанию получает значение `text`. Все новые типы — это просто усовершенствования. Формы

остаются вполне доступными и в тех браузерах, которые не поддерживают новых типов, поэтому не ждите полной браузерной поддержки новых типов полей ввода и смело реализуйте их в своих формах.

required

Чтобы пометить поле формы как обязательное, пользуйтесь атрибутом `required`. Если пользователь пытается отправить форму, а поле формы, помеченное как `required`, оставлено пустым либо содержит недопустимое значение, то форма не отправится, а фокус перейдет на первый из неверно заполненных обязательных элементов. В браузерах, поддерживающих такую возможность, пользователь получает сообщение об ошибке, например: «Введите значение», если поле оставлено пустым, или «12-12 не соответствует формату, требующемуся на этой странице», когда значение не соответствует необходимому шаблону. Атрибут `pattern`, описывающий такие шаблоны, рассмотрен далее. Могут выдаваться и другие подобные сообщения.

Атрибут `required` валиден с любыми типами полей ввода, кроме `buttons`, `range`, `color` и `hidden`:

```
<label>Email: <input type="email" name="email" required="required" /></label>[1]
<label>Phone: <input type="tel" name="phone" required /></label>
```

Если для элемента формы не выбрано значение, он может и не соответствовать шаблону, которому обычно соответствуют элементы данного типа, — конечно, если этот элемент не является обязательным. Например, тип электронного сообщения, для которого не указан тип, пуст, а значит, не должен соответствовать какому-либо формату для таких сообщений. Однако если атрибут `required` присутствует, то отправка формы не состоится, если обязательное поле будет пустовать или содержать значение в недопустимом формате.

ПРИМЕЧАНИЕ

Совет для профессионалов: те браузеры, которые поддерживают атрибут required, также поддерживают псевдоклассы :required и :invalid. Можно давать пользователю визуальные подсказки, чтобы он видел, какие поля являются обязательными. О том, что ввод данных прошел успешно, можно сообщить также с помощью CSS:

```
input:focus:invalid {
  background-color: #CCCCCC;
}
input:valid {
  background-color: #00FF33;
}
input:required {
  border: 2px solid #0066FF;
}
```

[1] Синтаксис таков: либо просто required, либо required="required", если вы пишете на XHTML Strict.

Селекторы пользовательского интерфейса, используемые в CSS3, будут рассмотрены в главе 7.

ПРИМЕЧАНИЕ

Совет по доступности: чтобы улучшить доступность приложения, сопровождайте каждый атрибут required атрибутом ARIA aria-required="true" (мы поговорим об ARIA, стандарте доступности активных интернет-приложений, в главе 6):

```
<input type="tel" name="phone" required aria-required="true"/>
```

Минимальные и максимальные значения: атрибуты min и max

Чтобы задать диапазон допустимых значений, можно пользоваться атрибутами `min` и `max`.

Атрибуты `min` и `max` можно использовать, в частности, только с двумя типами полей для ввода даты и времени: `number` и `range`. Если в браузере предоставляется виджет пользовательского интерфейса для одного из таких типов полей ввода, то пользователь просто не сможет выбрать значение вне разрешенного диапазона `min`/`max`, поскольку такие значения не будут отображаться в данном виджете.

В браузерах, полностью поддерживающих тип поля ввода `number`, отображается блок с изменяемым числовым значением. Этот элемент работает в диапазоне, ограниченном минимальным и максимальным значениями `min` и `max`. В тех пользовательских интерфейсах, где подобные данные вводятся в свободной форме — допустим, в поле типа `number`, — элемент формы может считаться обязательным. Если в таком обязательном поле оказывается значение, не относящееся к диапазону от `min` до `max`, то браузер не отправит такую форму на сервер.

В поле ввода типа `range` самое левое поле будет установлено в значение `min`, а самое правое — в значение `max` при условии, что `max` больше `min`. Мы обсудим эти свойства ввода в следующем разделе, когда будем подробно исследовать типы полей ввода `number` и `range`.

Минимальные и максимальные значения часто задействуются при валидации форм, поэтому данные атрибуты очень полезны веб-разработчику. Например, если вы пишете систему бронирования билетов, то уже знаете, сколько посадочных мест в вашем распоряжении. Эту информацию можно закодировать в самой форме на странице, поэтому пользователь просто не сможет указать недопустимое время, а значит, не получит грозного сообщения об ошибке. Странице заранее известно, сколько ячеек на ней имеется, пользователю остается лишь правильно выбрать время:

```
<label>Reservation Time:
  <input type="time" min="17:00" max="22:00" name="dinner" required>
</label>
```

Допустим, мы программируем систему заказа мест в ресторане. В этом заведении ужин начинают подавать с 17:00, а в 22:00 заканчивается подача блюд. В браузерах, поддерживающих такую функцию, вы без всякого применения JavaScript сможете гарантировать, что ваша система будет принимать заказы на ужин только на этот период.

step

Атрибут step неявно включается в полях с типами ввода, предусматривающими дату/время — range и number, но при таких типах ввода атрибут step можно определять и явно. Например, если вы пишете программу для магазина мелких товаров, где все цены делятся на 5, а максимальная стоимость товара составляет 1 доллар, то можете включить следующий код в ваш графический пользовательский интерфейс, где задается цена:

```
<p>
  <label for="cost">Price </label>
  <input type="number" min="5" max="100" step="5" name="cost" id="cost"/>
</p>
```

Если в пользовательском интерфейсе предоставляется виджет, например слайдер диапазонов, то при перемещении этого слайдера в направлении возрастания значение на каждом шаге будет увеличиваться на величину атрибута step.

В пользовательском интерфейсе, где допускается ввод данных в свободной форме, например с типом поля ввода number, форма будет отправлена лишь при условии, что введенное значение — допустимый шаг, превышающий заданный минимум. Например, в предыдущем примере значение 7 было бы недопустимым. Если бы минимальное значение было равно 2, а атрибут step равен 5 (как в следующем примере кода), то значение 7 было бы допустимым, а значение 100 — нет:

```
<p>
  <label for="cost">Price </label>
  <input type="number" min="2" max="100" step="5" name="cost" id="cost"
  required/>
</p>
```

ПРИМЕЧАНИЕ

В примерах с атрибутом step я использовал явные метки с атрибутом for. До этого работал с неявными метками. Об атрибуте for мы поговорим далее в этой главе.

placeholder

Вероятно, самая распространенная функция форм, связанная с JavaScript, — запись в поле формы текста-заполнителя. Такой текст дает подсказку или предоставляет инструкцию о том, какой тип ввода ожидается в этом поле. Ранее текст-заполнитель исчезал при попадании поля в фокус, а теперь — при вводе данных. Если информация из поля удалена и оно остается пустым, то в нем вновь появляется текст-заполнитель. Таблицы стилей пользовательского агента оформляют текст-заполнитель как фоновый, чтобы было очевидно, что элемент формы попрежнему пуст. В HTML5 такая функциональность предоставляется изначально, с улучшенной доступностью. Атрибут placeholder решает именно ту задачу, которую обычно выполняла активно используемая, но далеко не всегда легкодоступная функция-заполнитель. В результате доступность форм значительно повышается, а необходимость в применении JavaScript отпадает.

Одно из существенных различий между труднодоступными сценариями и легкодоступным атрибутом placeholder заключается в том, что текст placeholder исчезает при внесении первых изменений, а не при попадании поля в фокус. В большинстве современных браузеров текст placeholder остается на месте до тех пор, пока пользователь не начнет вводить в поле свой текст.

Атрибут placeholder — это краткая подсказка, поясняющая пользователю, какие данные должны вводиться в конкретное поле. Если требуется сравнительно длинная подсказка, опишите тип поля ввода в атрибуте title либо расположите соответствующий текст рядом с элементом <input>, но не вместо <label> или placeholder. Чтобы гарантировать, что формы будут легкодоступны, снабжайте элементы форм надписями. Надпись — это элемент <label>, а не <title> и не <placeholder>. Это необходимое условие доступности форм.

Хотя атрибут placeholder релевантен только при работе с типами полей ввода text, search, url, telephone, email и password, необходимо учитывать, что пока не все браузеры правильно поддерживают типы полей ввода date и color. Поэтому целесообразно добавлять значение placeholder, чтобы пользователь знал, в каком формате вводить данные. Особенно это касается случаев с использованием атрибута pattern, описанного далее. Значения placeholder содержатся в большинстве примеров кода к этой главе.

Псевдокласс пользовательского интерфейса :placeholder-shown был добавлен в спецификации CSS-селекторов уровня 4. Если этот псевдокласс поддерживается, то он обеспечивает оформление элемента <input> в зависимости от наличия или отсутствия текста-заполнителя (см. приложение):

```
input:placeholder-shown {}
input:not(:placeholder-shown) {}
```

ПРИМЕЧАНИЕ

Включайте в код атрибуты, описанные в этой главе, даже если они не полностью поддерживаются во всех браузерах. Если браузер «не поймет» какие-то атрибуты, он их просто проигнорирует. Такие атрибуты все равно будут полезны, их можно будет использовать вместе с JavaScript для имитации нужных функций в любых браузерах.

Можно применять JavaScript для перехватывания содержимого неподдерживаемых атрибутов, в частности placeholder, min, max, pattern, а также для приема неподдерживаемых типов полей ввода. Такие решения на JavaScript называются полизаполнениями.

pattern

Атрибут pattern поддерживается лишь при условии, что поддерживается и атрибут placeholder, — и это разумно. Атрибут pattern содержит регулярное выражение в стиле JavaScript, которому должно соответствовать значение элемента <input>. Форма будет отправлена на сервер, лишь если такое совпадение соблюдается.

В атрибуте pattern можно указать регулярное выражение, с которым будет сравниваться значение элемента управления. В настоящее время атрибут pattern чувствителен к регистру и требует полного совпадения. Язык регулярных выражений, используемый с этим атрибутом, почти не отличается от языка регулярных выражений в составе JavaScript, за тем исключением, что атрибут pattern должен

совпадать со всем значением, а не с его частью. Если вы хотите позволить пользователю добавить больше символов, чем предполагает ваше регулярное выражение, поставьте в конце выражения символ *.

В табл. 4.1 приведена базовая информация о регулярных выражениях.

Таблица 4.1. Некоторые метасимволы регулярных выражений, используемые при сравнении значения атрибута for с шаблоном

Метасимвол	Значение
?	Совпадение с предыдущим символом 0 или 1 раз
*	Совпадение с предыдущим символом 0 или более раз
+	Совпадение с предыдущим символом 1 или более раз
{n}	Совпадение с предыдущим символом ровно n раз
{n,m}	Совпадение с предыдущим символом не менее *n*, но не более *m* раз
[]	Совпадение с любым символом, содержащимся в квадратных скобках, происходит один и только один раз. Так, [123] означает совпадение с 1, 2 или 3
[n-m]	Дефис в квадратных скобках — это разделитель диапазона, позволяющий задавать диапазон. Выражение [123] можно записать как [1–3]
[^n-m]	Знак ^ в квадратных скобках выражает отрицание. Совпадение будет происходить с любым символом, не относящимся к диапазону *n–m*
\d	Совпадает с любой цифрой. Эквивалентно [0–9]
\D	Совпадает с любым символом, не являющимся цифрой. Эквивалентно [^0–9]
\s	Совпадает с любым пустым символом (пробелом или знаком табуляции)
\S	Совпадает с любым непустым символом (не являющимся пробелом или знаком табуляции)
\w	Совпадает с любыми буквой или числом. Эквивалентно [0–9A–Za–z]
\W	Совпадает с любым символом, не являющимся буквой или числом. Эквивалентно [^0–9A–Za–z]
()	Скобки могут использоваться для группирования (объединения) частей выражения
|	Вертикальная черта означает нахождение значений, расположенных слева или справа от нее. gr(a|e)y соответствует gray или grey

Подробное обсуждение регулярных выражений выходит за рамки этой книги. Просто учитывайте, что если вы допустите ошибку и `pattern` не окажется допустимым регулярным выражением, то при валидации эта последовательность символов будет игнорироваться, как если бы ее и не было.

При использовании атрибута `pattern` рекомендуется сопровождать его атрибутом `title`, в котором приводится описание шаблона. Продолжим рассматривать пример с цветом и номером кредитной карты:

```
<label for="col"> Color: </label>
<input pattern="#[0-9A-Fa-f]{6}"
  name="col" type="color" placeholder="#ffffff "
  id="col" title="A hash sign followed by 6 hexadecimal digits"/>
<label for="cc"> Credit Card: </label>
<input type="text" pattern="[0-9]{13,16}"
  name="cc" id="cc" title="13 to 16 digit credit card number"
  placeholder="credit card #"/>
```

Некоторые браузеры поддерживают тип поля ввода `color`, предоставляя цветовой виджет для выбора оттенка. Об этом мы поговорим далее в данной главе. Другие браузеры поддерживают атрибут `pattern`, но пока не поддерживают тип поля ввода `color`. Пока мы дожидаемся реализации полной поддержки, можно пользоваться атрибутом `pattern`, как показано в предыдущем коде. Он позволяет потребовать правильного формата ввода в тех браузерах, где поддержка пока остается половинчатой.

Некоторые браузеры полностью поддерживают операции с регулярными выражениями. Если в таком браузере пользовательский ввод не соответствует шаблону, указанному в атрибуте `pattern`, то форма не отправится, а браузер поместит в фокус первое поле для ввода, содержащее ошибку. В результате вы получите сообщение об ошибке валидации (рис. 4.1). Скриншот сделан в браузере, уже поддерживающем нативную валидацию.

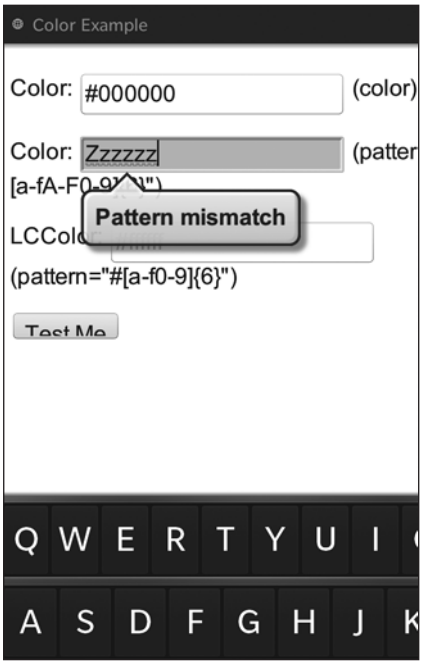


Рис. 4.1. Нативная валидация. При несовпадении с шаблоном выводится сообщение об ошибке (BlackBerry 10)

ПРИМЕЧАНИЕ

Совет по работе с CSS. Пользуйтесь псевдоклассом :invalid для нацеливания при оформлении на такие элементы, которые не соответствуют атрибуту pattern или по какой-то другой причине не являются валидными. Псевдокласс :valid обеспечивает совпадение, когда контент соответствует паттерну или вообще является валидной записью.

readonly

При наличии атрибута `readonly` соответствующий контент доступен только для чтения. Этот атрибут применяется с текстом, паролями, адресами электронной

почты, URL, а также при вводе времени/даты и числовых значений. Кроме того, этот атрибут может сопровождать элемент <textarea>. Он не используется с переключателями, флажками, элементами для закачки файлов, переключателями диапазонов, элементами для выбора, а также всеми видами кнопок, поскольку они по определению не предназначены для редактирования. Этот атрибут неновый, поэтому он поддерживается во всех браузерах, в том числе в старых версиях IE. Атрибут readonly является логическим, поэтому он может быть записан одним из двух следующих способов:

```
<input type="text" value="Not Editable" readonly/>
<input type="text" value="Not Editable" readonly="readonly"/>
```

disabled

Атрибут disabled отключает элемент формы. Он может применяться к любому элементу формы, кроме <output>. В HTML 4 атрибут disabled не применялся с элементом <fieldset>. Если этот атрибут применяется с элементом <fieldset>, он переопределяет атрибуты disabled у всех дочерних элементов управления в формах, даже если эти элементы и <fieldset> находятся в разных формах (см. подраздел «form» далее). Иными словами, элемент формы отключается, если у него установлен атрибут disabled, а также если атрибут disabled установлен у его родительского элемента <fieldset>.

ПРИМЕЧАНИЕ

Совет по работе с CSS: пользуйтесь псевдоклассом :disabled, чтобы нацеливаться на отключенные элементы и оформлять их.

Итак, в чем же разница между readonly и disabled? Оба этих атрибута не позволяют вносить изменения, но на элемент с атрибутом readonly можно перейти с помощью табуляции и отправить его вместе с формой. Элемент формы с атрибутом disabled не может получить фокус, а также не отправляется на сервер вместе с формой.

maxlength

Атрибут maxlength применяется с типами полей ввода text, password, url, search, telephone и email, а также с элементами <textarea>. Однако он не применяется с типами полей ввода «дата/время» или number. В HTML 4 этот атрибут применялся только с типами полей ввода text и password.

Допускается использовать maxlength при работе с адресами электронной почты и URL, но я не рекомендую этого делать без очень веской причины. Хотя этот атрибут и поддерживается во всех браузерах, напрашивается вопрос: почему кто-то разрешает пользовательскому интерфейсу трактовать некоторые адреса электронной почты или URL как «слишком длинные»? Да, возможно, это делается по причинам, связанным с обеспечением безопасности и в случаях, когда на самом деле имеются жесткие ограничения на количество символов. Однако если вас в первую

очередь волнует пользовательское восприятие, лучше откажитесь от применения этого атрибута. Даже в Twitter он не применяется, так как иногда пользователи все-таки записывают в сообщении более 140 символов, а уже потом удаляют некоторые символы или даже целые слова, чтобы уложиться в объем твита.

size

Атрибут `size` также используется довольно давно. Исторически у него было две функции: определение количества опций, которые должны отображаться в таком элементе управления, как `<select>`, а также определение количества символов в элементе формы — так контролировалась ширина этого элемента. Атрибут `size` элемента `<input>` следует считать устаревшим и реализовывать соответствующие функции на уровне CSS на этапе создания макета формы.

В течение некоторого времени атрибут `size` действительно считался нежелательным, но вновь появился в черновых вариантах спецификаций HTML5. Атрибут `size` не указывает, сколько символов может быть введено в форму (для этого существует атрибут `maxlength`) или сколько опций может быть выбрано (для этого пользуйтесь атрибутом `multiple`).

form

HTML-формы, появившиеся только в HTML5, не обязательно должны быть вложены в форму. Новый атрибут `form` позволяет ассоциировать HTML-форму с любой формой на странице. Кроме того, такой элемент может быть вложен в одну форму, но отсылаться с другой.

HTML-форма может иметь атрибут `form` с `id`, равным `<form>`, где `<form>` — тот элемент-форма, с которым ассоциирована данная HTML-форма. Таким образом, HTML-формы могут находиться на странице где угодно, в частности вне той формы, которая должна быть отправлена.

Все это несколько сложно объяснить, поэтому приведу пример:

```
<form id="form1">
  <-- здесь находится все вложенное содержимое формы -->
</form>

<p>
  <label for="userid">User ID</label>
  <input type="text" id="userid" name="user" form="form1"/>
</p>
```

`#userid <input>` не является потомком `#form1`. В предыдущих версиях HTML имя `name` и значение `value`, относящиеся к `#userid`, *не отсылались* вместе с формой при ее отправке на сервер. В браузерах, поддерживающих атрибут `form` из HTML5, `id` формы включается в качестве значения атрибута `#userid form`. Поэтому при отправке `#form1` значение `#userid` отсылается вместе с формой, хотя и не является ее потомком.

До появления таких веб-форм, как в HTML5, соответствующие элементы управления-формы приходилось вкладывать в родительский элемент <form>. При работе с HTML5 элементы формы и элементы <fieldset> ассоциированы с формами, указанными в их атрибуте form. Если такой атрибут отсутствует, то они ассоциируются с ближайшим родительским элементом <form>.

ВНИМАНИЕ

Обратите внимание: пустая строка form="" отменяет связь элемента с какими-либо формами, даже с той, которая является его предком. Это может привести к нежелательным последствиям. Как правило, следует использовать removeAttribute('form'), а не setAttribute('form', '');, чтобы избежать такого разрыва между вложенным элементом и формой.

autocompletion

Возможность автозавершения нативно предоставляется во многих браузерах[1]. Когда в браузере реализуется функция автозавершения, он может сохранять введенное пользователем значение. Таким образом, когда пользователь вернется на конкретную страницу, браузер может автоматически предварительно внести в формы некоторые значения. Атрибут autocomplete позволяет автору сайта (вам) подсказывать пользовательскому агенту, что вы хотите (или, наоборот, не хотите) применять автозавершение в конкретном поле формы. Атрибут autocomplete может принимать одно из трех значений: on, off или default. Ключевое слово on соответствует включенному состоянию, а ключевое слово off — выключенному.

Состояние off указывает, что информация, находящаяся в данном элементе формы, является конфиденциальной (например, это пароль) либо ее не планируется повторно использовать (допустим, это капча). Пользователю придется каждый раз заново вводить данные в такое поле, и браузер не должен предварительно заполнять его. Напротив, состояние on указывает, что пользователь может положиться на браузер, который запомнит информацию, ранее введенную в этот элемент управления. Пропуская значение, мы переводим HTML-форму в стандартное состояние (по умолчанию). В таком случае этот элемент управления должен иметь такое же значение автозавершения, как и та форма, с которой он ассоциирован:

```
<p>Login: </p>
<p>
  <label for="user">Username: </label>
  <input type="text" name="user" id="user" autocomplete="on"/>
</p>
<p>
  <label for="pwd"> Password:</label>
  <input type="password" name="pwd" id="pwd" autocomplete="off"/>
</p>
```

[1] Компания Google занимается разработкой API requestAutocomplete(), который предполагается сделать веб-стандартом. Этот стандарт нужен для того, чтобы можно было приказать браузеру внести в форму уже известную ему информацию.

autofocus

Атрибут `autofocus` указывает, что данный элемент должен находиться в фокусе, когда завершится загрузка страницы. На каждой странице в каждый момент времени только один элемент формы может обладать автофокусом. Атрибут `autofocus` является логическим, он может быть включен в один `<input>` (кроме скрытого), `<button>`, `<select>` или `<textarea>` на странице. Если атрибут `autofocus` присвоен более чем одному элементу, то в автофокусе окажется *последний* элемент с таким атрибутом.

Как было указано ранее, по причинам, связанным с удобством использования и доступностью, не рекомендуется пользоваться атрибутом `autofocus`. Если прибегнуть к jQuery, то функция этой библиотеки, в точности аналогичная рассматриваемому атрибуту, будет такой:

```
$('[autofocus]').last().focus();
```

Эта строка кода означает: «Найди все элементы с атрибутом `autofocus`, получи последний и помести его в фокус». Вероятно, вы хотели не этого. Чтобы работать было удобнее, лучше подсветить первый элемент, а не последний, то есть выполнить противоположную операцию.

Обратите внимание: фокусировка на текстовом поле при загрузке в iOS отключена, так как при этом экран накрывала бы виртуальная клавиатура.

В HTML5 добавилась масса полезных атрибутов и типов полей ввода. Теперь существует 23 типа полей ввода, а атрибутов ввода — даже больше. Как вы уже видели, некоторые атрибуты могут сопутствовать только определенным типам ввода. Браузеры, не поддерживающие при вводе конкретный атрибут `type`, вполне могут поддерживать другие атрибуты, связанные с `<input>`. Браузер может поддерживать атрибуты для типа поля ввода `text` (например, `maxlength` или `size`). В таком случае он поддерживает некоторые атрибуты, относящиеся к типу, не поддерживаемому в данном браузере, поскольку для ввода по умолчанию задается тип `text`. Так, в предыдущем примере мы убедились: пусть и не все браузеры поддерживают тип поля ввода `color`, они тем не менее поддерживают атрибут `disabled` для всех типов полей ввода.

Типы и атрибуты элемента <input>

В настоящее время существует 23 значения для типа поля ввода. Некоторые из них довольно старые, другие появились только в HTML5. Мы поговорим обо всех этих типах.

Повторное знакомство с уже известными типами полей ввода

Сначала давайте вспомним те типы полей ввода, которые могли встречаться нам до появления HTML5. Большинство разработчиков, считающих: «Я пишу на HTML уже много лет, все там знаю», порой даже не задумывались о том, какими разными

бывают различные типы <input>. Поэтому данный экскурс будет вам полезен, даже если вы считаете себя настоящим профессионалом.

ПРИМЕЧАНИЕ

Совет для профессионалов: как правило, требуется оформлять кнопки иначе, чем текст и другие типы полей ввода. Можно использовать селекторы атрибутов для нацеливания на поля ввода в формах в зависимости от их значения type. Следующий фрагмент кода создает серую обводку на границах всех элементов форм, кроме тех, которые обладают типом ввода submit:

```
input:not([type=submit])){
  border: 1px solid #666666;
}
```

Мы обсудим селекторы атрибутов и псевдокласс :not в главе 8.

Текст: <input type="text">

Текстовые поля, обладающие типом ввода данных text, то есть type="text", отображаются на экране как рамки для ввода информации, которую пользователь может записывать в одну строку. Значение text задается по умолчанию для обязательного атрибута type: если type пропущен или не поддерживается, то по умолчанию он будет иметь значение text.

ПРИМЕЧАНИЕ

Значение text задается по умолчанию для элемента <input>. Если атрибут type пропущен либо его значение записано с ошибками или не поддерживается, то браузер будет интерпретировать это поле ввода как имеющее тип type="text". Таким образом, если браузер не поддерживает новый тип ввода из HTML5, то по умолчанию будет приниматься тип ввода text. Поэтому смело пользуйтесь любыми типами полей ввода HTML5, даже если перед вами стоит задача поддерживать древний браузер Netscape 4.7. В худшем случае ваши пользователи везде увидят текстовые поля.

Атрибут value является опциональным. Если он присутствует, то значение атрибута value будет отображаться в текстовом поле сразу после загрузки страницы. Значение для атрибута value следует указывать лишь в случаях, когда вы предварительно заполняете форму (вносите в нее часть информации за пользователя) данными, которые хотели бы получить назад.

Другие атрибуты таковы: name, disabled, form, maxlength, readonly, size, autocomplete, autofocus, list, pattern, required и placeholder. Именно благодаря новым атрибутам, которые мы рассмотрели ранее, тип поля ввода text в HTML5 оказывается таким интересным и полезным:

```
<label for="username">Имя пользователя</label>
<input type="text" name="username" id="username"/>
```

ПРИМЕЧАНИЕ

Не рекомендуется указывать инструкции в качестве значений атрибута value, так как большинство пользователей отправят ваши же инструкции, а не будут сами заполнять эти поля в формах. Форма отправляется на сервер именно с теми значениями, которые находятся в ее полях в момент отправки. Поэтому если вы не предусматриваете предварительного заполнения формы, то не включайте в нее атрибут value. Если необходимо дать пользователю подсказку, то для этого используется атрибут placeholder — давнее решение этой еще более древней проблемы.

Если вы хотите включить в форму инструкции, отображаемые в текстовом поле по умолчанию, делайте это с помощью атрибута `placeholder`.

Ранее уже было указано, что *весьма целесообразно* предварительно заполнять текстовые поля значениями из базы данных, если пользователь уже зарегистрирован в системе, данная информация у вас есть и ее предзаполнение не создает проблем с безопасностью. Как правило, такое предзаполнение значительно улучшает пользовательское восприятие.

Пароль: <input type="password">

Тип поля ввода `password`, обозначаемый как `type="password"`, — это информация для заполнения полей с паролями. Такие поля почти не отличаются от обычных текстовых, за тем исключением, что значение, вводимое пользователем, либо значение, вносимое по умолчанию с помощью атрибута `value`, маскируется в пользовательском интерфейсе (обозначается звездочками). Введя текст `pAssW0rd`, пользователь увидит на экране ••••••••. Итак, хотя это значение и остается скрытым в пользовательском интерфейсе, значение пароля отсылается на сервер как обычный текст.

ПРИМЕЧАНИЕ

Обратите внимание: запрашивая у пользователя его пароль, пользуйтесь методом формы POST через SSL-соединение. Хотя пароль и маскируется в окне браузера, он уходит на сервер в виде обычного текста. Если бы вы воспользовались методом GET, то URL вашей страницы для отправки формы мог бы выглядеть так: https://www.website.com/index.php?user=Estelle&password=pAssW0rd.

Запрашивая пароль в одном из полей формы, пользуйтесь методом POST и передавайте информацию по протоколу HTTPS. Конечно, метод POST все равно отсылает пароль обычным текстом, но делает это неявно, что не является вопиющим нарушением соображений безопасности:

```
<label for="password">Пароль</label>
<input type="password" name="password" id="password"/>
```

В WebKit также существует возможность оформлять обычные текстовые поля как поля с паролями. Это делается с помощью свойства CSS `-webkit-text-security`. Установите свойство `-webkit-text-security` в значение `circle`, `square`, `disc` или `none`, чтобы изменить внешний вид тех значков, которые заменяют собой символы, вводимые пользователем.

По умолчанию на некоторых мобильных устройствах для повышения удобства работы с формой на крошечных клавиатурах применяется следующий подход: на экране отображается только последний символ, который только что был введен в поле с паролем.

Флажок: <input type="checkbox">

Тип поля ввода `checkbox`, обозначаемый как `type="checkbox"`, широко известен как флажок. По умолчанию такое поле выглядит как маленький квадратик. Если он

отмечен (установлен), то в квадратике ставится галочка. Если не отмечен — галочки нет, если состояние не определено (`indeterminate`), то через середину квадратика проходит горизонтальная черта. Флажки отлично подходят для ответов типа «да» или «нет» либо в случаях, когда пользователь может дать несколько ответов. Например, пользователю предлагаются правила конфиденциальности, действующие для данной веб-формы. Пользователь может согласиться с ними, установив флажок, либо не соглашаться и не устанавливать. На сайте турагентства вы можете выбрать вылет из Сан-Хосе или Сан-Франциско, а вылет из Окленда вас не устраивает. Поэтому вы устанавливаете флажки только для первых двух городов.

Обязательно указывайте атрибуты `name` и `value` для каждого из ваших флажков. Если в отправляемой форме установлен флажок, то значения его атрибутов `name` и `value` будут передаваться как пара «имя/значение». Неустановленные флажки просто исключаются из данных, отправляемых с формой:

```
<input type="checkbox" name="remember" value="true">
<label for="remember">Запомнить меня</label>
```

ПРИМЕЧАНИЕ

Совет для профессионалов. Можно специально оформлять флажок в зависимости от того, установлен он или нет. Это делается на уровне CSS с помощью селектора псевдокласса :checked. В следующем примере надпись, расположенная рядом с флажком, становится серой, если пользователь отмечает этот флажок:

```
input[type=checkbox]:checked + label {
  color: #cccccc;
}
```

Если этот код CSS вам пока непонятен, не волнуйтесь. В главе 8 мы обсудим селекторы атрибутов, селекторы смежных элементов и псевдокласс `:checked`.

Переключатель: <input type="radio">

Тип поля ввода `radio`, обозначаемый `type="radio"`, более известен под названием «переключатель». По умолчанию этот элемент представляет группу из маленьких кружков (положений переключателя). Такой кружок может быть заполнен, если он отмечен, либо пуст — в противном случае.

Как правило, переключатели используются при оформлении групп родственных значений, причем в такой группе может быть выбрано только одно значение. Например, широко известны вопросы с множественным выбором, на каждый из которых можно дать всего один ответ. Если вы предлагаете вопрос с множественным выбором, на который можно одновременно дать несколько ответов, пользуйтесь флажками. Если у вас есть только один вариант, пользуйтесь переключателем, а не флажком.

Положения одного переключателя должны иметь одинаковое значение атрибута `name` и разные значения атрибута `value`. Работая с переключателями, важно помнить о следующих вещах:

- в любом переключателе одновременно может быть выбрано только одно положение;
- при отправке формы на сервер отсылается значение атрибута `value` только выбранного положения переключателя вместе с его именем `name`. Поэтому обязательно сопровождайте каждое положение переключателя уникальным атрибутом `value`;
- пользователи могут устанавливать переключатель в разные положения, но не могут изменять их значения;
- снять выбор с одного положения переключателя можно лишь одним способом: выбрав другое положение. Иными словами, если в переключателе одно положение уже установлено, придется щелкнуть на другом положении, чтобы снять имеющееся выделение. Однако без помощи JavaScript вы никак не сможете сбросить все положения переключателя и вернуть форму в такое состояние, когда ни одно из положений не выбрано:

```
<p>Какой цвет ваш любимый (выберите один):</p>
<ul>
  <li>
    <label>red:
    <input type="radio" name="favoritecolor" value="red"/>
    </label>
  </li>
  <li>
    <label>green:
    <input type="radio" name="favoritecolor" value="green"/>
    </label>
  </li>
  <li>
    <label>blue:
    <input type="radio" name="favoritecolor" value="blue"/>
    </label>
  </li>
</ul>
```

Обратите внимание: все положения переключателя имеют одинаковые имена `name`. Имя `name` должно быть идентичным у всех положений одного переключателя. Если в этой группе присутствуют `ID`, то они должны быть уникальными.

ПРИМЕЧАНИЕ

Как вы могли заметить, в этом примере используются неявные метки. По соображениям доступности всегда следует сопровождать каждое поле ввода меткой — явной или неявной. Если метка явная, ее атрибут for должен совпадать с id элемента формы. Мы подробно рассмотрим элемент <label> далее в этой главе.

Кнопка отправки: <input type="submit">

Тип поля ввода `submit`, значение которого — `type="submit"`, обычно реализуется как кнопка отправки. Нажав эту кнопку, вы отправляете форму на сервер. Кноп-

ка отправки отсылает информацию формы, и воспрепятствовать этому можно двумя способами. Во-первых, для некоторых полей может быть задан атрибут `disabled`. Во-вторых, можно применить JavaScript, установив `return false` или `preventDefault()` на обработчике события отправки. Когда установлен атрибут `disabled`, «отключается» весь пользовательский интерфейс и пользователь не может нажать на нем ни одной кнопки. Метод JavaScript `preventDefault()` или `return false` не влияет на внешний вид интерфейса и не отменяет его «кликабельности».

```
<input type="submit" value="Отправить эту форму"/>
```

ПРИМЕЧАНИЕ

Обратите внимание: при использовании события onsubmit оно должно быть ассоциировано с элементом <form>, а не с кнопкой отправки формы. Ведь на сервер отправляется вся форма, а не только эта кнопка.

По умолчанию элемент интерфейса для отправки формы выглядит как кнопка, на которой записано значение ее атрибута `value`, выровненное по центру. Если также указать атрибут `name`, то пара «имя/значение» будет отправлена вместе с остальной формой.

В HTML5 кнопка отправки формы не обязательно должна содержаться в элементе `<form>`, содержимое которого мы собираемся отправлять. Можно ассоциировать HTML-форму с элементом `<form>`, который не является ее предком. Для этого достаточно применить атрибут `form`: `form="id_of_form_to_submit"`. В таком случае будет отправляться форма, указанная по значению атрибута `form`.

Итак, изучив этот элемент графического интерфейса, мы знаем достаточно, чтобы написать на HTML форму входа в игру CubeeDoo:

```
<form>
<ul>
<li>
  <label for="username">Имя пользователя</label>
  <input type="text" name="username" id="username"/>
</li>
<li>
  <label for="password">Пароль</label>
  <input type="password" name="password" id="password"/>
</li>
<li>
  <input type="checkbox" name="remember" value="true"/>
  <label for="remember">Запомнить меня</label>
</li>
<li>
  <input type="submit" name="submit" value="Зарегистрироваться"/>
</li>
</ul>
</form>
```

Сброс: <input type="reset">

Тип поля ввода, называемый «сброс» и имеющий значение `type="reset"`, более известен как кнопка сброса. Эта кнопка возвращает всю информацию в форме к исходным значениям, задаваемым по умолчанию, если этому не препятствует атрибут `disabled` или код JavaScript. Если в коде присутствует событие `onreset`, оно должно быть ассоциировано с элементом `<form>`, а не с кнопкой сброса. Ведь возврат к стандартным значениям происходит во всей форме, а не только у этой кнопки.

Одна из самых неприятных ошибок, которые могут случиться при завершении работы с формой, когда пользователь собирается ее отправить, — это нажатие кнопки сброса вместо кнопки отправки. Поэтому по возможности старайтесь не использовать в формах кнопку сброса. Если эта кнопка необходима, она должна располагаться довольно далеко от кнопки отправки и от того места, где пользователь ожидает увидеть такую кнопку. Так можно застраховаться от непреднамеренного сброса введенной информации.

Когда-то давно кнопка сброса была популярна, но сейчас ее днем с огнем не сыщешь, поскольку слишком многим пользователям доводилось одним махом стереть уже заполненную форму с множеством кропотливо введенной информации.

```
<input type="reset" value="Очистить эту форму"/>
```

Единственная ситуация, в которой кнопка сброса бывает очень полезной: у вас в форме есть переключатель и вы хотите позволить пользователю сбросить все положения.

По умолчанию графический элемент `reset` выглядит как кнопка, на которой записано содержимое атрибута `value`. В отличие от кнопки отправки формы, в случае с кнопкой сброса пара «имя/значение» не отсылается на сервер при отправке формы.

Файл: <input type="file">

Тип поля ввода `file`, обозначаемый как `type="file"`, отличается от других типов полей ввода. Этот элемент предоставляет пользователю возможность закачать файл на сервер, прикрепить его или каким-то другим образом разместить в Интернете файл со своего локального компьютера или из локальной сети. Ввод такого типа не предоставлялся в iOS ниже 6.0 (в браузере Safari для iPhone/iPad). В старых версиях iOS ввод типа `file` отображается как отключенный. Кроме того, ввод типа `file` отключен в IE10 в Windows Phone 8, но работает в Windows Phone 8 RT для планшетов.

В большинстве браузеров в лучшем случае предоставляются ограниченные возможности оформления полей ввода и кнопок. Не допускается оформление или изменение текста на связанных с ними кнопках, которые могут называться Browse (Обзор) или Choose (Выбрать). Однако современные браузеры начинают открывать объектную модель документа, ранее остававшуюся в тени. Кстати, в некоторых браузерах можно украшать с помощью CSS любые элементы форм, в том числе ввод типа `file`:

```
input[type="file"] {
  /* Здесь оформляется текст "выберите файл" */
}
input[type="file"]::-webkit-file-upload-button {
  /* здесь оформляется кнопка для выбора файла */
}
```

Элементы с типом ввода `file` могут иметь следующие атрибуты: `name`, `disabled`, `accept`, `autofocus`, `multiple`, `required` и `capture`. Если присутствует атрибут `value`, то он игнорируется.

Атрибут `accept` может применяться для создания списков с типами содержимого, разделенных запятыми. В этих списках указываются типы содержимого, которые сервер может правильно обработать. У типа ввода `file` не предусмотрены атрибуты `min` или `max`, соответственно, нельзя задать количество файлов, которое может загрузить пользователь. Но логически этот тип ввода принимает по умолчанию значения 0 или 1 для `min` и для `max` соответственно. Эти значения можно перезаписывать, задействуя атрибут `multiple`.

В некоторых мобильных браузерах атрибут `accept` обеспечивает доступ к камере, микрофону, а на отдельных устройствах — и к камкордеру:

```
<input type="file" name="image" accept="image/*;capture=camera">
<input type="file" name="video" accept="video/*;capture=camcorder">
<input type="file" name="audio" accept="audio/*;capture=microphone">
```

Эти значения не пользуются универсальной поддержкой, но работают в некоторых браузерах. В частности, это браузер Android 3.0, Chrome для Android (0.16), Firefox Mobile 10.0 и Opera Mobile 14.

Недавно спецификации были изменены[1]. Ранее компонент `capture` входил в состав атрибута `accept`, а теперь является самостоятельным логическим атрибутом. Приведенный ранее код в настоящее время записывается так:

```
<input type="file" name="image" accept="image/*" capture>
<input type="file" name="video" accept="video/*" capture>
<input type="file" name="audio" accept="audio/*" capture>
```

На устройстве можно выбрать, какой из поддерживаемых механизмов сохранения медиа будет использоваться. Если подходящий механизм для управления сохранением информации отсутствует, допустим и такой вариант функционирования, как если бы атрибут `capture` в принципе отсутствовал.

Скрытые элементы: <input type="hidden">

Тип поля ввода `hidden`, имеющий значение `type="hidden"`, требует всего три атрибута: `type="hidden"`, `name="somename"` и `value="some value"`. Элементы типа `hidden` не отображаются для пользователя, но применяются для передачи определенной

[1] В настоящее время активно разрабатывается черновик спецификации по сохранению медиа.

информации на сервер. Часто элементы типа hidden применяются для сохранения сеансовых ID, пользовательских IP-адресов или данных, собранных с предыдущих страниц многостраничной формы.

Многие разработчики также пользуются типом hidden для поддержки текущего состояния или других вспомогательных операций, помогающих в обработке хитросплетений клиентского JavaScript. В HTML5 в нашем распоряжении есть и альтернативные варианты, в частности элемент <output> и хранилище LocalStorage, а также старые добрые cookie. В результате такие нежелательные манипуляции с типом поля ввода hidden выходят из употребления. Я говорю здесь «нежелательные», так как традиционно тип поля ввода hidden используется только для скрытия некоторых пар «имя/значение», которые вы тем не менее хотите отправить на сервер.

Изображение: <input type="image">

Тип поля ввода image, обозначаемый type="image", в поведенческом отношении похож на тип поля ввода submit и принимает все атрибуты <img>, а именно src и alt. Если также включаются атрибуты value и name, то пара «имя/значение» кнопки с изображением отправляется вместе с формой. Если вы собираетесь сделать фотоснимок или закачать картинку на сервер, посмотрите ранее раздел о типе ввода "file".

Кнопка: <input type="button">

Тип поля ввода button, обозначаемый type="button" и широко известный как кнопка, практически не функционирует без обработчиков событий. Поэтому, как правило, его следует добавлять к форме с JavaScript в рамках постепенного улучшения. По умолчанию поле этого типа выглядит как кнопка, на которой записано содержимое атрибута value:

```
<input type="button" value="Я ничего не делаю"/>
```

ПРИМЕЧАНИЕ

Многие путают поле ввода типа button с элементом <button>, который гораздо проще оформлять, нежели компонент type="button". Более того, элемент <button> может без дополнительного кода выполнять некоторые операции, в частности отправлять форму или сбрасывать ее содержимое — JavaScript для этого не требуется. Пользуйтесь input type="button", если требуется сделать элемент формы внешне похожим на кнопку отправки. В других случаях лучше пользоваться элементом <button>, так как он более удобен при работе с таблицами стилей.

Оформление полей ввода разных типов

В каждом браузере по умолчанию применяется то или иное оформление для различных элементов форм. В браузерах WebKit и Mozilla мы можем изменять стандартное оформление с помощью еще не стандартизированного свойства appearance. В браузерах этих производителей действуют определяемые изготовителем префиксы -webkit-appearance и -moz-appearance. С их помощью можно изменять внешний

вид кнопок и других элементов управления так, чтобы они напоминали нативные. Кроме того, здесь мы получаем более широкие возможности переопределения стандартного внешнего вида элементов управления в формах.

Для свойства appearance поддерживается очень много значений, и здесь мы не можем рассмотреть их все. Только представьте себе, сколько стандартных элементов пользовательского интерфейса может содержаться в форме, от флажков и кнопок до диапазонов. Таблица стилей, применяемая пользовательским агентом по умолчанию, будет содержать такие значения appearance, как checkbox (флажок), button (кнопка) и slider-horizontal. В последнем случае речь идет о вложении теневого элемента DOM со значением slider-horizontal. Можно управлять значениями appearance с помощью CSS.

Новые значения типов <input>

А сейчас начинается самое интересное!

Раньше поле с типом ввода text использовалось для указания всевозможной информации: дат, адресов электронной почты, телефонных номеров и URL. Затем эту информацию приходилось валидировать на клиенте перед отправкой на сервер. Теперь все изменилось. Точнее сказать, все изменится, когда HTML5 будет полностью доработан, а вы реализуете все возможности, изученные в этой главе. Элемент <input> стал гораздо более разнообразным. В HTML5 определяются 13 новых значений для атрибута type HTML-элемента <input>:

- search;
- tel;
- url;
- email;
- datetime;
- date;
- month;
- week;
- time;
- datetime-local;
- number;
- range;
- color.

Поддержка форм HTML5 в браузерах для мобильных устройств и для ПК значительно улучшилась. Эти новые типы полей ввода, подобно radio, checkbox и button, имеют характерное графическое представление, которое зачастую отражает суть атрибута type. Кроме того, если речь идет о браузере сенсорного устройства с динамической виртуальной (а не физической) клавиатурой, то выводимая на экран клавиатура будет соответствовать типу ввода.

Хороший пример смартфона с динамической клавиатурой — BlackBerry 10. Если тип ввода равен `tel` и пользователь помещает поле ввода в фокус, отображается не клавиатура QWERTY, а телефонная клавиатура, как будет показано на рис. 4.4.

ПРИМЕЧАНИЕ

В большинстве браузеров поддерживается CSS-селектор псевдокласса пользовательского интерфейса :invalid. Вместо использования JavaScript при клиентской валидации и создании сообщений об ошибках (или вместе с JavaScript) можно указать недопустимые значения ввода с помощью CSS:

```
input:focus:invalid {background-color: #CCCCCC;}
```

Электронная почта: <input type="email">

Поле с типом ввода `email` напоминает текстовое и используется для указания адреса электронной почты.

При попадании такого элемента в фокус на сенсорном экране открывается виртуальная клавиатура, оптимизированная для ввода адреса электронной почты. Поля с типом ввода `email` поддерживаются на iPhone, начиная с версии 3.1. Клавиатура для электронной почты содержит буквы A–Z, символ @, точку и кнопку, на которой написано _123. Эта кнопка открывает обычную числовую клавиатуру, как на рис. 4.2:

```
<p>
  <label for="email">Электронная почта: </label>
  <input id="email" type="email" name="email"
  placeholder="name@domain.com" required multiple/>
</p>
```

Поля ввода с типом `email` поддерживают логический атрибут `multiple`, позволяющий вводить несколько адресов электронной почты, разделенных запятыми.

ПРИМЕЧАНИЕ

Чтобы указать несколько адресов электронной почты, разделяйте их запятой либо запятой и пробелом.

Согласно актуальному черновому варианту спецификации поля ввода с типом `email` должны поддерживать атрибуты `name`, `disabled`, `form`, `autocomplete`, `autofocus`, `list`, `maxlength`, `pattern`, `readonly`, `required`, `size` и `placeholder`.

URL: <input type="url">

Подобно полям с типом ввода `email`, поля с типом ввода `url` напоминают обычные текстовые и предназначаются для указания веб-адреса. Когда такой элемент попадает в фокус на сенсорном экране, на многих устройствах открывается виртуальная клавиатура, оптимизированная для ввода веб-адресов. На устройствах с iOS для полей с типом ввода `url` открывается клавиатура с буквами A–Z, точкой, прямым слешем, а также кнопкой, на которой написано .com. Однако, как показано на рис. 4.3, здесь отсутствует двоеточие. На BlackBerry открывается подобная клавиатура, но на ней нет ни двоеточия, ни прямого слеша.

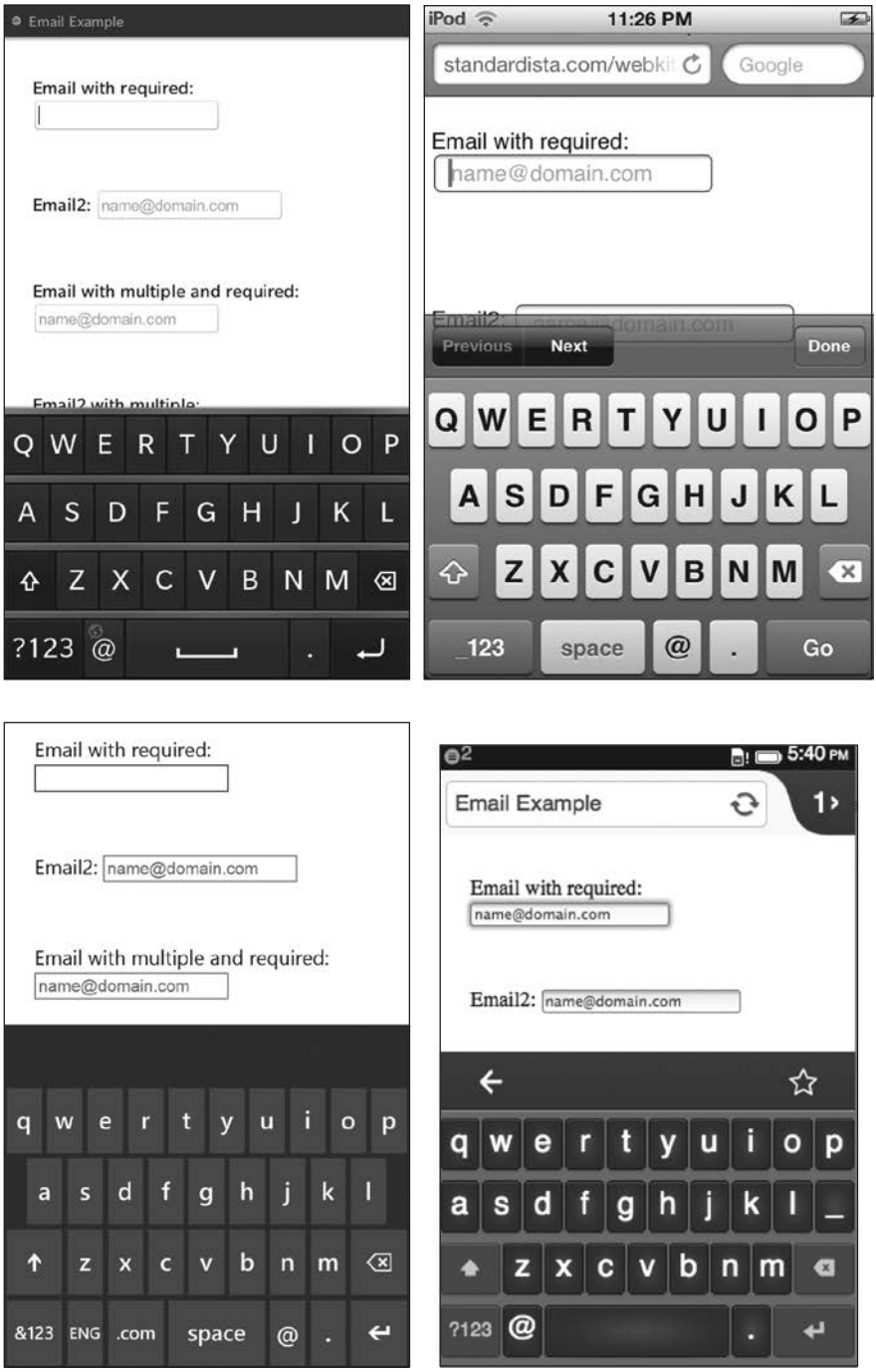


Рис. 4.2. Поле для ввода электронной почты и динамическая клавиатура. Примеры с BlackBerry 10, iPod, Windows Phone и Firefox OS

Рис. 4.3. Динамические клавиатуры для ввода URL на Firefox OS, iPod, BlackBerry 10, Windows Phone и Chrome на планшете Android

Браузеры, поддерживающие поля с типом ввода `url`, расценивают ввод как валидный, если URL начинается с указания протокола Интернета — любого подобного протокола, даже вымышленного. Так, Q:// будет ничуть не хуже, чем ftp://.

В настоящее время браузеры не проверяют сам URL (хотя это и требуется согласно спецификации HTML5), поскольку отсутствует код, проверяющий соответствие URL/URI действующим спецификациям. Консорциум W3C получил сообщение о такой ошибке. Тем временем была разработана чуть более удобная клавиатура (см. рис. 4.3), хотя по умолчанию она все-таки должна содержать еще и двоеточие.

Чтобы разрешить использование лишь конкретных протоколов, можно применять атрибут `pattern`:

```
<p>
  <label for="url">Веб-адрес: </label>
  <input id="url" type="url"
    pattern="^(http|https|ftp)\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]*"
    placeholder="http://www.domain.com" required />
</p>
```

Телефонный номер: <input type="tel">

`tel` означает номер телефона. В отличие от типов `url` и `email`, тип `tel` не требует конкретного синтаксиса или соответствия шаблону. Допустимы числа, буквы и, если уж на то пошло, практически любые символы, кроме перехода на новую строку. В разных странах телефонные номера могут состоять из разных наборов символов. Кроме того, могут различаться и способы записи телефонного номера. Например, в США номер +1(415) 555-1212 будет так же понятен, как и 415.555.1212.

Итак, зачем нужны поля с типом ввода tel? По умолчанию для таких полей выводится телефонная виртуальная клавиатура (см. рис. 4.4). Для ввода каждого типа данных нужно подбирать такую клавиатуру, которая наилучшим образом для этого подходит, — пользователи будут вам за это благодарны!

Можно подсказать желаемый формат записи телефонного номера, воспользовавшись атрибутом placeholder с таким синтаксисом и дополнительно разместив под полем формы образец такого синтаксиса в качестве комментария. Можно затребовать нужный формат, воспользовавшись атрибутом pattern и методом setCustomValidity() (об этом мы поговорим в разделе «Валидация форм» далее). Так обеспечивается создание специальных сообщений об ошибках в процессе клиентской валидации:

```
<p>
  <label for="tel">Телефон: </label>
  <input id="tel" type="tel" placeholder="XXX-XXX-XXXX"
    pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" required />
</p>
```

Номер: <input type="number">

Поля для ввода типа number напоминают текстовые, но предназначены для указания номера. Когда при работе с сенсорным экраном элемент типа number оказывается в фокусе, на экране появляется клавиатура, показанная на рис. 4.5. При этом могут использоваться атрибуты min, max и step.

Атрибут min соответствует минимальному допустимому значению, max — максимальному. Атрибут step указывает величину шага между доступными значениями. По умолчанию значение step равно 1, но допускает и использование чисел с плавающей точкой. Атрибуты min, max или step могут принимать значения в виде чисел с плавающей точкой.

В пользовательском интерфейсе браузеров для ПК, полностью поддерживающих эту функцию, можно уменьшать или увеличивать значение счетчика без ввода с клавиатуры. В мобильных браузерах такой блок с изменяемым значением мне пока не встречался. Даже если поле для ввода номеров снабжено графическим счетчиком со стрелками, указывающими вверх и вниз, оно все равно принимает и ввод данных с клавиатуры. При использовании блока с изменяемым значением (в настоящее время он поддерживается лишь в некоторых браузерах для ПК) можно щелкнуть кнопкой мыши на одной из стрелок или коснуться стрелки пальцем — и значение счетчика изменится на величину step. При этом будут отображаться только допустимые значения. Если элемент формы является обязательным полем, то форма не отправится, если в этом поле будет находиться недопустимое значение. К недопустимым значениям относятся нечисловые, числовые, превышающие max или не превышающие min, либо недопустимые количества шагов выше min. Действительно, при попытке отправить форму с таким недопустимым значением оно попадает в фокус:

```
<input type="number" min="0" step="5">
```

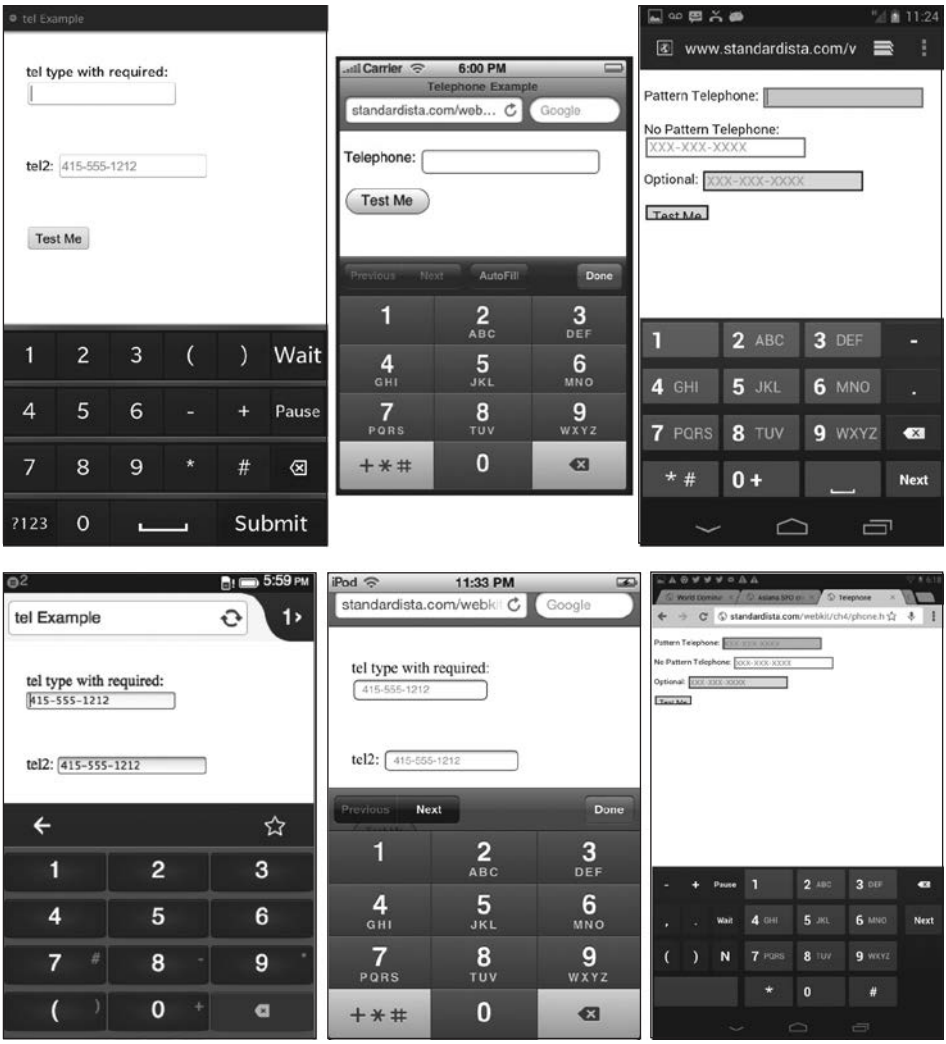


Рис. 4.4. Поле для ввода телефонных номеров и соответствующие клавиатуры

Чтобы значение было валидным, оно должно представлять собой число, равное `min + n * step`, где `n` — любое целочисленное значение, относящееся к диапазону `min`/`max`. Например, если `min=2`, `max=10` и `step=5`, то значение `7` является допустимым, а `10` — нет:

```
<p>Любое число между 100 и 999</p>
<p>
  <label for="number">Число: </label>
  <input id="number" type="number" placeholder="100 to 999"
    pattern="[1-9][0-9]{2}" min="100" max="999" required />
</p>
```

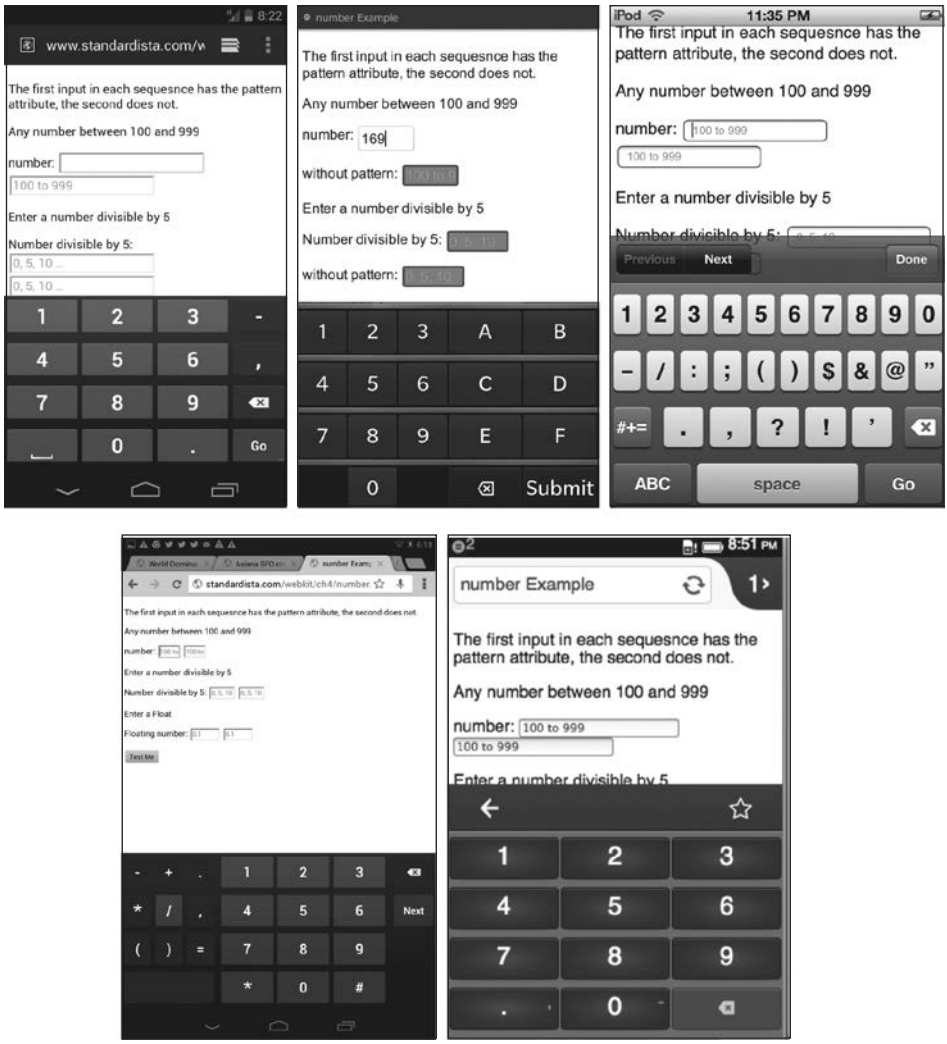


Рис. 4.5. Ввод номеров в специальное числовое поле формы

```
<p>Введите число между 0 и 1000, кратное 5</p>
<p>
  <label for="even">Число, кратное 5: </label>
  <input id="even" type="number" placeholder="0, 5, 10 &hellip;"
    pattern="[0-9]*[05]" min="0" max="1000" step="5" required />
</p>
<p>Введите положительное дробное число меньше 10.0</p>
<p>
  <label for="float">Дробное число: </label>
  <input id="float" type="number" placeholder="0.1"
    pattern="[0-9](?\.[0-9])?" min="0.1" max="9.9" step="0.1"/>
</p>
```

При работе с новыми типами полей ввода, применяемыми в HTML5, используются новые API. С атрибутом step используются методы stepUp() и stepDown():

```
input.stepUp(x)
input.stepDown(x)
```

Два этих метода изменяют значение элемента формы на величину, указанную в атрибуте step, умноженную на x либо на 1, если параметр не передан. Значения должны относиться к диапазону от min до max.

Атрибут pattern не поддерживается с полями типа number, но я указываю его здесь, так как он поддерживается шире, чем тип поля ввода типа number. Атрибут pattern можно считать механизмом постепенного подстраивания функций (graceful degradation) для браузеров, поддерживающих атрибут pattern, но не вполне поддерживающих поля для ввода того или иного типа[1].

Если браузер поддерживает поля с типом ввода number, то эта возможность имеет приоритет над шаблоном. Шаблоны вида pattern="[0-9]*" или pattern="\d+|\d+\.\d+" практически эквивалентны типу number. Однако если в регулярном выражении необходимо обеспечить соответствие значениям min, max и step, то такое выражение может получиться очень неудобным.

Диапазон: <input type="range">

Поле с типом ввода range — это слайдер, похожий на тот, что показан на рис. 4.6. Пользователь может перетаскивать по нему ползунок, а на сенсорном экране также касаться конкретного значения и таким образом выбирать его. Как и в случае с number, минимальное значение слайдера задается в атрибуте min, максимальное — в атрибуте max, а шаг между двумя соседними значениями — в атрибуте step. Хотя поля с типом ввода появились еще в Safari 2.0, только в браузере Safari 5 полноценно поддерживаются атрибуты min, max и step. Таким образом, этот тип ввода наконец-то можно использовать в WebKit. Opera, BlackBerry, IE10 и Chrome также поддерживают range. Mobile Firefox поддерживает поля с типом ввода range, начиная с версии 23. В Android range поддерживается частично.

По умолчанию ползунок слайдера находится посередине между минимумом и максимумом. Это стандартное положение ползунка можно изменить, указав другую величину в атрибуте value. Если крайние значения диапазона равны 20 и 30, то по умолчанию ползунок будет находиться в точке 25. Если вы не зададите значения для атрибутов min, max или step, то они по умолчанию будут равны 0, 100 и 1 соответственно. Поскольку определенное значение задается по умолчанию, поля с типом ввода range всегда возвращают значения на всех устройствах, на которых поддерживаются.

Меня часто спрашивают: «А можно сделать слайдер вертикальным, а не горизонтальным?» Да, в некоторых браузерах. Вот как это делается в WebKit:

```
input[type=range]{-webkit-appearance: slider-vertical;}
```

[1] Например, на момент написания этой книги браузер Safari для iOS 6 поддерживал атрибут pattern, но не работал с полями ввода типа number и не выполнял валидации данных при отправке.

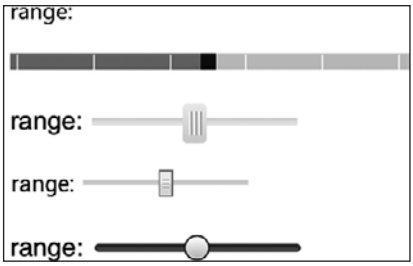


Рис. 4.6. Слайдеры для работы с диапазонами значений в Windows Phone, BlackBerry 10, iPhone и Chrome

Кроме того, можно объявить для высоты большее значение, чем для ширины. Таким образом вы создадите вертикальный слайдер в старых версиях Opera, основанных на Presto.

Поле для поиска: <input type="search">

Поля с типом ввода `search` предназначены для поисковых запросов. Хотя в спецификации не регламентируются конкретные элементы пользовательского интерфейса для новых типов форм, поисковое поле часто оформляется как прямоугольник со скругленными углами. Во многих, но не во всех браузерах, когда это поле непустое, справа от него находится кнопка `search-cancel-button`. При нажатии на нее весь текст в поисковом поле стирается, как показано на рис. 4.7.

На некоторых устройствах отображается клавиатура со словом search (поиск) или значок в виде лупы. Как правило, на такой клавиатуре есть кнопка Go (Вперед) или Enter (Ввод).

Цвет: <input type="color">

Если поле с типом ввода `color` полностью поддерживается, то оно выглядит так, как показано на рис. 4.8. Поле с типом ввода `color`, как и все новые поля такого рода, может поддерживаться не полностью, тогда оно выглядит как обычное текстовое. Цветовые значения записываются в нижнем регистре в обычном цветовом формате. По умолчанию палитра имеет значение `#000000`. Таким образом, если поле с типом ввода `color` поддерживается, то оно всегда возвращает значение.

В течение некоторого времени отдельные браузеры поддерживали именованные цвета, например `indianred` (ярко-красный). Именованные цвета не упоминаются в спецификации, и такая поддержка была прекращена.

Нативные интерфейсы с цветовыми палитрами очень красиво выглядят в браузерах, но пока они не слишком распространены. Чтобы «сымитировать» поддержку цветовых значений, программа должна принимать только такие значения, которые соответствуют регулярному выражению `#[a-zA-Z0-9]{6}`. Для этого используется атрибут `pattern` с заполнителем `placeholder`, указывающим, что требуются шестнадцатеричные цветовые значения. Согласно спецификациям, ни один из этих атрибутов

Рис. 4.7. Поисковое поле в BlackBerry 10 (обратите внимание на значок для удаления текста в поле) и iOS 6.1 (обратите внимание на клавишу Search (Поиск))

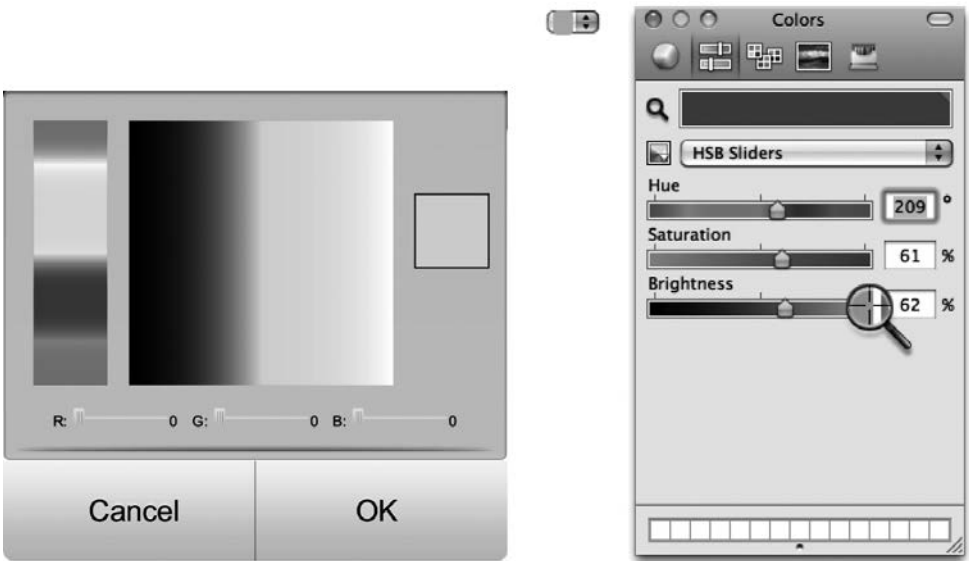


Рис. 4.8. Палитра в BlackBerry 10 и в браузере Opera на Mac

не поддерживается полями ввода с типом color, но они просто игнорируются в браузерах, полностью поддерживающих поля с таким типом ввода:

```
<label for="clr">Цвет: </label>
<input id="clr" name="clr" type="color" placeholder="#000000"
    pattern="#[0-9A-Fa-f]{6}" required />
```

Поля для ввода даты и времени

Существует ряд полей с новыми типами ввода, предназначенных для указания даты и времени, в частности date, datetime, datetime-local, month, time и week. Все показатели времени соответствуют стандарту ISO 8601. Браузеры, поддерживающие такие возможности, предоставляют интерактивные виджеты, напоминающие нативный браузерный календарь. Значение date обладает более качественной кросс-браузерной поддержкой, чем любой другой тип полей ввода, связанных с датой и временем.

Дата: <input type="date">

Поле с типом ввода date позволяет указывать информацию о дате: год, месяц и день. Как правило, в пользовательском интерфейсе такое поле выглядит как графический компонент с диапазоном дат. В браузере также может поддерживаться графический компонент с календарем.

В разных странах даты записываются по-разному. В некоторых браузерных элементах сначала указывается год, в других — день, но сама дата перед отправкой на сервер всегда преобразуется в один и тот же формат: ГГГГ-ММ-ДД:

```
<p>
  <label for="date">Дата: </label>
  <input id="date" name="date" type="date"
    placeholder="YYYY-MM-DD" required />
</p>
<p>
  <label for="dob">Дата рождения: </label>
  <input id="dob" name="dob" type="date"
    placeholder="YYYY-MM-DD" min="1900-01-01" required />
<p>
```

Пока поля с типом ввода date не поддерживаются во всех браузерах, можно использовать сопоставление с шаблоном (pattern). Тем не менее, полагаясь на сопоставление с шаблоном, вы не сможете создать для пользователей нативный календарный виджет. Также вам может понадобиться код для валидации на JavaScript, чтобы учитывать такие факторы, как високосные годы и т. п., поскольку регулярные выражения для работы с датами — это кромешный ад.

Как понятно из рис. 4.9, в браузерах, поддерживающих поля с типом ввода date, предоставляется элемент для выбора даты и не допускается непосредственный ввод таких значений. Так гарантируется, что пользователь в любом случае укажет допустимую дату (возможно, неверную, но допустимую).

Тип `date` предусматривает полную дату без указания времени или часового пояса. Такой элемент обладает наиболее широкой поддержкой по сравнению с различными типами полей ввода даты/времени. Мы вкратце рассмотрим и остальные виды таких полей и поговорим о том, как они поддерживаются.

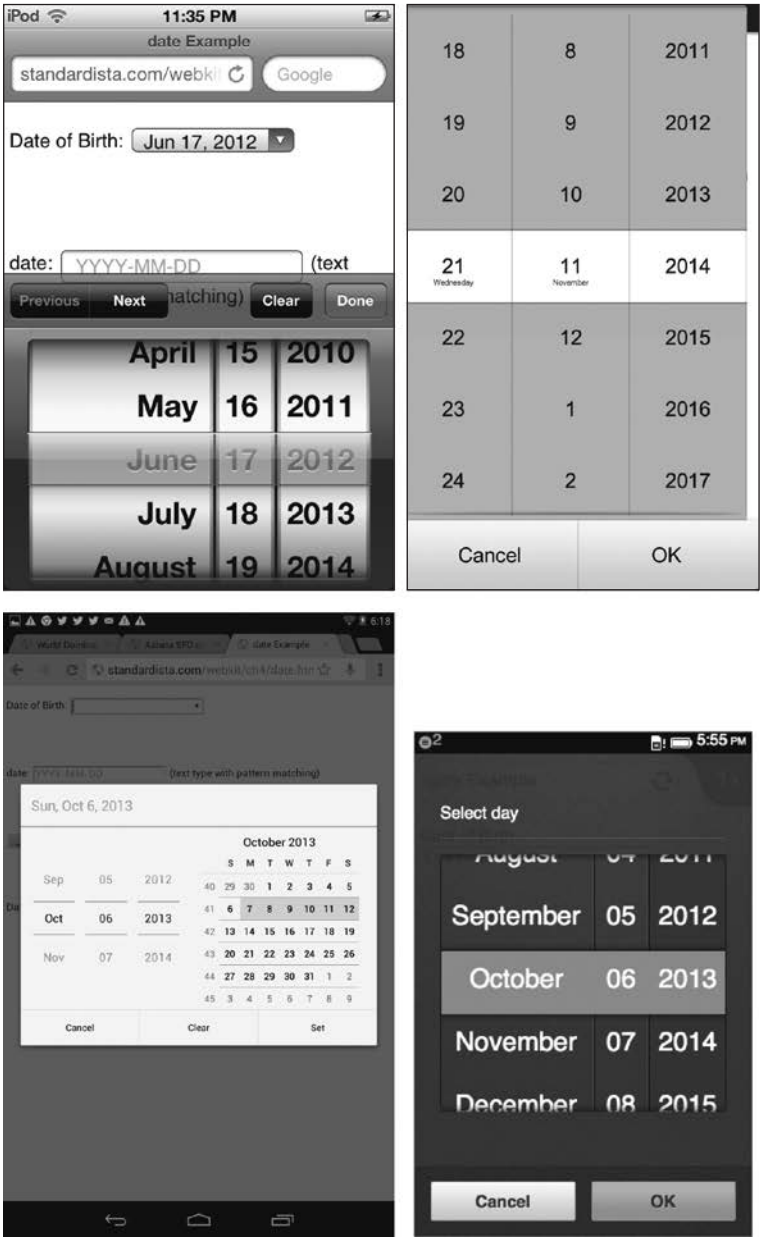


Рис. 4.9. Элемент для выбора даты, предоставляемый в iOS, BlackBerry, Android и Firefox OS, когда поле для ввода даты попадает в фокус

Дата и время: <input type="datetime">

Поле с типом ввода datetime на самом деле состоит из двух полей: в одном из них указывается дата (год, месяц, день), а в другом — время (час, минута, секунда, доля секунды). При этом часовой пояс указывается в формате всемирного координированного времени (UTC) с учетом минут и секунд, но не долей секунды. Можно задать атрибуты min, max и step для ограничения диапазона значений, например: min="2012-03-01T12:00Z".

Возможно, вам будет удобнее использовать сопоставление с шаблоном, но тем не менее все равно придется валидировать ввод на соответствие истинному времени и допустимым датам:

```
<p>
  <label for="datetime">Дата и время: </label>
  <input id="datetime" name="datetime" type="datetime" placeholder="YYYY-
   MM-DD"
    min="2010-01-01T00:00Z"max="2011-12-31T23:59Z" required />
</p>
<p>
  <label for="dte">Дата и время: </label>
  <input id="dte" name="dte" type="text" placeholder="YYYY-MM-DDT00:00Z"
    pattern="\d{4}\-\d{2}\-\d{2}T\d\d\:\d\dZ" required />
</p>
```

Локальные показатели даты и времени: <input type="datetime-local">

Значение datetime-local идентично datetime, но указывается не в системе UTC (Z не включается).

Месяц: <input type="month">

В полях с типом ввода month должна указываться информация о месяце и годе, но не должны указываться день месяца и часовой пояс. Значения, задаваемые по умолчанию на разных устройствах, различаются. Например, значение min по умолчанию может равняться 0001-01, а max — 214748 3647-12. Поэтому рекомендую указывать значения min и max. Можно также использовать атрибут step. Например, пользуйтесь step="6", чтобы ограничить выбор месяцев. В таком случае можно будет выбрать только январь или июль:

```
<p>
  <label for="month">Месяц: </label>
  <input id="month" name="month" type="month" placeholder="YYYY-MM" required
    min="2010-01" max="2020-01" step="6"/>
</p>
```

В JavaScript нумерация начинается с нуля, но в данном случае январю соответствует значение 01.

Время: <input type="time">

Поля с типом ввода time обеспечивают механизм для ввода времени в 24-часовом формате. Значение времени должно быть больше или равно 0 часов, но не должно превышать 24 часов. Для ввода более жестких ограничений применяются атрибуты min и max.

Значение времени увеличивается в секундах, а не в минутах. Задав шаг 60, равный 60 секунд, мы значительно улучшим пользовательское восприятие. В нашем примере атрибут step имеет значение 900 — это количество секунд, содержащееся в 15 минутах. Соответственно, значения времени увеличиваются с шагом 15 минут:

```
<p>
  <label for="time">Время встречи: </label>
  <input type="time" min="09:00" max="17:00" name="time" id="time"
    step="900" placeholder="12:00" required />
</p>
```

Здесь type означает не истекшее время, а время суток. Когда такой тип ввода поддерживается, браузер должен показывать виджет для работы со временем, например, напоминающий часы. Временной пояс при этом не указывается.

Неделя: <input type="week">

Поля с типом ввода week позволяют указывать номер недели в году, не включая год, месяц, день или время дня. Значение в рамках года варьируется от 01 до 52. Например, первая неделя 2014 года будет записываться так: 2014-W01.

Недельный календарь начинается не с 1 января, а с той недели, на один из дней которой приходится 4 января. Эта неделя может и не содержать 1 января:

```
<input type="week" name="week" id="week" min="2010-W01" max="2020-W02" required />
```

Поля с типом ввода week поддерживаются наименее широко из всех полей ввода даты и времени. Вот мы и добрались до такого поля...

Итак, мы можем оперировать всего 23 типами полей для ввода информации. Вот они:

- button;
- checkbox;
- color;
- date;
- datetime;
- datetime-local;
- email;
- file;
- hidden;
- image;

- month;
- number;
- password;
- radio;
- range;
- reset;
- search;
- submit;
- tel;
- text;
- time;
- url;
- week.

Валидация форм

В настоящее время веб-разработчики пользуются сценариями на языке JavaScript для выполнения валидации форм на клиентской стороне. Все ближе тот день, когда мы сможем просто написать приведенную далее простую форму, а браузер сам не допустит ее отправки, если она будет содержать недопустимые данные. Никакой клиентской валидации на JavaScript для этого не потребуется. Правда, в любом случае понадобится выполнять серверную валидацию. Пользователь сможет отправить такую форму, только если во всех полях ввода будут указаны удовлетворяющие им значения. Так, в последних двух полях должны находиться соответственно один адрес электронной почты и один URL:

```
<form>
  <ul>
    <li>
      <label>
        Имя: <input name="name" required"/>
      </label>
    </li>
    <li>
      <label>
        Электронный адрес: <input name="email" type="email" required />
      </label>
    </li>
    <li>
      <label>
        Веб-адрес: <input name="website" type="url" required />
      </label>
    </li>
```

```
    <li>
      <input type="submit" value="Отправить"/>
    </li>
  </ul>
</form>
```

При отправке такой формы браузер проверяет, все ли необходимые условия выполняются. Если все поля заполнены, а значения в `email` и `url` имеют верный формат, то форма будет отправлена успешно. Если в каком-либо обязательном элементе формы будет ошибка или пропуск, то пользователь увидит сообщение об ошибке, примерно такое, как на рис. 4.10.

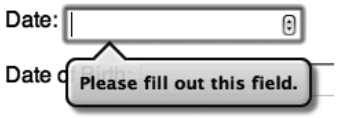


Рис. 4.10. Сообщение об ошибке, требующее заполнить обязательные поля формы, которые оказались пустыми в момент отправки формы

Если обязательные поля оставлены незаполненными либо введенное значение не соответствует атрибуту `type` или `pattern`, то отображаются сообщения об ошибках (рис. 4.11). В большинстве браузеров поддерживается нативная валидация форм, а в iOS и Android в ходе такой работы обеспечивается запаздывание ввода.

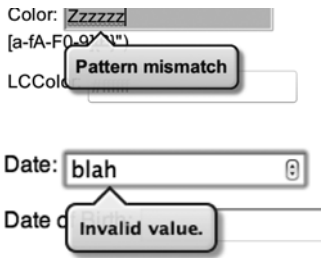


Рис. 4.11. Поля форм и сообщения, указывающие на то, что валидация не была пройдена

При нативной браузерной валидации данных в формах недопустимые данные обнаруживаются, после чего такие поля помещаются в фокус и сопровождаются сообщениями об ошибках. В браузерах, поддерживающих HTML5, нативная валидация осуществляется перед отправкой формы на сервер. Форма уходит на сервер лишь при условии, что все правила, заданные для полей в атрибутах, удовлетворяют условиям нативной валидации в браузере.

Не допуская отправки невалидных данных, браузер экономит один этап двухсторонней связи с сервером. Хотя в конечном итоге нативная браузерная валидация должна заменить собой клиентскую валидацию JavaScript, она никогда не сможет заменить серверную валидацию. Браузерная валидация является недостаточно тщательной, чтобы обработать все ошибки. Обязательно используйте серверную

валидацию, поскольку злоумышленники всегда смогут подделать HTTP-запросы или иным образом вмешаться в отправку данных.

Когда валидация форм выполняется нативно, первый же элемент формы, содержащий недопустимые данные, будет сопровождаться сообщением об ошибке и попадать в фокус, если пользователь попытается отправить не вполне валидную форму. В конечном итоге нативная клиентская валидация позволяет обеспечить правильное заполнение формы на стороне клиента без применения JavaScript. HTML5 значительно снижает требования к клиентской валидации форм, но определенную работу по-прежнему приходится выполнять.

В спецификациях HTML5 описаны свойства и методы DOM, позволяющие выполнять валидацию без преодоления разнообразных кросс-браузерных преград. В HTML5 появилось восемь новых свойств элементов форм, предоставляемых посредством объекта состояния `validity`[1].

Свойство `validity` входит в состав API ограничения допустимости. Оно доступно для всех элементов форм, поддерживающих валидацию. Для обращения к нему используется свойство `validity`, которым обладают относящиеся к формам элементы управления:

```
var element = document.querySelector('#form_control_id');
var validityStateObject = element.validity;
```

или

```
var validityStateObject = document.form_id.form_control_id.validity;
```

Объект `validityStateObject` содержит ссылки на несколько свойств `validity`:

- `element.validity.valueMissing`. Если у элемента, обязательного для заполнения (имеющего атрибут `required`), отсутствует значение, то свойство `valueMissing` равно `true`, в противном случае — `false`. Свойство `valueMissing` проверяет, есть ли в наличии нужное значение, но не проверяет его допустимость;
- `element.validity.typeMismatch`. Возвращает `true`, если значение элемента имеет неверный синтаксис, в противном случае — `false`. Например, если тип — `number`, `email` или `url`, а значение не является соответственно числом, адресом электронной почты или URL, то свойство `typeMismatch` возвращает `true`;
- `element.validity.patternMismatch`. Если содержащийся в форме элемент управления требует соответствия определенному шаблону, указанному в атрибуте `pattern`, а значение элемента формы не соответствует этому шаблону, то свойство `patternMismatch` равно `true`, в противном случае — `false`. Атрибут `pattern` ограничивает возможные значения набором конкретных форматов. Эти форматы определяются регулярным выражением, присваиваемым атрибуту `pattern` в качестве значения. В сущности, `patternMismatch` — это свойство, требующее выполнения любых правил `pattern`, заданных для элемента формы.

---

[1] Если быть точной, существует десять свойств допустимости, но спецификация развивается. В этой главе я расскажу о первых восьми свойствах, которые уже широко поддерживаются в браузерах.

Как было указано ранее, при использовании атрибута pattern необходимо задавать также атрибут title, описывающий правила соответствия конкретному формату. Так улучшается доступность ресурса;

- element.validity.tooLong. Когда для элемента формы задан атрибут maxlength, может использоваться свойство tooLong. Оно возвращает true, если значение элемента превышает допустимую максимальную длину, в противном случае — false. Таким образом гарантируется, что значение не будет содержать слишком большого количества символов. На самом деле сам атрибут maxlength должен предотвратить ввод излишне длинной последовательности символов. Свойство tooLong — это своеобразный механизм дополнительной проверки, удостоверяющий соответствие атрибуту maxlength;
- element.validity.rangeUnderflow. Свойство rangeUnderflow регламентирует минимальное значение для элемента формы, если задан атрибут min. Свойство rangUnderflow возвращает true, если значение элемента оказывается меньше заданного минимума;
- element.validity.rangeOverflow. Свойство rangeOverflow является парным для rangeUnderflow. Оно задает максимальное значение для элемента формы, если задан атрибут max;
- element.validity.stepMismatch. Возвращает true, если значение элемента не соответствует правилам, указанным в атрибуте step, в противном случае — false. Свойство stepMismatch гарантирует, что значения элементов форм одновременно удовлетворяют как значению step, так и значению min. Любое значение такого элемента должно быть кратным step плюс минимальное значение;
- element.validity.valid. Если элемент формы полностью соответствует правилам допустимости, то свойство valid объекта ValidityState возвращает true, в противном случае — false. Считайте это свойство квинтэссенцией предыдущих семи, а также свойства customError, описанного далее. Если все эти восемь свойств вернут false, то valid будет равно true. Если какие-то из восьми свойств не пройдут валидацию (окажутся равны true), то valid будет равно false;
- element.validity.customError. Кроме описанных ранее свойств допустимости существует свойство customError. Оно возвращает true, если в элементе возникает специально описанная (пользовательская) ошибка, обеспечивающая проверку ошибок, но не допустимости как таковой.

  По умолчанию для всех элементов управления задается свойство customError, получающее в качестве значения пустую строку. Все пустые строки дают значение «ложь». Чтобы значение этого свойства возвращало «истина», необходимо вызвать в элементе формы метод setCustomValidity(message), где message — это текст, содержащийся в выводимом на экран сообщении об ошибке (см. рис. 4.1, 4.10 и 4.11). При отправке сообщения элемент формы получает значение customError, равное true, так как специально заданное со-

общение об ошибке уже не является пустым. Пока это свойство не получит значение `false`, вы не сможете отправить форму.

Когда задано специальное сообщение, описывающее нарушение правил допустимости, элемент формы является невалидным и возвращает ограничение `customError` со значением `true`. Чтобы убрать ошибку, просто вызовите в этом элементе управления метод `setCustomValidity("")`, сообщив этому методу пустое строковое значение.

ПРИМЕЧАНИЕ

Если установить customError в значение true, задав пользовательское сообщение о допустимости, форма все равно не будет отправлена, даже если остальные ее поля заполнены правильно. Обязательно сбрасывайте такое значение и ставьте вместо него пустую строку, чтобы правильно заполненные формы отправлялись успешно.

В некоторых браузерах вы можете не только изменять содержание сообщений об ошибках, но и управлять внешним видом этих сообщений. Описанные ранее возможности оформления ошибок в формах и других деталей пользовательского интерфейса до некоторой степени поддаются обработке с помощью CSS.

Графическое оформление экранных сообщений об ошибках

WebKit позволяет оформлять экранные сообщения об ошибках таким образом, чтобы они выглядели и работали как нативные. Такое сообщение, заключенное в стилизованное облако, состоит из четырех компонентов, входящих в состав теневой модели DOM. Их оформление выполняется с помощью псевдоэлементов, применяемых к отдельным частям графического сообщения об ошибках:

```
::-webkit-validation-bubble
::-webkit-validation-bubble-arrow-clipper
::-webkit-validation-bubble-arrow
::-webkit-validation-bubble-message
```

Объемлющий элемент получает абсолютное расположение: `::-webkit-validation-bubble`. Его дочерний элемент `::-webkit-validation-bubble-arrow-clipper` обрезает стрелку `::-webkit-validation-bubble-arrow` так, чтобы она имела высоту 16 пикселов. Так создается «хвостик» графического облачка. Элемент `::-webkit-validation-bubble-message` содержит текстовый узел актуального сообщения об ошибке. Оформление, применяемое по умолчанию, задается в пользовательском агенте стилей, и это оформление можно переопределить с помощью собственных CSS[1].

[1] Такая возможность присутствует в WebKit, но отсутствует в Blink. К моменту выхода оригинала этой книги данная возможность уже удалена из Chrome. Надеюсь, что сохранится возможность оформления графических валидационных сообщений с помощью веб-компонентов.

Оформление для повышения удобства использования

Браузеры, поддерживающие HTML5, позволяют в определенной степени оформлять не только сообщения об ошибках, но и все элементы управления, находящиеся в формах. Вы как разработчик стремитесь максимально облегчить работу пользователю (так и должно быть) и знаете такие тонкости, как изменение внешнего вида указателя над отключенными и активными элементами форм. В таких случаях форма указателя указывает на ожидаемое действие пользователя. Большинство пользователей могут испытывать некоторые затруднения, поэтому целесообразно оптимизировать работу с формами с помощью таблиц стилей.

Например, нужно дать пользователю понять, что данный элемент формы является обязательным для заполнения. Можно поставить рядом с этим полем астериск. Более того, астериск можно заключить в класс, чтобы повысить его заметность. Для этого потребуется некоторая дополнительная разметка, а именно:

```
<span class="required">*</span>
```

Мы добавляем контент только по презентационным причинам и всего лишь подсказываем некоторым пользователям, что этот элемент обязателен для заполнения. Астериск не несет никакой дополнительной информации о самом элементе формы.

Когда к работе подключаются атрибуты HTML5, помогающие создавать различные состояния элементов управления в формах, можно использовать CSS. С помощью таблиц стилей удобно оформлять обязательные, отключенные, отмеченные элементы, те элементы, которые предназначены только для чтения, валидные, невалидные, находящиеся в фокусе элементы, те, на которые наведен указатель мыши и т. д. Существуют разнообразные псевдоклассы, позволяющие оформлять элементы в зависимости от их состояния, в частности:

```
input:required,
input:invalid {
    background-color: #FFFFFF;
border: 1px solid #FF0000;
}
input:valid {
  border: 1px solid #999999;
}
input:read-only {
  background-color: #DDDDDD;
  border: 1px solid #666666;
}
input:checked + label {
  color: #666666;
  font-style: italic;
}
```

Таблицы стилей будут рассмотрены в главах 6–8. Поэтому, если вам пока не хватает знаний о псевдоклассах и элементах, можете забежать вперед и вернуться к этой главе, когда освоите всю необходимую информацию.

Добавив атрибут ARIA `aria-required="true"`, а также указав в атрибутах `title` или `aria-labeledby` инструкции по вводу данных, вы добьетесь еще более доступной организации сайта.

Новые элементы форм

Существует еще несколько элементов форм, которые мы пока не успели обсудить. В HTML5 их пять: `<datalist>`, `<output>`, `<keygen>`, `<progress>` и `<meter>`. Кроме того, мы обсудим новые черты некоторых других элементов, появившихся до выхода HTML5.

Элемент <datalist> и атрибут list

При работе с полями для ввода информации типа `text`, `email`, `url`, а также информации, относящейся к обозначению даты, времени и прочих числовых разновидностей элемента `<input>`, применяется атрибут `list`. Он указывает дополнительную информацию, которая должна предоставляться пользователю на выбор при вводе произвольного значения.

В качестве значения атрибут `list` принимает `id` ассоциированного с ним элемента `<datalist>`. Новый элемент `<datalist>` появился только в HTML5.

Элемент `<datalist>` содержит несколько элементов `<option>`, в которых заданы заранее определенные опции для элементов `<input>`. Речь идет об элементах `<input>`, которые ассоциируются с элементом `<datalist>` и делают это с помощью атрибута `list`, сопровождающего каждое поле формы. Однако каждый элемент формы, имеющий атрибут `list`, может быть ассоциирован только с одним элементом `<datalist>`.

В элементе `<datalist>` содержится список значений данных, которые представлены как ряд элементов `<option>`. Если у элемента формы есть атрибут `list`, то опции в составе `<datalist>` содержат определяемые автором значения автозавершения для данного элемента управления. Пользователь все равно может вводить значения самостоятельно в свободной форме, но опции предоставляются в виде `<select>` (рис. 4.12):

```
<p>
<label for="url">Animals: </label>
<input id="animals" type="text" placeholder="animals and sounds"
  requiredlist="animalnames" name="animals"/>
</p>
<datalist id="animalnames">
      <option value="quack">duck</option>
      <option value="banana slug" label="sssss"/>
      <option value="sheep" label="bah"/>
      <option value="horse" label="neigh"/>
</datalist>
```

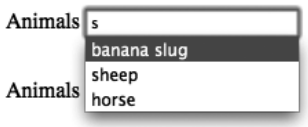


Рис. 4.12. Так выглядит элемент <datalist> в браузерах, которые его поддерживают (скриншот сделан в браузере Opera 10.6)

Функционально этот элемент похож на варианты завершения запроса, предлагаемые в поисковике Google. При работе с `<datalist>` список предлагаемых терминов может быть жестко запрограммирован либо генерируется динамически. Когда пользователь начинает вводить текст, варианты из списка данных сопоставляются с вводимыми символами. Подходящие данные выстраиваются в виде раскрывающегося меню, состоящего из значений элементов `<option>`, входящих в состав `<datalist>`. Как при автозавершении поисковых запросов, применяемом в Google, список может динамически обновляться. При этом задействуются AJAX-запросы.

Постепенное подстраивание при работе с элементом <datalist>. В содержимом элемента может присутствовать резервный контент для тех браузеров, которые еще не поддерживают `<datalist>`. Поскольку разные пользователи могут работать с разными браузерами, у них сохраняется возможность вводить данные в произвольной форме либо выбирать их из списка вариантов, перечисленных в элементах `<option>`. Эти варианты представляют собой значения, заранее заданные или рекомендуемые для элементов форм, ссылающихся на данный список. Каждый вариант имеет надпись и значение. По умолчанию элемент `<datalist>` и его потомки не отображаются (являются скрытыми).

В элементе `<datalist>` содержатся варианты для заполнения элемента формы, но также пользователь может свободно вводить информацию в данное поле формы. Это очень удобное усовершенствование, которое обязательно стоит реализовывать для работы с унаследованными браузерами. Чтобы упростить работу пользователям таких агентов, которые не поддерживают `<datalist>` (а именно так обстоит ситуация в большинстве мобильных браузеров), заключайте предлагаемые на выбор функции в элементе `<select>`. Если браузер поддерживает элемент `<datalist>`, то он просто проигнорирует все функции, не входящие в состав этого элемента, и отобразит понятный ему контент:

```
<p>
  <label for="url">Web Address: </label>
  <input id="url" type="url" placeholder="http://www.domain.com"
    requiredlist="mydatalist" name="url"/>
</p>
<datalist id="mydatalist">
  <p><label>Or select from the list</label>
    <select name="url2">
      <option value=http://www.standardista.com label="Standardista"/>
      <option value="http://www.oreilly.com" label="O'Reilly"/>
      <option value="http://www.evotech.net" label="Evolution
       Technologies"/>
```

```
    </select>
  </p>
</datalist>
```

Резервный контент будет отображаться лишь в тех браузерах, которые не поддерживают <datalist> (рис. 4.13). В браузерах с поддержкой <datalist> элементы <p>, <label> и <select> будут игнорироваться. Все дочерние элементы <datalist>, кроме <option>, будут выглядеть как на рис. 4.12. Если при выполнении постепенного подстраивания вы стараетесь обходиться без JavaScript, убедитесь в том, что сервер правильно настроен для получения данных из обеих разновидностей полей форм.

Рис. 4.13. В данной реализации <datalist> применяется постепенное подстраивание. Если браузер не поддерживает <datalist>, то отображается лишь содержимое потомков <datalist>, а непонятный элемент просто игнорируется. Обратите внимание: конфигурация клавиатуры специально предназначена для ввода URL, элементы <p> и <datalist> видимы. Нажав кнопку Next (Далее), вы активизируете меню для выбора данных в <datalist>

Есть еще более удобный резервный вариант, который, однако, требует небольшого количества JavaScript. Код HTML выглядит так:

```
<input type="text" name="animal" list="dl_animals" id="animals" />
<datalist id="dl_animals">
  <select id="slct_animal">
    <option value="moo">Cow</option>
```

```
    <option value="ssssss">Banana Slug</option>
    <option value="bah">Sheep</option>
  </select>
</datalist>
```

А вот соответствующий JavaScript:

```
var select = document.getElementById('slct_animal'),
  input = document.getElementById('animals');
select.addEventListener('change', function() {
  input.value = this.value;
}, false);
```

и CSS:

```
input[list],
datalist select {
  float: left;
  height: 1.4em;
  position: relative;
}
input[list] {
  z-index: 2;
  width: 20em;
}
datalist select {
  width: 1.2em;
}
```

В предыдущем примере мы добавили `select` без атрибута `name`, поэтому `select` не будет отправлен вместе с формой. Мы также оформили `select` таким образом, чтобы он выглядел как блок для выбора элементов из списка (select spinner). Этот блок находится справа от поля ввода, снабженного атрибутом `list`.

Элемент <output>

Элемент `<output>` функционально подобен `<span>`, с той оговоркой, что на уровне DOM он считается элементом формы. Элемент `<output>`, появившийся только в HTML5, может иметь атрибуты `form`, `name` и `for`, а также сопровождаться событиями `onchange`, `onforminput` и `onformchange`; кроме того, он может иметь любые универсальные атрибуты и обработчики событий.

У элемента `output` нет атрибута `value`. Значение элемента определяется не этим атрибутом, а тем содержимым внутри строки, которое находится между открывающим и закрывающим тегами. Элемент `<output>` обязательно должен иметь и открывающий, и закрывающий тег. Его значение можно как устанавливать, так и получать на уровне DOM.

Значение атрибута `for` у элемента `<output>` немного отличается от значения одноименного атрибута у элемента `<label>`. Атрибут `for` элемента `<output>` принимает в качестве значения разделенный пробелами список ID других элементов, ассоциированных с данным полем вывода.

Элемент <output> следует использовать в случаях, когда пользователю гарантированно не придется напрямую манипулировать значением и когда значение выводится из других, например является результатом тождества, построенного из значений элементов, перечисленных в атрибуте for.

CubeeDoo. В игре CubeeDoo мы будем использовать элемент формы <output> для вывода следующей информации: очки, уровень игры, количество секунд, оставшихся до конца раунда. В нашем примере с кодом актуальные значения этих параметров содержатся в элементе <output> и обновляются путем манипуляций с DOM. HTML-разметка в данном случае очень простая:

```
<output name="score">0</output>
<output name="level">1</output>
<output name="time">120</output>
```

Мы предварительно заполняем исходные значения, так как, когда страница загрузится перед началом игры, у пользователя будет 0 очков, он будет находиться на первом уровне, а время до конца первого раунда составит 2 минуты. С помощью JavaScript мы будем динамически поддерживать и обновлять значения.

<meter>

Как правило, элемент <meter> используется в качестве датчика, позволяющего выполнять измерения в известном диапазоне. Элемент <meter> применяется для указания того, как актуальное значение соотносится с максимальным и минимальным. Поэтому он напоминает датчик со стрелкой. Возможные варианты применения такого датчика — создание графических индикаторов надежности пароля, вводимого пользователем, а также подобных индикаторов для визуальной обратной связи.

Атрибуты элемента <meter>, принимающие в качестве значений числа с плавающей точкой, — это min, max, high, low и optimum. В атрибуте optimum указывается значение той точки, которая считается идеальной позицией стрелки датчика. Атрибуты min и max соответствуют минимальному и максимальному значениям. Атрибутами high и low должны сопровождаться соответственно минимальное значение, уже относящееся к области высоких, и максимальное значение, еще относящееся к области низких.

В приведенном далее пользовательском интерфейсе показан датчик, минимальные значения на шкале которого расположены слева, максимальные — справа. Разноцветная полоса заполняет датчик слева направо пропорционально в зависимости от значения. Внешний вид этой полосы зависит от актуального значения применяемых атрибутов и конкретного браузера. Полоса может быть красной, зеленой или желтой. Она приобретает зеленый цвет, если значение датчика отказывается в диапазоне между high и low. Если же происходит выход за пределы этого диапазона, то полоса приобретает желтый или красный цвет. Цвет полосы связан со значением optimum (рис. 4.14).

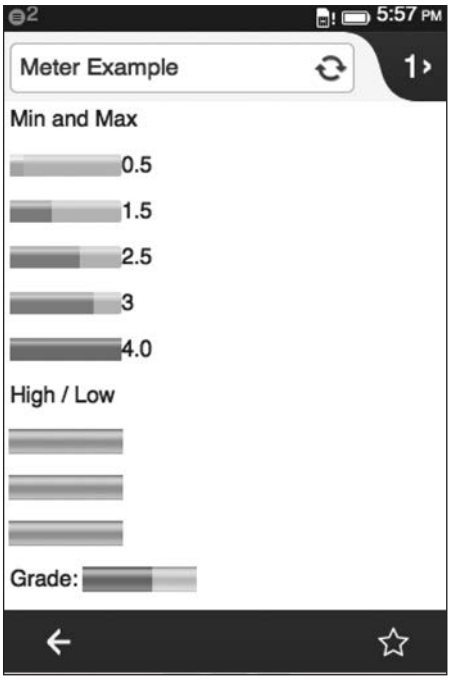


Рис. 4.14. Пользовательский интерфейс и цвета <meter>

Элемент <meter> не должен применяться для отображения протекания процессов. Для этой цели существует специальный элемент <progress>. Пользуйтесь <meter>, если вам точно известны минимальное и максимальное значения, а также если значения в диапазоне могут повышаться или понижаться, например при измерении артериального давления или количества топлива в бензобаке. Если значение должно изменяться только в одном направлении, пользуйтесь элементом <progress>.

<progress>

Элемент <progress> напоминает <meter>, но действует не как датчик, а как индикатор продвижения к определенной цели или к решению задачи. В отличие от <meter>, где актуальное значение отображается относительно минимального и максимального, индикатор <progress> указывает, насколько далеко продвинулся процесс выполнения задачи и сколько примерно времени осталось до его завершения. Например, элемент <progress> может использоваться для отображения скорости протекания функции на JavaScript, на выполнение которой требуется существенное количество времени (рис. 4.15).

Элемент <progress> принимает значение value и атрибут max. Оба этих значения являются положительными числами с плавающей точкой, причем value должно быть меньше max.

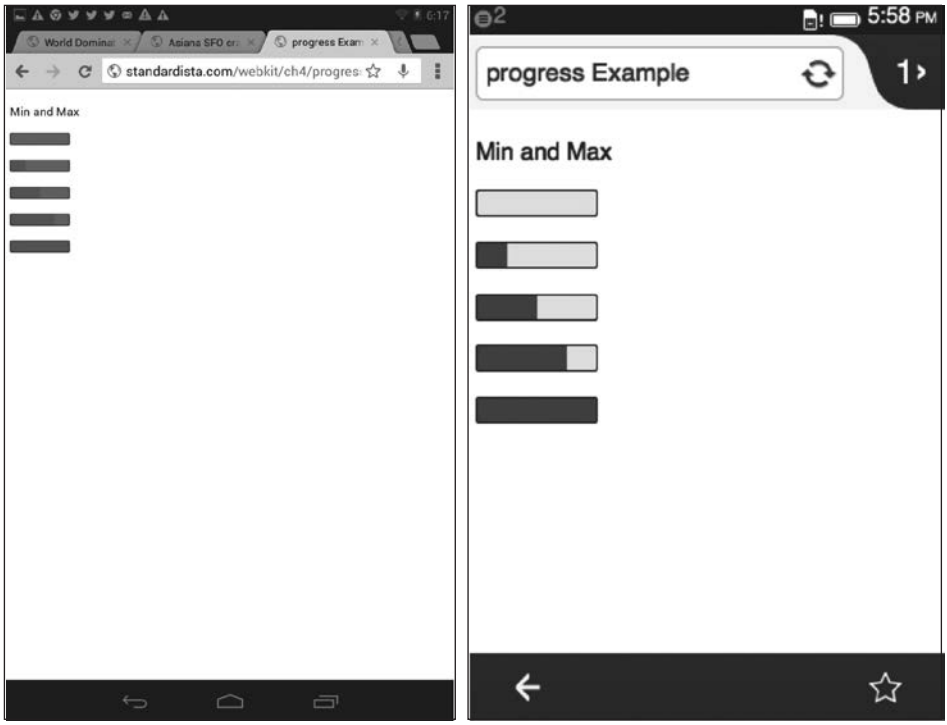


Рис. 4.15. Элемент <progress> в Firefox OS и Chrome для Android

<keygen>

Самозакрывающийся элемент <keygen> представляет собой поле ввода, которое одновременно служит генератором пары «ключ/значение». Элемент <keygen> удобен для разработчиков клиентских приложений и используется с протоколами авторизации. Элемент <keygen> применяется для генерации пары, состоящей из открытой и закрытой частей ключа и для отправки его открытой части. Если элемент <keygen> принимает атрибуты `challenge`, `keytype`, `autofocus`, `name`, `disabled` и `form`, значение `keytype` равно `rsa`, а атрибут `challenge` принимает в качестве значения строку запроса, отправляемую вместе с открытым ключом. Этот элемент поддерживается в браузерах Opera, WebKit и Firefox. Он отображается как меню для выбора, в котором можно генерировать зашифрованные ключи; предоставляются также другие возможности.

Другие элементы форм

В следующих разделах кратко описаны различные элементы форм. Возможно, вы знакомы со многими из этих элементов, но я описываю их здесь, чтобы подчеркнуть новые возможности HTML5.

<form>

Элемент <form> претерпел ряд изменений. Теперь форма автоматически валидирует типы ввода при отправке. Появился новый логический атрибут novalidate (novalidate="novalidate"), обеспечивающий нативную валидацию формы при отправке.

Теперь элементы форм могут и не быть потомками вышестоящего элемента <form>. В данной версии элементы форм могут сопровождаться атрибутом form, указывающим, с какой формой на странице ассоциирован данный элемент. Кроме того, у элемента <form> появился атрибут autocomplete.

Также необходимо отметить, что теперь можно обходиться без атрибута action. Если он пропущен, то форма отправится сама в себя, как если бы значением action была текущая страница.

<fieldset> и <legend>

Элемент <fieldset> объединяет в группу несколько элементов формы. Опциональным первым потомком <fieldset> может быть элемент <legend>, дающий имя группе. Остальные потомки элемента <fieldset> представляют собой компоненты этой группы. Элемент <legend> выступает в качестве заголовка для того содержимого, которое вместе с ним находится в <fieldset>. Таким образом, <legend> может быть только потомком <fieldset> и должен иметь закрывающий тег.

Если указан логический атрибут disabled, то он отключает все дочерние элементы <fieldset>, за исключением потомков <legend>. Можно использовать атрибут form, позволяющий ассоциировать элемент <fieldset> с конкретной формой (подробнее об этом — в описании атрибута form). Атрибут name представляет собой имя элемента.

<select>, <option>, <optgroup>

Тег <select> указывает меню для выбора элементов. В элементе <select> должны содержаться один или несколько элементов <option> либо один или несколько элементов <optgroup>, в которых находятся элементы <option>. В браузере Safari, где для этого тега явно задается атрибут size, поле ввода напоминает комбинированный список. В других браузерах это поле больше похоже на всплывающее меню.

<textarea>

<textarea> — это элемент формы для ввода текста в свободном формате, номинально не имеющий ограничений, связанных с разрывами строк. Этот элемент представляет собой прокручиваемое поле для многострочного текстового ввода.

В HTML5 появился новый атрибут wrap. Атрибут wrap может сопровождать элемент <textarea> и иметь значение soft (по умолчанию) или hard. Вариант soft означает, что отправляемый текст будет содержать лишь такие разрывы строк, которые мог задать пользователь, а значение hard допускает и наличие автоматических разрывов строк. Если вы устанавливаете wrap в hard, то указывайте и атрибут cols.

В HTML 4 необходимо было указывать, каков будет размер элемента <textarea> на экране, задавая значения для rows и cols. В HTML5 атрибуты rows и cols элемента <textarea> уже не являются обязательными, как в HTML 4. Исключение — случай, когда атрибут wrap имеет значение hard, тут атрибут cols становится обязательным. В любых других ситуациях атрибуты rows и cols опциональны. Для определения высоты и ширины элементов следует использовать CSS. Закрывающий тег обязателен.

<button>

Элемент <button> может относиться к одному из трех типов: submit, reset и button (по умолчанию — submit). В отличие от <input type="button"/> элемент <button> не является самозакрывающимся. Необходимо ставить закрывающий тег </button>. Этот элемент по сравнению с предыдущими версиями не изменился.

<label>

Элемент <label> не является новинкой в HTML5, но, поскольку он часто используется не по назначению, остановимся на нем отдельно.

В элементе <label> находится название-метка, сопровождающее элемент формы в пользовательском интерфейсе. Это название ассоциируется с конкретным элементом формы одним из двух способов. Если требуется создать явную метку, то для этого применяется атрибут for. Однако можно создать и неявную метку, для этого нужно расположить элемент формы внутри элемента <label>.

Значение атрибута for у явной метки должно совпадать со значением атрибута id элемента формы.

Необходимо отметить, что отношение «элемент формы/метка» не способствует доступности экранных дикторов. При нажатии на метку или прикосновении к ней на сенсорном экране в ассоциированном с ней элементе формы происходит событие щелчка. Так, при прикосновении к метке флажка этот флажок переключается в противоположное состояние. Соответственно, поле формы становится более доступным для всех пользователей, а не только для тех, кто работает с экранными дикторами или закадровым текстом. При щелчке на метке, ассоциированной с положением переключателя, или прикосновении к ней это положение становится отмеченным. При прикосновении к элементу <label>, ассоциированному с текстовым полем, это поле попадает в фокус и пользователь может ввести туда данные.

Резюме

Когда элементы и атрибуты полей ввода HTML5 будут полностью поддерживаться, на сайтах не будет такой острой необходимости в применении JavaScript для клиентской валидации, поскольку большую часть этой сложной работы станет выполнять сам браузер. Однако серверы по-прежнему будут выполнять валидацию данных. Поэтому у злоумышленников пока остаются лазейки, позволяющие обходить клиентскую проверку типов и валидацию.

5 Масштабируемая векторная графика, холст, аудио и видео

Мы уже обсудили большую часть новых элементов HTML5, за исключением тех, которые уникально связаны с сетевыми API, пока находящимися в разработке, и еще одной категории элементов, о которых пойдет речь в этой главе. Это хорошо поддерживаемые элементы, относящиеся к работе с медиа и предназначенные для обращения с масштабируемой векторной графикой (SVG), холстом (Canvas), аудио и видео. Элементы, связанные с вышеупомянутыми веб-API, постоянно изменяются, поэтому мы не будем рассматривать их в этой книге. Элементам из второй категории посвящена эта глава.

Здесь будут рассмотрены основные возможности, которые пригодятся в повседневной практике веб-разработчику клиентского кода. Вы научитесь применять все новые возможности при программировании для мобильных браузеров. Все современные мобильные браузеры (за исключением Opera Mini) поддерживают элементы `<canvas>`, `<video>` и `<audio>`, а также веб-API, обеспечивающие геолокацию, работу с локальным хранилищем данных, создание офлайновых веб-приложений и т. д.

Можно было бы написать по книге на каждую из отдельных тем, рассмотренных в этой главе, — правда, на большинство тем такие книги уже есть. Надеюсь, что смогу дать здесь достаточное количество ознакомительной информации для того, чтобы читатель мог сам принять решение: «Ага, почитаю-ка книгу на эту тему» или «Нет, пока мне это неинтересно». Мы не будем углубленно исследовать ни одну из этих тем, но вы приобретете достаточно знаний, чтобы приступить к самостоятельному изучению того, что вас заинтересует. Гораздо важнее, что вы узнаете о достоинствах и недостатках всех этих технологий в мобильной сфере.

API HTML5 для работы с мультимедиа

Исходная спецификация HTML описывала работу исключительно с текстовым контентом. В ней даже отсутствует информация об элементе `<img>`. С тех пор немало воды утекло. HTML5 позволяет создавать масштабируемую векторную графику по технологии SVG, а также имеет элемент `<canvas>`, в котором содер-

жится пустое пространство для отрисовки — так называемый холст. Кроме графики, HTML5 поддерживает также включение в разметку элементов <video> и <audio>, причем для этого не требуется никаких сторонних плагинов.

SVG

С помощью технологии SVG можно создавать сложные изображения в формате *масштабируемой векторной графики*. Технология SVG появилась в 2001 году и представляет собой открытый стандарт для определения двухмерной векторной графики. «Масштабируемый» аспект SVG означает, что одна и та же графика будет иметь одинаковую резкость и на большом мониторе, и на маленьком дисплее мобильного устройства, не требуя никаких дополнительных изменений.

В спецификации SVG определяется XML-грамматика для фигур, линий, кривых, изображений и текста, в частности такие возможности, как прозрачность, произвольная геометрия, эффекты фильтров (тени, подсветка и т. д.), работа со сценариями и анимация.

Поскольку речь идет о текстовом формате для работы с изображениями, файлы в этом формате зачастую получаются очень маленькими. Такой файл имеет объектную модель, и ее можно изменять с помощью сценариев. Изображения векторные, и поэтому они масштабируются без возникновения мозаичности и зазубренных краев. Этот язык декларативен, и поэтому его легко понять. Вдобавок SVG поддерживает анимацию.

Существуют различные формы SVG, степень их подержки в браузерах различна. Базовая поддержка любых файлов с расширением .svg существует во всех современных браузерах для мобильных устройств с Android, в версии Android 3 и выше. SVG в качестве исходного кода для элемента <img> поддерживается в iOS 3.2 и выше, Android 3.0 и выше и в мобильной версии IE 8 и выше.

Формат файлов SVG в качестве значения для свойства CSS background-image поддерживается в Android 3 и выше, iOS 3.2 и выше, а также давно поддерживается в Opera Mobile. Мы даже можем использовать HTML-элемент <svg> на страницах HTML5 в iOS 5 и выше, Android 3 и выше, IE 9 и выше (а также во всех других современных браузерах). Android 2.3.3 и ниже, Amazon Silk и уже практически исчезнувшая система WebOS от HP — вот и все мобильные браузеры, в которых отсутствует полная поддержка SVG. Статическая SVG поддерживается даже в Opera Mini (как и элемент <canvas>, поддерживаемый во всех мобильных браузерах), но такая SVG не анимируется, поскольку в Opera Mini нет достаточной для этого поддержки JavaScript.

Как и в любом языке на основе XML, корневой элемент SVG — не <html>. В данном случае корневым элементом является <svg>. Как и все XML-документы, файл SVG начинается с XML-пролога и соответствующего объявления типа документа (SVG DTD). В корневом элементе <svg> содержится весь контент документа. В SVG не используются элементы <head> и <body>. В данном случае весь контент, в том числе все вложенные элементы <svg>, содержится в корневом элементе <svg>.

Начнем знакомство с SVG с изображения японского флага. Он представляет собой простой белый прямоугольник с красным кругом (солнцем) в центре (рис. 5.1).

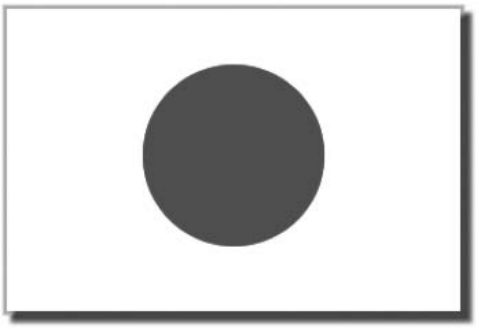


Рис. 5.1. Японский флаг, созданный по технологии SVG

```
1  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
2        "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
3  <svg xmlns="http://www.w3.org/2000/svg" height="220" width="320"
   version="1.0">
4    <title>Японский флаг</title>
5    <desc>Красный круг на белом флаге</desc>
6    <rect x="10" y="10" width="300" height="200"
7      style="fill: #ffffff; stroke: #e7e7e7;"/>
8  <circle cx="160px" cy="107px" r="60px" fill="#d60818"/>
9  </svg>5
```

Итак, что же это все означает? В строках 1–3 содержатся объявление документа SVG DTD и, соответственно, корневой элемент `<svg>`. О корневом элементе необходимо отметить следующее: размер векторной графики специально объявляется. Чтобы вы могли использовать свойство CSS `background-position` с фоновыми изображениями типа SVG, размер SVG должен быть специально объявлен. Это важно и при создании спрайт-файла на SVG.

Можете воспользоваться элементом `<title>` (строка 4), если файл SVG применяется независимо от других ресурсов. В строке 5 расположен элемент `<desc>`. Здесь можно поместить текстовое описание, которое не будет нативно отображаться в браузере, если уже отображается SVG. Если записать информативный контент в `<desc>` или `<title>`, то значительно повысится доступность ресурса. Поскольку SVG поддерживается не во всех экранных дикторах, для повышения доступности можно добавить атрибут `aria-label`.

В строке 6 находится элемент `<rect>`, соответствующий прямоугольнику. В SVG доступны следующие виды фигур и линий: `<path>`, `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<polyline>` и `<polygon>`. Мы располагаем четырьмя значениями атрибутов: `x`, `y`, `width` и `height` — для отступа по оси *X*, отступа по оси *Y* (координаты размещения фигуры

или линии), обозначения ширины и высоты соответственно. Кроме того, здесь применяется атрибут `style`.

Как и в обычном HTML-документе, в документе SVG можно использовать CSS для оформления элементов этого документа. Стили могут объявляться внутристрочно с помощью атрибута `style`, как было сделано в предыдущем примере. В качестве альтернативы можно включить в код встроенную или внешнюю таблицу стилей, указывая элементы с помощью селекторов, — точно так же, как это делается в HTML.

Имена свойств CSS немного отличаются от тех, к которым вы привыкли, но они приспособлены для чтения человеком. Свойство `fill` напоминает `background`. Оно обеспечивает цвет фона. Свойство `stroke` похоже на принятое в CSS свойство `border`. Мы могли бы задать здесь градиент или узор.

На практике существует возможность использовать большинство CSS-свойств и их значений в SVG-файлах. Однако по причинам, связанным с безопасностью содержимого, некоторые производители браузеров[1] не допускают импорта растровых изображений или сценариев при работе с файлами SVG, когда такие файлы включаются в страницу как изображения переднего плана в теге `<img>`.

В элементе `<circle>` (строка 8) описан круг красного цвета. У элемента `<circle>` вместо ширины и высоты есть атрибут `r`, означающий радиус. Элемент `<circle>` размещается на экране не в левом верхнем углу, как `<rect>`, а по центру круга. Значение `cx` — это координата по оси *X*, соответствующая центру круга, `cy` — координата по оси *Y*, соответствующая центру круга.

Внимательно изучив атрибуты, вы заметите, что параметр `fill` использован с элементом `<rect>` как свойство CSS, а с элементом `<circle>` — как атрибут.

Включение SVG в ваши документы

Можно включить масштабируемую векторную графику непосредственно в документ, воспользовавшись одним из следующих трех элементов: `<img>`, `<object>` и `<embed>`:

```
<img src="flag.svg" alt="Японский флаг"/>
```

или так:

```
<embed type="image/svg+xml" src="flag.svg" width="320" height="220"/>
```

или так:

```
<object data="flag.svg" type="image/svg+xml" width="320" height="220"></object>
```

Обратите внимание: хоть у элементов `<embed>` и `<object>` нет атрибута `alt`, SVG можно сделать доступной. Для улучшения доступности можно описать иллюстрацию в элементе `<desc>` или `<title>` либо добавить атрибут `aria-label` со значением,

[1] В настоящее время в браузерах WebKit и Mozilla не допускается импорт сценариев и растровых изображений в файлах SVG с применением тега <img>, даже если SVG и растровые изображения взяты из одного и того же источника.

соответствующим данному SVG-заголовку. Указывая высоту и ширину для <svg>, эти значения можно не включать в качестве атрибутов в <img>, <embed> или <object>, но необходимо включать в CSS.

Метод «автомобиль клоуна»: SVG для адаптивных изображений переднего плана

Масштабируемая векторная графика может использоваться для создания и подачи адаптивных изображений. Можно активно использовать браузерную поддержку SVG, а также предоставляемую в SVG поддержку медиазапросов и растровых изображений, чтобы выводить на экран нужный рисунок.

По опыту работы с фоновыми изображениями в CSS нам известно, что действительно можно загрузить только нужные картинки. Аналогично, чтобы SVG не скачала сразу все включенные изображения, используем в нашем SVG-файле фоновые картинки CSS, а не изображения переднего плана. При работе с адаптивными SVG мы включаем все файлы изображений, которые, возможно, понадобится предоставить, а потом показываем только нужные изображения в зависимости от поступающих медиазапросов (медиазапросы будут подробнее рассмотрены в главе 7):

```
<svg xmlns="http://www.w3.org/2000/svg"
  viewBox="0 0 300 329" preserveAspectRatio="xMidYMid meet">

<title>Поместите сюда старые атрибуты</title>

<style>
  svg {
    background-size: 100% 100%;
    background-repeat: no-repeat;
  }

  @media screen and (max-width: 400px) {
    svg {
      background-image: url(images/small.png");
    }
  }

  @media screen and (min-width: 401px) and (max-width: 700px) {
    svg {
      background-image: url(images/medium.png);
    }
  }

  @media screen and (min-width: 701px) and (max-width: 1000px) {
    svg {
      background-image: url(images/big.png);
```

```
    }
  }

  @media screen and (min-width: 1001px) {
    svg {
      background-image: url(images/huge.png);
    }
  }
</style>
</svg>
```

Чтобы сохранить нужное соотношение сторон объемлющего элемента и гарантировать, что он будет масштабироваться правильно, используются атрибуты `viewbox` и `preserveAspectRatio`. Значение атрибута `viewbox` — это четыре числа, разделенные пробелами или запятыми: `min-x` (минимум по оси *X*), `min-y` (минимум по оси *Y*), `width` (ширина) и `height` (высота). Определяя значения ширины и высоты области просмотра (viewbox), мы определяем соотношение сторон в SVG-файле.

Поскольку существуют некоторые проблемы безопасности при работе с `<img>` и импортированием растровых картинок в SVG, используем тег `<object>` для помещения на сайт адаптивных изображений. Элемент `<object>` позволяет интерпретировать внешний ресурс как изображение:

```
<object data="awesomefile.svg" type="image/svg+xml"></object>
```

По умолчанию `<object>` должен быть равен по ширине своему родительскому элементу. Но как и при работе с изображениями, мы можем задать высоту и ширину с помощью атрибутов `width` и `height` либо свойств CSS `width` и `height`. Поскольку в SVG-файле содержатся объявления `viewbox` и `preserveAspectRatio`, тег `<object>` по умолчанию будет поддерживать заданное соотношение сторон лишь при условии, что указана любая из величин, `width` или `height`.

Поскольку в данном случае используется тег `<object>`, а не `<img>`, мы не располагаем атрибутом `alt`. Чтобы обеспечить доступность такого метода, когда (и если) экранные дикторы начнут поддерживать SVG[1], необходимо гарантировать, что содержимое элемента `<title>`, используемого с SVG, в точности соответствовало информации, которую вы включили бы в атрибут `alt`.

В теге `<object>` SVG встраивается на страницу. SVG извлекает из имеющегося набора такое фоновое изображение, которое соответствует запросу `@media`. Нужное изображение определяется по размеру `<object>`, а не по величине области видимости. В приведенном ранее коде выполняются два HTTP-запроса: один для получения SVG, а другой — для получения изображения подходящего размера. Чтобы уложить всю эту информацию в единственный HTTP-запрос, указывайте в качестве значения атрибута `data` элемента `<object>` экранированный URI данных[2].

---

[1] http://www.iheni.com/just-how-accessible-is-svg/

[2] Экранирование URI данных требуется для работы с браузером IE9 и выше. Это лишь краткий обзор метода «автомобиль клоуна». Подробное описание, примеры, а также резервные варианты для браузеров, не поддерживающих SVG, приводятся по адресу https://github.com/estelle/clowncar

Я называю этот метод «автомобиль клоуна», так как мы умещаем множество больших изображений (клоунов) в один файл изображения SVG (автомобиль).

Изучение SVG

Пока мы лишь слегка коснулись темы SVG. Масштабируемую векторную графику можно сделать доступной, она подстраивается под любое разрешение экрана (то есть масштабируется), а также поддерживает анимацию на основе синтаксиса SVG или с применением JavaScript. Полный контроль над каждым элементом обеспечивается благодаря API объектной модели SVG-документа. Возможности SVG настолько широки, что детальное их обсуждение выходит за рамки этой книги. В спецификации W3C подробнее рассказано обо всех этих элементах, атрибутах и анимационном API.

Японский флаг — пример очень простого SVG. Обычно файлы SVG гораздо более сложные. Если вы имеете опыт работы с Adobe Illustrator, то, наверное, знаете, что эта программа позволяет экспортировать иллюстрации в формате SVG. Хотя это и удобный способ создания высокоточных SVG-файлов, при таком подходе создается много кода и работа с программой оказывается затратной.

Amaya — свободное ПО, поддерживающее упрощенную версию SVG, в частности простейшие фигуры, текст, изображения, технологию foreignObject, альфа-прозрачность, трансформации и анимацию. Программу Amaya можно скачать прямо на сайте W3C. Amaya помогает научиться работать с SVG, так как в этой программе можно просматривать и редактировать исходный код. Также можете познакомиться с Inkscape — свободно распространяемым редактором векторной графики, функционально схожим с Illustrator, CorelDraw или Xara. В Inkscape используется формат SVG, соответствующий стандарту W3C.

Масштабируемая векторная графика для игры CubeeDoo

В CubeeDoo мы дважды воспользуемся SVG. Во-первых, у нас будет SVG-спрайт для фонового изображения «фигурной» темы нашей игры. Во-вторых, будем работать с URI данных SVG для значка отключения звука.

Мы предлагаем пользователю на выбор несколько тем. В частности, есть темы с числами, фигурами и цветовая тема. Мы сможем создавать фигуры с помощью простого SVG-спрайта. Код, использованный для создания этого спрайта (спрайт для лицевой стороны одной из карточных колод показан на рис. 5.2), таков:

```
1  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
2      "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
3  <svg xmlns="http://www.w3.org/2000/svg" height="400" width="400"
   version="1.0">
4    <desc>Квадраты, круги, ромбы и треугольники</desc>
```

```
5
6    <!-- Цветные квадраты -->
7    <rect x="10" y="10" width="80" height="80" style="fill: #d60818;"/>
8    <rect x="10" y="110" width="80" height="80" style="fill: #ffff33;"/>
9    <rect x="10" y="210" width="80" height="80" style="fill: #00FF00;"/>
10    <rect x="10" y="310" width="80" height="80" style="fill: #0000FF;"/>
11
12    <!-- Цветные круги -->
13    <circle cx="150" cy="50" r="40" style="fill: #d60818;"/>
14    <circle cx="150" cy="150" r="40" style="fill: #ffff33;"/>
15    <circle cx="150" cy="250" r="40" style="fill: #00FF00;"/>
16    <circle cx="150" cy="350" r="40" style="fill: #0000FF;"/>
17
18    <!-- Ромбы -->
19    <polygon points="250,10 210,50 250,90 290,50" style="fill:
    #d60818;"/>
20    <polygon points="250,110 210,150 250,190 290,150" style="fill:
    #FFFF33;"/>
21    <polygon points="250,210 210,250 250,290 290,250" style="fill:
    #00FF00;"/>
22    <polygon points="250,310 210,350 250,390 290,350" style="fill:
    #0000FF;"/>
23
24    <!-- Треугольники -->
25    <polygon points="310,10 350,90 390,10" style="fill: #d60818;"/>
26    <polygon points="310,110 350,190 390,110" style="fill: #FFFF33;"/>
27    <polygon points="310,210 350,290 390,210" style="fill: #00FF00;"/>
28    <polygon points="310,310 350,390 390,310" style="fill: #0000FF;"/>
29  </svg>
```

В строке 1 находится объявление типа документа (DTD). В строке 3 объявляется корневой элемент, в нее также включаются данные о высоте и ширине SVG-изображения. Хотя спецификации этого и не требуют, необходимо включить эти атрибуты, если вы планируете использовать SVG-изображение в качестве фонового. В строке 4 находится описание, повышающее не только доступность, но и поисковую оптимизацию.

В строках 7–10 находятся объявления для четырех квадратов. Строка 9 означает: «Построй прямоугольник, начиная с точки, расположенной в 10 пикселах от левого края и в 210 пикселах от верхнего края. Задай для прямоугольника ширину 80 пикселов и высоту 80 пикселов. Заполни этот прямоугольник цветом #00FF00».

```
<rect x="10" y="210" width="80" height="80" style="fill: #00FF00;"/>
```

В строках 13–16 определяются четыре круга (диска). Строка 16 означает: «Найди точку, расположенную в 150 пикселах от левого края и в 350 пикселах от верхнего края и сделай ее центром круга радиусом 40 пикселов. Задай для этого круга фоновый цвет #0000FF».

```
<circle cx="150" cy="350" r="40" style="fill: #0000FF;"/>
```

Рис. 5.2. SVG-спрайты фигур

В строках 18–28 определяются восемь многоугольников: четыре ромба и четыре треугольника. Для описания многоугольников определяются их вершины. Строка 19 означает: «У этого многоугольника четыре вершины, верхняя точка находится в 250 пикселах от левого края и в 50 пикселах от верхнего края. Вторая точка находится в 210 пикселах от левого края и в 50 пикселах от верхнего края. Нижняя точка находится в 90 пикселах от верхнего края, а самая правая — в 290 пикселах от левого края и 50 пикселах от верхнего края. Область, ограниченная этими четырьмя точками, должна быть заполнена цветом #d60818 — это оттенок красного».

```
<polygon points="250,10 210,50 250,90 290,50" style="fill: #d60818;"/>
```

Мы решили построить квадраты, круги, ромбы и треугольники, расположенные углом вниз.

Мы также могли бы включить эти маленькие картинки в виде URI данных непосредственно в CSS-файл либо оформить их как изображения переднего плана. Например, вот как можно включить закодированный SVG в виде URI данных:

```
background-image: url(data:image/svg+xml,%3Csvg%20xmlns%3D%22http%3A%2F%2F
www.w3.org%2F2000%2Fsvg%22%20version%3D%221.0%22%3E%3Crect%20x%3D%220%22%20y
%3D%220%22%20fill%3D%22%23abcdef%22%20width%3D%22100%25%22%20height
%3D%22100%25%22%20%2F%3E%3C%2Fsvg%3E);
```

В CubeeDoo у нас будет также значок для отключения звука. Вот URI данных для него:

```
background-image:
  background-image:
  url("data:image/svg+xml;utf8,%3Csvg%20xmlns=
'http://www.w3.org/2000/svg'%20width='100'%20height='100'%3E
```

```
%3Cpolygon%20points='39,13%2022,28%206,28%206,47%2022,48%2039,63%2039,14'%20
style='stroke:#111111;stroke-width:5;stroke-linejoin:round;
fill:#111111;'%20/%3E%3Cpath%20d='M%2048,50%2069,26'%20%20style='fill:none;
stroke:#111111;stroke-width:5;stroke-linecap:round'%20/%3E%3Cpath%20%20d='M%20
69,50%2048,26'%20style='fill:none;stroke:#111111;stroke-width:5;
stroke-linecap:round'%20/%3E%3C/svg%3E");
```

Пути в этом примере человеку прочитать почти невозможно. Они были созданы с помощью Amaya. Но синтаксис должен быть вам знаком. Здесь используется свойство CSS `background-image`. Вместо применения `url(path/mute.jpg)` или даже `url(path/mute.svg)` мы задействуем код `url("data:image/svg+xml;utf8,<svg.../></svg>");`. Весь SVG-файл экранирован и заключен в кавычки.

Для тех версий Internet Explorer, которые поддерживают SVG (в настоящее время это IE9 и IE10), URI данных должны экранироваться — этого требует спецификация.

Холст

Спецификация холста для HTML5 (HTML5 Canvas) — это API JavaScript для создания рисунков. API холста позволяет определять на HTML-странице объект контекста холста, представляемый в виде элемента `<canvas>`. В этом элементе можно рисовать. Можно даже включать в CSS-файлы изображения, отрисованные на холсте, и использовать их в качестве фоновых рисунков.

Можно рисовать как в двухмерном, так и в трехмерном (WebGL) контексте. Двухмерный контекст для рисования поддерживается во всех современных браузерах. WebGL все прочнее закрепляется в мобильном пространстве, но при необходимости эту технологию следует включать в программу только при наличии соответствующего аппаратного ускорения. Такое требование обусловлено соображениями производительности.

Двухмерный контекст для рисования предоставляет простой, но мощный API для выполнения быстрых операций отрисовки на плоской растровой поверхности. Специального файлового формата для этого не существует, вы можете рисовать только с помощью сценария. У вас не будет никаких узлов DOM для отрисовываемых фигур, ведь в `<canvas>` вы рисуете пикселы, а не векторы. При наличии единственного узла холст более удобен для использования на мобильных устройствах, но анимация JavaScript значительно нагружает процессор устройства и заряд батареи быстро расходуется. Однако эффективность использования батареи повышается, если применяется аппаратное ускорение.

Ваш первый <canvas>. Здесь будет сделано лишь базовое введение в работу с холстом, поэтому мы обсудим только простейшие фигуры и линии. Если вы не знаете JavaScript, то синтаксис на первый взгляд может показаться запутанным. Имеющим опыт работы с JavaScript разобраться будет гораздо проще.

Сначала нужно добавить элемент `<canvas>` в ваш документ. В HTML эта операция выполняется за один шаг:

```
<canvas id="flag" width="320" height="220">
  Ваш браузер не поддерживает холст. Иначе вы бы увидели флаг.
</canvas>
```

Вот и весь код холста, который относится к HTML. Я могла бы написать просто `<canvas></canvas>`. Атрибут `id` указан здесь для упрощения нацеливания при работе с JavaScript, хотя такое нацеливание можно реализовать и на уровне DOM. Кроме того, я включила сюда альтернативный контент для пользователей, у которых браузеры не поддерживают `<canvas>` либо содержимое `<canvas>` не видно по каким-то другим причинам.

ПРИМЕЧАНИЕ

На момент написания книги API элемента <canvas> реализован таким образом, что не поддерживает никаких средств обеспечения доступности, кроме атрибута aria-label.

Итак, мы создали пустое пространство для отрисовки, или холст. Все остальное будет происходить в коде JavaScript. В данном примере мы снова создадим японский флаг (рис. 5.3).

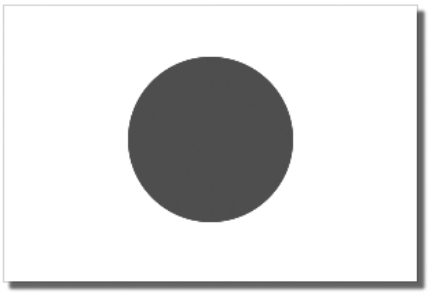


Рис. 5.3. Изображение японского флага, созданное на холсте

Приступим к рисованию на холсте. Далее вся работа идет только с кодом на JavaScript. Мы можем нацелиться на узел `<canvas>` с помощью простого JavaScript одним из следующих способов:

```
document.getElementById('flag')
document.getElementsByTagName('canvas')[0]
document.querySelector('#flag')
```

Затем инициализируем двухмерный контекст и начнем рисовать, пользуясь командами из API для двухмерных контекстов. Итак, рисуем японский флаг:

```
1  <script>
2  var el= document.getElementById("flag");
3
4  if (el && el.getContext) {
5    var context = el.getContext('2d');
6      if (context) {
7        context.fillStyle = "#ffffff";
```

```
8       context.strokeStyle = "#cccccc";
9       context.lineWidth = 1;
10      context.shadowOffsetX = 5;
11      context.shadowOffsetY = 5;
12      context.shadowBlur = 4;
13      context.shadowColor = 'rgba(0, 0, 0, 0.4)';
14      context.strokeRect(10, 10, 300, 200);
15      context.fillRect(10, 10, 300, 200);
16      context.shadowColor='rgba(0,0,0,0)';
17      context.beginPath();
18      context.fillStyle = "#d60818";
19      context.arc(160, 107, 60, 0, Math.PI*2, false);
20      context.closePath();
21      context.fill();
22      }
23  }
24  </script>
```

В строке 2 мы находим элемент <canvas> по его атрибуту id. Перед тем как создать двухмерный контекст, убеждаемся, что элемент-холст найден *и* браузер поддерживает холст. Для этого в строке 4 проверяется наличие метода getContext.

ПРИМЕЧАНИЕ

Можно задействовать сценарии обнаружения возможностей, например Modernizr. Они позволяют определить, поддерживает ли браузер работу с холстом и другие современные функции. Modernizr позволяет обнаруживать все доступные возможности либо отдельно взятые возможности, которыми вы планируете пользоваться. Здесь мы не применяем Modernizr, а рассматриваем, как обнаруживать возможности напрямую. Если только перед вами не стоит задача свести к минимуму использование внешних сценариев и HTTP-запросов, то Modernizr следует применять во всех случаях, когда это представляется целесообразным.

В строке 5 я создаю ссылку на контекст с помощью метода getContext(contextId) элемента-холста. 2d — это как раз тот контекст, который нужен для работы с <canvas>. Если удалось создать контекст, что и проверяется в строке 6, то мы наконец можем приступать к рисованию. Весь оставшийся код сценария занят именно рисованием.

Уже существует (пока чисто экспериментальная) возможность использовать рисунок с холста в качестве фонового изображения. В браузере WebKit это можно делать с помощью CSS[1], не вызывая элемент-холст на уровне DOM, а включая его в качестве фонового изображения:

```
background: -webkit-canvas(theCanvas);
```

код для CSS и:

```
var context = document.getCSSCanvasContext("2d", "theCanvas", 320, 220);
```

код для JavaScript. В коде для JavaScript второй параметр — это имя используемого холста (в коде CSS это имя указывается без кавычек).

[1] Firefox 4+ также поддерживает работу с холстом в CSS и позволяет динамически создавать виртуальный элемент-холст с применением элемента -moz-element('#myCanvas').

К 6-й и даже к 13-й строке мы еще ничего не успели нарисовать. Все, что мы сделали, — это определили контекст холста, на котором можем отрисовывать и перерисовывать пикселы.

Перед отрисовкой фигуры необходимо определить для нее желаемый внешний вид. Для этого зададим свойства объекта `context`. Мы определяем внешний вид границ (`stroke` и `linewidth`), цвета фона (`fill`) и теней (`shadowOff`, `setX`, `shadowOffsetY`, `shadowBlur` и `shadowColor`) первого прямоугольника. Этот прямоугольник рисуем с помощью метода `strokeRect()` в строке 14. Передаем такие же параметры, как и в более раннем примере с SVG: 10, 10, 300, 200. Эти четыре значения указывают соответственно отступ по оси *X*, отступ по оси *Y*, ширину и высоту.

Когда сценарий выполнит команду, он о ней сразу же «забывает» и переходит к следующей строке кода. В отличие от примера с SVG в предыдущем разделе, тот прямоугольник, который отрисован на холсте, не входит в состав объектной модели документа. Поскольку `stroke`, `fill`, `linewidth` и `border` — это свойства, программа их запоминает, но браузер и сценарий «не представляют», что уже отрисовано. Если вы хотите отслеживать, что и где рисуется на холсте, используйте в этом контексте метод `getImageData()`. Данный метод позволяет контексту получать соответствующие вашим пикселам значения красного, зеленого, голубого и альфа-прозрачности.

Когда в строке 15 мы рисуем второй прямоугольник с помощью метода `fillRect` (этот метод при отрисовке прямоугольников использует заданное ранее свойство `fillStyle`), требуется вновь сообщить координаты, так как объектная модель документа не сохранила никакой информации о первом прямоугольнике (хотя она и имеет доступ к данным о пикселах).

Вызовы обоих методов для работы с прямоугольниками (в строках 14 и 15) имеют одни и те же параметры: 10, 10, 300, 200. Мы нарисовали прямоугольник с заливкой прямо поверх прямоугольника с оттенением. Мы могли бы создать объект с такими координатами и передать его обоим методам, но не можем приказать холсту получить доступ к координатам первого прямоугольника и скопировать их во второй после вызова метода.

Сначала мы нарисовали контур прямоугольника, потом заполнили его цветом. Если бы мы действовали в обратном порядке, то оттенение располагалось бы поверх цвета заливки. Поскольку начало координат у двух прямоугольников находится в одной и той же точке, а ширина границы составляет всего 1 пиксел, то ширина результирующей границы будет равна всего 0,5 пиксела. Внутреннюю часть границы накроет заливка.

Как было указано ранее, когда мы приступаем к отрисовке диска (солнца) на флаге, DOM должна «вспомнить», что вы нарисовали, как только диск окажется на холсте. Действительно, JavaScript фиксирует значения установленных вами свойств, например значение `shadowColor`. Кроме того, программа запоминает последние шаги отрисовки независимо от того, произошла ли в итоге отрисовка. Однако пикселы, создаваемые на холсте, — это всего лишь разноцветные точки. Поскольку нам не нужна тень у красного круга, перед отрисовкой `shadowColor` мы должны сделать его прозрачным. Это выполняется в строке 16.

Команды по отрисовке круга начинаются с `beginPath()` (строка 17) и заканчиваются `closePath()` (строка 20). Сценарий запоминает шаги отрисовки независимо от того, оказались ли соответствующие элементы на холсте. Если мы нарисуем круг, а потом, не закрывая контекста, еще несколько линий, то все этапы отрисовки круга будут сохраняться в памяти и линия может пересечь круг — допустим, разделить его пополам. Чтобы избежать этого, мы открываем и закрываем отрисовку отдельных контуров командами `beginPath()` и `closePath()` соответственно.

Описываем круг: `context.arc(x-offset, y-offset, radius, startAngle, endAngle, anticlockwise)` добавляет точки для дугообразного контура, создавая виртуальную окружность. Это круг, описываемый аргументами `context.arc(160, 107, 60, 0,Math.PI*2, false);`. В качестве начальной точки выберем стартовый угол, который послужит нам правым горизонтом. Конечную точку определим по заключительному углу. Проведем линию от начального угла в заданном направлении (в нашем случае — по часовой стрелке). Если конечный угол окажется меньше 2π, то круг получится уплощенным: начальная и конечная точки будут соединены прямой линией. При значении π получится полукруг.

Кроме того, переопределим цвет заливки с белого на красный (строка 18). Затем закрасим отрисованный круг с помощью метода `fill()` (строка 21), который заполняет область под описанной дугой цветом `fillStyle`.

Здесь мы практически не коснулись возможностей `<canvas>`. Рекомендую посмотреть сайт http://ie.microsoft.com/testdrive/Graphics/CanvasPad/Default.html — на этой интересной странице можно научиться работать на холсте с простыми фигурами, цветами, тенями, текстом, изображениями, трансформациями, анимацией и движениями мыши.

Сравнение холста и масштабируемой векторной графики

Технологии Canvas HTML5 и SVG обладают некоторым сходством, их часто сравнивают, но нередко подчеркивают и различия между ними. Обе эти веб-технологии позволяют создавать в браузере насыщенную графику, но на самом деле между ними существует фундаментальная разница.

Как мы уже убедились, в SVG отрисовка происходит на языке XML. Напротив, на холсте мы рисуем с помощью JavaScript. На холсте рисуются отдельные пикселы: как только пиксел оказывается на своем месте, программа о нем «забывает». SVG же создает узлы объектной модели документа. Эти узлы остаются доступными до тех пор, пока не будут удалены либо пока пользователь не покинет страницу. У обеих технологий есть свои достоинства и недостатки.

Рисунки, выполненные в виде масштабируемой векторной графики, не зависят от разрешения экрана. Поэтому SVG отлично подходит для применения в пользовательских интерфейсах любых размеров, масштабируется и подстраивается под величину конкретного экрана. Файлы SVG создаются в формате XML, благодаря чему обеспечивается их доступность. SVG можно анимировать с помощью

декларативного синтаксиса или с применением JavaScript. Каждый элемент становится частью модели SVG DOM и доступен через SVG DOM API на языке JavaScript. Однако любая программа, многократно обращающаяся к объектной модели документа, замедляет работу веб-страницы, а это особенно заметно в мобильной среде.

Графика на холсте отрисовывается в виде пикселов. При увеличении картинка может становиться мозаичной. Холст по определению не располагает к доступности: обеспечение доступности сводится к включению резервного контента, который будет показан пользователю, если холст отобразить не удастся. Для обеспечения интерактивности требуется перерисовывать каждый пиксел. Ни для каких отрисовываемых объектов не предоставляются узлы DOM. Отсутствует анимационный API. Вместо этого для обновления холста с краткими интервалами используются таймеры или `requestAnimationFrame`. Холст представляет собой поверхность, для рисования на которой применяется API выбранного вами контекста. Однако холст очень удобен для редактирования изображений, генерирования растровой графики (например, для игр или фракталов) и для операций, требующих манипуляций с пикселами. Рисунки, созданные с применением API холста, также можно экспортировать как изображения.

Двухмерный контекст `<canvas>` хорошо поддерживается во всех браузерах (IE9 и выше). SVG также неплохо поддерживается, но в различных форматах (начиная с IE9 и Android 3). Несмотря на то что и SVG и Canvas поддерживаются хорошо, у них есть свои недостатки.

SVG может работать плохо. Мобильные браузеры могут пробуксовывать, если количество элементов DOM значительно увеличивается. На каждый дополнительный узел DOM тратится память, ее требуется пересчитывать, когда происходит перераспределение элементов на странице. По этим причинам следует ограничить количество узлов DOM, добавляемых в веб-приложения, если речь идет о разработке приложений для мобильных устройств. SVG состоит из узлов DOM, а повышенное количество элементов DOM может отрицательно сказываться на производительности мобильных браузеров. В особо тяжелых случаях возможно даже аварийное завершение. В то же время, если на холсте выполняется анимация, а не отрисовка неподвижных изображений, из-за этого может серьезно расходоваться заряд батареи. Во всех крупных браузерах для работы с холстом поддерживается аппаратное ускорение, благодаря чему отрисовка и отображение протекают значительно быстрее, а энергия расходуется не так быстро.

Прежде чем выбрать одну из этих технологий (либо отказаться от обеих), взвесьте все «за» и «против».

WebGL. Поддержка трехмерной графики (технология WebGL) пока только зарождается и не очень распространена на мобильных устройствах, поскольку вызывает проблемы с производительностью и излишний расход батареи. На момент написания книги поддержка WebGL была наиболее качественно реализована в BlackBerry 10. Сравнительно недавно такая поддержка появилась и в Firefox OS. Конечно, эта технология поддерживается и на более старых мобильных

устройствах, но рекомендую дважды подумать, прежде чем ее задействовать. Дело в том, что из-за активной работы процессора батарея быстро разряжается, а широкое использование JavaScript сильно нагружает процессор. На устройствах с хорошей поддержкой WebDL эта технология обслуживается с помощью графического процессора (GPU), а не основного процессора (CPU). В результате энергия расходуется несколько экономнее, но я все равно нахожу использование WebGL нецелесообразным. Если вы все-таки решитесь задействовать эту технологию, всегда учитывайте вопросы производительности, в частности использование памяти и расход батареи.

Аудио/видео

До появления HTML5 не существовало никакого стандартизированного способа встраивания видео в веб-страницу. Все видеоролики на сайтах приходилось воспроизводить с помощью сторонних плагинов, например Flash или QuickTime. Кроме того, поскольку не существовало удобного способа создания доступных медиа, включаемая таким образом информация оставалась недоступной для людей с ослабленным зрением или слухом.

В HTML5 определяется стандартный способ включения видео и аудио в веб-страницы, для этого используются элементы `<video>` и `<audio>`. И `<video>` и `<audio>` поддерживаются во всех мобильных браузерах (кроме Opera Mini), но набор поддерживаемых аудио- и видеоформатов в разных браузерах различается. Прежде чем мы обсудим, как именно аудио и видео включается в документ, поговорим о кодеках медиафайлов и браузерной поддержке. Дело в том, что в разных браузерах нам придется пользоваться разными типами медиа, а также предоставлять резервный контент для тех браузеров, где выбранный нами тип медиа не поддерживается.

Типы мультимедиа

Когда в браузерах появилась поддержка элементов HTML5 `<video>` и `<audio>`, причем сохранилась поддержка стандартных типов медиа, подобные медиа можно стало воспроизводить и без использования сторонних плагинов. Однако в настоящее время в различных браузерах набор поддерживаемых кодеков для аудио и видео различается. Вы, вероятно, знаете, что на iPhone и iPad не поддерживается технология Flash. Они поддерживают элементы `<video>` и `<audio>`, причем для работы с видео применяется формат H.264, а для аудио — формат AAC (подробнее о них — в следующем разделе). Во всех современных браузерах поддерживается видео HTML5, но для этого применяются разные видеоформаты. В Firefox, Chrome, Android и Opera поддерживается Ogg/Theora (`.ogv`). В IE9, Safari, Chrome, Android и iOS поддерживается MPEG4/H.264 (`.mp4`). В Firefox 4+, Chrome, Opera и Android (2.3+) поддерживается WebM/VP8 (`.webm`). Этот формат поддерживается и в IE9, если в системе установлены соответствующие кодеки (табл. 5.1).

Таблица 5.1. Поддержка видеокодеков в браузерах (в IE9 можно отдельно устанавливать Ogg и WebM)

	iPhone/ iPad	Android	Black-Berry	Opera Mobile	Opera Mini	Win-dows/IE	Chrome Android	Firefox Android
<video>	Да	Да	7	11		9	Да	Да
Ogg		2		11		(9*)		Да
H.264	Да	3.0[1]	7			9	Да	Да*
WebM		2.3		14		(9*)	Да	Да

Существует ряд видеокодеков. Наиболее важными из них являются Theora/Ogg, VP8 и H.264. Theora/Ogg (.ogv) — это открытый стандарт, нативно поддерживаемый в Firefox 3.5, Chrome 4 и Opera 10.5+. В IE для поддержки этого формата требуется установить специальный плагин. Формат WebM, используемый с видеокодеком VP8, — более новый формат, нативно поддерживаемый в самых современных версиях Chrome, Mozilla Firefox и Opera 10.6.

В настоящее время VP8 не требует лицензионных отчислений. Эта технология запатентована, но ее владелец — компания Google — предоставляет бесплатное лицензирование. К сожалению, хотя кодек WebM/VP8 отлично поддерживается в мобильных браузерах, он является предметом патентных споров между Google и Nokia, поэтому в обозримом будущем вряд ли станет общепринятым интернет-стандартом.

Формат H.264 предоставляется в нескольких вариантах — для устройств с узкой, средней и широкой полосой соединения. Он воспроизводится с помощью Adobe Flash, а также на мобильных устройствах, в том числе Android и iPhone, но не является открытым стандартом. Сообщалось, что Chrome планирует отказаться от поддержки данного формата, однако до сих пор этого не произошло. В Firefox поддержка H.264 появилась в 2013 году при условии его установки в операционной системе. В Opera для мобильных устройств складывается аналогичная ситуация.

В данный момент отсутствует формат, который работал бы во всех браузерах, — в этом вы можете убедиться, изучив табл. 5.1. Чтобы ваше видео работало где угодно, его нужно предоставлять в нескольких форматах.

Добавление элемента <video> на сайт

Для максимальной совместимости с различными устройствами необходимо создать две версии видео. Сделайте версию WebM (видео VP8 и аудио Vorbis) и версию MP4 (видео H.264 и аудио AAC). Подключите оба видеофайла с помощью элемента `<video>`, используемого в HTML5, а также его дочерних тегов `<source>`. По умолчанию задайте использование видеоплеера, использующего технологию Flash.

[1] См. http://www.broken-links.com/2010/07/08/making-html5-video-work-on-android-phones/

Атрибуты элементов <video> и <audio>

У элементов <video> и <audio> есть ряд атрибутов, управляющих внешним видом и функционированием встраиваемых медиа.

Элементы <video> и <audio> поддерживают следующие атрибуты:

- src. Атрибут src, обозначающий источник контента, принимает в качестве значения URL аудио- или видеофайла. Его можно заменить множественными дочерними узлами <source>;
- autoplay. При наличии логического атрибута autoplay браузер получает команду сразу же начать воспроизведение видео, а не дожидаться, пока пользователь сам запустит ролик. Его следует использовать только на тех веб-страницах, где видеоконтент является основным;
- loop. При наличии логического атрибута loop видео или аудио будет воспроизводиться циклически до тех пор, пока пользователь не поставит ролик на паузу или не остановит вообще. При наличии такого атрибута видео- или аудиоролик, дойдя до конца, начинает воспроизводиться заново;
- controls. При наличии логического атрибута controls браузер должен отображать элементы для управления медиа (хронометр, кнопка Play, кнопка Pause и т. д.);
- preload. Атрибут preload подсказывает браузеру, какой объем видео следует предзагрузить перед тем, как оно начнет воспроизводиться. Если этот атрибут пропущен или установлен в значение none, то медиаинформация не предзагружается. Если этот атрибут имеет значение metadata, то должна быть получена информация о параметрах видео, его длительности и т. п., но весь медиаобъект загружать не требуется.

Следующие атрибуты применяются только с <video>, но не с <audio>:

- poster. Атрибут poster принимает в качестве значения URL того изображения, которое должно использоваться в качестве визуальной заставки до тех пор, пока не начнется воспроизведение видео. Если этот атрибут отсутствует, видеоплеер будет использовать в качестве заставки первый кадр видео, который обычно выглядит как черный прямоугольник;
- width. Атрибут width принимает в качестве значения ширину элемента-контейнера, в котором воспроизводится видео (в пикселах);
- height. Атрибут height принимает в качестве значения высоту элемента-контейнера, в котором воспроизводится видео (в пикселах).

Вот пример объявления <video> (здесь содержится описание всех компонентов, перечисленных в табл. 5.2):

```
<video autoplay controls loop poster="poster.jpg" preload="metadata"
src="video.mp4" height="360" width="480">Fallback Text</video>
```

Таблица 5.2. Компоненты типичного объявления <video>

Компонент	Описание
<video>	Тег для видео
autoplay	Если этот атрибут установлен, то видео начинает автоматически воспроизводиться сразу же после загрузки
controls	Если этот атрибут установлен, отображается панель управления
loop	Если этот атрибут установлен, видео воспроизводится циклически
poster="/img/poster.jpg"	Если этот атрибут установлен, отображается картинка для предварительного просмотра
preload="metadata"	Может иметь одно из трех значений: none, metadata и auto
src="/video/video.mp4"	Ссылка на видеофайл
height="360"	Высота видео
Fallback Text (Резервный текст)	Здесь может находиться любой HTML-код, как правило ссылка на видео
</video>	Обязательно требуется закрывающий тег </video>

Элементы HTML5 `<audio>` и `<video>` позволяют ассоциировать заголовки со встраиваемыми медиа. Эти элементы входят в состав объектной модели документа HTML5. Соответственно, их можно оформлять с помощью CSS, также для них предоставляется мощный API. С помощью этого API разработчик приобретает контроль над воспроизведением видео, который реализуется посредством множества новых свойств и методов JavaScript, в частности `play()`, `pause()`, `muted` и `ended`.

Когда видео HTML5 будет полностью поддерживаться, причем во всех браузерах станет использоваться один и тот же кодек, код будет прост:

```
<video src="myVideo.mp4" width="400" height="300"
      controls poster="myImage.jpg">
    You don't support HTML5, but you can still
    <a href="myVideo.ogv"> download the video here</a>.
</video>
```

К сожалению, этот код и сегодня не является кросс-браузерным. Как было указано ранее, кодеки у разных браузеров различаются. Для разных браузеров приходится предоставлять разные источники.

Для этой цели в HTML5 имеется элемент `<source>`. С его помощью можно указать более одного источника мультимедиа. У элемента `<source>` есть три атрибута (кроме глобальных): `src`, `type` и `media`.

ПРИМЕЧАНИЕ

Чтобы динамически изменить воспроизводимое медиа, модифицируйте атрибут src у тегов <video> и <audio>. Модификация атрибута src у элемента <source> не поможет. Воспользуйтесь методом canPlayType() для выбора типа, поддерживаемого в браузере.

Атрибут `type` указывает тип медиаресурса. Поэтому прежде, чем загружать ресурс, браузер может определить, понимает ли он медиатип этой информации. Если вы указываете это значение, оно должно соответствовать одному из валидных типов MIME.

Пока не существует универсального кодека, который поддерживался бы во всех браузерах, но код тем не менее не слишком сложен. В примере с игрой мы можем предоставить обучающее видео, демонстрирующее, как в нее играть. Пока мы его не включили, но вполне могли бы это сделать. В таком случае для поддержки нашего видео во всех браузерах потребовалось бы написать следующий код:

```
<video width="400" height="300" preload="none" poster="posterImg.jpg"
      controls>
  <source src="myVideo.mp4" type="video/mp4; codecs=avc1.42E01E,
   mp4a.40.2"/>
  <source src="myVideo.webm" type="video/webm; codecs=vp8, vorbis"/>
  <source src="myVideo.ogg" type="video/ogg; codecs=dirac, speex"/>
  <object width="400" height="324" type="application/x-shockwave-flash"
      data="myVideo.swf"/>
    <param name="movie" value="myVideo.swf"/>
    <param name="flashvars"
      value="image=posterImg.jpg&amp;file=myVideo.mp4"/>
  <!-- резервный вариант -->
  <a href="linktovideo">
    <img src="posterImg.jpg" width="400" height="300"
      alt="Классное видео"/>
  </a>
  </object>
</video>
```

Если ваш браузер поддерживает видео HTML5, оно будет использоваться. Если браузер не поддерживает первый тип медиа, а соответствующий код включен, то он попробует следующий тип. Если не поддерживается видео HTML5, то применяется Adobe Flash. Не поддерживается ни Flash, ни `<video>` — пользователь увидит изображение-заставку. Кроме того, можно включить в код ссылки на скачивание видео.

ПРИМЕЧАНИЕ

В предыдущем коде источник для Flash-файла на 24 пиксела выше, чем другие версии медиа. Дело в том, что элементы управления Flash занимают под видеороликом лишние 24 пиксела, а не накладываются поверх видеоролика, как в HTML5.

Если мы программируем только для современных мобильных устройств, то могли бы обойтись без Flash и добавить треки (см. далее, в пункте об элементе `<track>`):

```
<video width="400" height="300" preload="none" poster=
      "posterImg.jpg" controls>
  <source src="myVideo.mp4" type="video/mp4;
      codecs=avc1.42E01E, mp4a.40.2"/>
  <source src="myVideo.webm" type="video/webm;
      codecs=vp8, vorbis"/>
  <source src="myVideo.ogg" type="video/ogg;
      codecs=theora, vorbis"/>
    <img src="posterImg.jpg" width="400" height="300"
```

```
        alt="Заголовок видео" title="Your browser does
        not support video"/>
  <track kind="subtitles" label="English" src="en.vtt"
      srclang="en" default></track>
  <track kind="subtitles" label="Deutsche" src="de.vtt"
      srclang="de"></track>
</video>
```

Как правило, в видеофайлах присутствуют как видео-, так и аудиодорожки (треки). На аудиотреках находятся маркеры, помогающие синхронизировать аудио и видео. У отдельных треков могут быть метаданные — например, соотношение сторон для области просмотра видео либо язык аудио. У контейнеров также могут быть метаданные, в частности заголовок самого видео, обложка альбома, номер серии и т. д.

Аналогичным образом в документ можно добавить элемент `<audio>`:

```
<audio id="sound">
  <source src="music.mp3" type="audio/mp3"/>
  <source src="music.ogg" type="audio/ogg"/>
    <!-- flash-версия аудио для браузеров, в которых не поддерживается такой тег -->
</audio>
```

На сайте Dev.Opera есть подробная статья об обнаружении поддержки.

Элемент <track>. Чтобы обеспечить доступность аудио- и видеофайлов для пользователей с ослабленным слухом, а также для людей, не владеющих данным языком, можно добавлять к видео соответствующие надписи. Они заключаются в элементах `<track>`, которые, в свою очередь, содержат ссылки на файлы с субтитрами.

Когда элемент `<track>` включается в код как потомок элемента `<video>` или `<audio>`, атрибут `src` элемента `<track>` ссылается на хронометрированный трек или данные, связанные с временем. Атрибут `kind` сообщает, на данные какого рода указывает атрибут `src`. К числу значений атрибута `kind` относятся `subtitles`, `captions`, `descriptions`, `chapters` и `metadata`.

Можно задать множество элементов-треков в качестве потомков медиаэлемента, но все они должны быть уникальными комбинациями, состоящими из информации о роде элемента и его языке:

- `subtitles`. Это значение атрибута `kind` задается по умолчанию. Указывает, на какой язык переводится диалог, и по умолчанию накладывается на видео или аудио. Это значение наиболее удобно, когда речь героев сложно расслышать либо они разговаривают на иностранном языке;
- `captions`. Указывает файл трека, в котором размещается транскрипция или перевод диалога. Напоминает субтитры, но включает также музыкальные эффекты, звуковые подсказки и другую аудиоинформацию, которая может полностью заменить звуковое сопровождение, если оно отсутствует. Значение наиболее полезно в случаях, когда видео работает с отключенным звуком или пользователь плохо слышит;

- descriptions. Треки — это описания видеосоставляющей медиаресурса, предназначенные для синтеза аудио в тех случаях, когда видео недоступно. Данное значение полезно для слабовидящих пользователей и для тех, кто по каким-то другим причинам не может просматривать видео или читать субтитры;
- chapters. Указывает трек, в котором записаны заголовки глав. Используется для навигации по медиафайлу;
- metadata. Указывает трек, который должен использоваться сценариями, а не отображаться для пользователя-человека.

Задайте источник трека, сопроводив его необходимым атрибутом src. В атрибуте srclang указывается язык, на котором записан текст в элементе <track>. В атрибуте label находится пригодный для прочтения человеком заголовок <track>. Браузер использует значение атрибута label для вывода в пользовательском интерфейсе субтитров, заголовка и описания аудио.

Если присутствует атрибут default, то соответствующий элемент <track> должен быть активизирован, если только в пользовательских настройках не выбран другой элемент <track>. В медиакоде может быть только один элемент <track>, который таким образом задается по умолчанию.

Видео, аудио и JavaScript

Если вы собираетесь использовать код JavaScript для управления элементами <audio> и <video>, то потребуется применить обнаружение возможностей (feature detection) для обеспечения необходимой поддержки и избавления от ошибок JavaScript:

```
if (createElement('audio').canPlayType) { /* аудио поддерживается */}
```

Можно использовать нативные элементы управления либо создавать собственные. Элементы <audio> и <video> поддерживают методы play() и pause(). Для создания собственных элементов можно написать код HTML для их отображения на экране и JavaScript для воспроизведения и приостановки видео. Для этого применяется примерно такой код:

```
<div id="controls" style="display: none">
  <button id="playButton">Play</button>
  <button id="pauseButton">Pause</button>
</div>
<script>
  if (document.createElement('audio').canPlayType) {
    if (document.createElement('audio').canPlayType('audio/mp3') ||
      (document.createElement('audio').canPlayType('audio/ogg')) {
      // HTML5 <audio> и включенный тип аудио поддерживаются
      document.getElementById('player').style.display = 'block';
    } else {
    ... Включите здесь flash или другое аудио ...
  }
</script>
```

Для создания собственных элементов управления можно воспользоваться следующим кодом:

```
var videoClip = document.querySelector('#clip');
var playButton = document.querySelector('#playButton');
var pauseButton = document.querySelector('#pauseButton');

playButton.addEventListener('touchEnd', function() {
  playVideo();
});
pauseButton.addEventListener('touchEnd', function() {
  pauseVideo();
});

function playVideo() {
    // воспроизводим видео
    videoClip.play();
    // обновляем элементы управления
    playButton.disabled = true;
    pauseButton.disabled = false;
}

function pauseVideo() {
    // ставим видео на паузу
    videoClip.pause();
    // обновляем элементы управления
    playButton.disabled = false;
    pauseButton.disabled = true;
}

function MuteUnMute() {
    // изменяем значение кнопки
    document.getElementById('mute').value = videoClip.muted ? 'Mute' :
    'Unmute';
    // изменяем состояние видео
    videoClip.muted = videoClip.muted ? false : true;
}
```

CubeeDoo. В нашей учебной игре будет несколько звуковых эффектов. Кроме необязательной фоновой музыки, которую пользователь при желании может отключить, мы будем использовать в игре звуки, означающие успешное или неуспешное выполнение некоторых операций, в частности переход на следующий уровень, совпадение элементов, несовпадение элементов и т. д.

Если работает фоновое звуковое сопровождение, то оно будет работать с самим тегом <audio>, поскольку пользователь может управлять такой музыкой. Звуки-отклики зависят от пользовательских действий и их успешности, поэтому я динамически генерирую эти звуки с помощью JavaScript.

Далее приведены два метода, которыми можно пользоваться при работе со звуком. Так, можно включить аудио непосредственно в наш HTML:

```
<audio id="nonmatchsound" preload src="notmatch.mp3"></audio>
<audio id="matchsound" preload src="match.mp3"></audio>
```

Мы предзагрузили аудио, однако не выполняем ни автоматического, ни циклического воспроизведения аудиофайлов. Вместо этого генерируем звук совпадения или несовпадения элементов с помощью JavaScript:

```
playSound: function(matched) {
      if (qbdoo.mute) {
        return false;
      }
      if (matched) {
        qbdoo.matchfound.play();
      } else {
        qbdoo.failedmatch.play();
      }
    },
```

Правда, можно и не вставлять код аудио прямо в HTML. Вместо этого аудио добавляется в объектную модель документа с помощью JavaScript, и прикреплять аудиофайлы к таблице не требуется:

```
playSound: function(matched) {
      // если в игре отключено звуковое сопровождение – пропускаем
      if (qbdoo.mute) {
        return false;
      }
      // если мы еще не создали аудиоузел – создаем его
      if (!qbdoo.audio) {
        qbdoo.audio = document.createElement('audio')
      }
      if (matched) {
        qbdoo.audio.src = qbdoo.matchedSound;
      }
      else {
        qbdoo.audio.src = qbdoo.failedMatchSound;
      }
        qbdoo.audio.play();
    },
```

Мы включили аудио лишь для того, чтобы продемонстрировать, как используется <audio>. Никогда не задавайте автоматическое воспроизведение музыки — это порочная практика. Как вы могли заметить, в игре есть кнопка для отключения звука. Если в игре есть звуковое оформление и по умолчанию оно включено, то учтите это при работе с LocalStorage (подробнее об этом — в главе 6).

Оформление видео

Элемент <video> принципиально ничем не отличается от других HTML-элементов. Это значит, что его можно оформлять. Можно определять высоту и ширину окна

видео с помощью CSS. Можно замаскировать видео, скруглить его углы и даже отразить содержимое. Работая с холстом, можно сэмплировать пикселы и инвертировать их (правда, это можно делать и с помощью CSS-фильтров).

Адаптивный подбор размера видео

Гораздо важнее то, что вам может потребоваться изменять размер видео в зависимости от диагонали экрана устройства и соотношения его сторон. Терри Кобленц предложил эффективный метод, позволяющий браузеру определять размеры окна видео по ширине объемлющего блока (или ширине всего окна). При этом в качестве ориентира берутся внутренние размеры (intrinsic dimensions). При изменении ширины (например, когда изменяется ориентация экрана) высота пересчитывается. Размер видео корректируется — то есть оно масштабируется точно так же, как масштабировалось бы изображение.

Для создания адаптивного видео создается специальная подвижная рамка с нужным соотношением сторон (4:3, 16:9 и т. д.). Затем видео растягивается по ширине этой рамки, заполняя ее целиком. При этом используются внутренние отступы, процентные соотношения и абсолютное размещение. Как правило, для внутренних отступов задается величина 56,25 или 75% актуальной ширины в зависимости от соотношения сторон. Поскольку мы можем воспользоваться сильными сторонами рамочной модели, элемент <video> занимает область с внутренними отступами по всей высоте и ширине.

Если вам требуется поместить видео с изменяемыми размерами области просмотра на сайте с адаптивным дизайном, можно поступить так:

```
.wrapper {
  position: relative;
  height: 0;
  width: 100%;
  padding-bottom: 56.25%;
  / * или */
  padding-bottom: 75%;
}
video {
  position: absolute;
  width: 100%;
  height: 100%;
  left: 0;
  top: 0;
}
```

Что необходимо знать о реализации <video>

Технология Flash используется в виде плагина, который поддерживается одной компанией и поэтому повсюду работает по схожим принципам. Но при работе с <video> в разных браузерах и операционных системах возникают различные причуды. Так, на iPhone, Android и Windows Phone 8 все видеоролики воспроизводятся в полноэкранном режиме. На iPad среди элементов управления есть кнопка

Fullscreen (Полный экран), она реагирует даже на жест щипка. В iOS и Android для работы с видео используется графический процессор (GPU), но в Android до версии 4 видео воспроизводилось вне CPU.

- Убедитесь, что сервер поддерживает mime-типы видео, так как в противном случае в Firefox может произойти ошибка. Добавьте в файл `.htaccess` информацию вида `AddType video/ogg .ogv`, если видео еще не поддерживается.
- На iPhone и iPad автоматическое воспроизведение (`autoplay`) включаться не будет, даже если такой атрибут имеется.
- Элементы управления выглядят нативно, как и другие элементы управления в данном браузере. Как было указано ранее, их внешний вид и функционал можно изменить с помощью JavaScript. Подробнее об оформлении элементов управления можно узнать на сайте http://videojs.com/.
- Если вы хотите приступить к добавлению собственных видеороликов, обратите внимание на свободно распространяемый по лицензии GPL мультиплатформенный многопоточный видеотранскодер Handbrake, доступный для операционных систем Mac OS X, Linux и Windows.

Напомню, что при воспроизведении видео и аудио сильно расходуется заряд батареи. Хотя на смартфонах поддерживаются обе эти возможности, программист должен пользоваться ими осторожно. Необходимо гарантировать, что ваше веб-приложение не будет чрезмерно истощать батарею пользовательского устройства.

6 Другие API HTML5

Веб-приложения, работающие в режиме офлайн

До недавнего времени пользователи веб-приложений могли работать с ними лишь при наличии соединения с Интернетом. В офлайновом режиме веб-почта, календари и другие подобные инструменты оставались недоступными, и такая ситуация с большинством подобных приложений сохраняется до сих пор.

Работая офлайн, пользователи все-таки могут обращаться к некоторым страницам тех сайтов, которые они посещали. Эти страницы находятся в браузерном кэше, но его объем ограничен, а самим кэшем не так просто управлять. Если пользователя выбрасывает в офлайн непосредственно во время важного процесса — например, написания электронного сообщения или заполнения формы, — то, нажав кнопку Submit (Отправить), он может потерять все введенные данные.

В спецификации HTML5 описано несколько решений этой проблемы, в частности локальное и сеансовое сохранение данных на устройстве, а также работа с офлайновым HTTP-кэшем приложений. Благодаря такой технологии приложение остается доступным даже в офлайновом режиме. В HTML5 есть ряд возможностей, позволяющих справляться именно с такими проблемами и создавать приложения, которые не теряют всех функций, оказываясь в офлайне. К числу таких решений относятся indexDB, API для офлайнового кэширования приложений, события соединения, статусы, а также API `LocalStorage` и `SessionStorage`.

У меня вообще есть соединение с Интернетом?

При реализации офлайновых возможностей нам, в частности, потребуется знать, когда устройство подключено и когда не подключено к Интернету. HTML5 определяет свойство `onLine` объекта Навигатор (Navigator). По значению этого свойства можно узнать, работает ли пользователь онлайн в данный момент:

```
var isOnline = navigator.onLine;
```

Этот код вернет `true` или `false` («истина» или «ложь»). Правда, если вы получили `true`, это может означать, что пользователь работает в интранете, а не в Интернете.

Кэш приложений

Если вы хотите создавать игры для Интернета, которые могли бы составить конкуренцию нативным играм с мобильных устройств, то необходимо гарантировать, что пользователь сможет играть в вашу игру, даже будучи в офлайне. Мы хотим, чтобы любители игры CubeeDoo могли развлекаться ею и дома, используя Wi-Fi, и на природе — некогда любоваться пейзажами, если под рукой такая классная игра, — и даже во время перелета через Тихий океан. *Кэш приложений* позволяет создавать такие веб-приложения, которые будут доступны и в режиме офлайн.

Ранее браузеры для ПК могли сохранять HTML-файл и ассоциированные с ним медиа лишь в папке на локальном компьютере. Этот метод работает со статическим контентом, но не предусматривает возможности обновления, а это обычно очень негативно воспринимается пользователями.

С повсеместным распространением веб-приложений как никогда важно стало обеспечивать доступность таких приложений в офлайновом режиме. Браузеры уже давно умеют кэшировать компоненты веб-сайта, но HTML5 позволяет справиться с некоторыми сложностями этого процесса, предоставляя API кэша приложений (AppCache).

Кэш приложений позволяет указывать, какие файлы должны кэшироваться и оставаться доступными в режиме офлайн. Таким образом, при отсутствии подключения к Интернету сайт будет работать правильно — в частности, нормально перезагружать страницы. При использовании кэша приложений ваша программа приобретает следующие преимущества:

- офлайновый просмотр страниц;
- ускорение перезагрузки страниц;
- снижение нагрузки на сервер.

При офлайновом просмотре страниц с применением кэша приложений пользователю будет доступен весь сайт.

В большинстве мобильных браузеров кэш приложений может хранить на устройстве до 5 Мбайт информации (или всего 5 Мбайт — это как посмотреть). У различных браузеров этот объем может варьироваться. В то время как пользователь может корректировать данную величину в настройках браузера, код нужно писать в расчете на значения, задаваемые по умолчанию, если только все ваши пользователи не являются достаточно опытными.

Чтобы кэш приложений функционировал, необходимо сопроводить открывающий тег `<html>` атрибутом `manifest`. Значение этого атрибута представляет собой URL листинга текстового файла, чьи ресурсы должны кэшироваться. Укажите в вашем HTML-файле `manifest="URL_of_manifest"`:

```
<!doctype HTML>
<html manifest="cubeedoo.appcache">
<meta charset="utf-8"/>
<title>...
```

Если элемент <html> сопровождается атрибутом manifest, который ссылается на валидный файл описания (манифеста), то, когда пользователь загрузит страницу, браузер кэширует файлы, перечисленные в файле описания, сам файл описания, а также загруженный документ. Вся эта информация будет доступна и в офлайновом режиме. Но, хотя актуальный документ и кэшируется по умолчанию, его лучше также указать в файле описания среди информации, предназначенной для кэширования.

ПРИМЕЧАНИЕ

Документ, ссылающийся на файл описания, кэшируется по умолчанию.

Как же работает этот механизм? Когда браузер встречает атрибут manifest, он скачивает файл описания и пытается кэшировать те файлы, которые в нем перечислены. Если пользователь, работая в офлайновом режиме, открывает сайт, сохраненный на локальном устройстве, то при этом используются уже кэшированные файлы. В онлайновом режиме браузер сначала обращается к кэшированному сайту и лишь потом проверяет, доступны ли обновления этой информации и, соответственно, требуется ли их кэшировать.

Если в кэшированный файл описания внесены изменения, то браузер прочитает весь кэш, находящийся в памяти, перед тем как вносить в него изменения. Браузер ищет изменения именно в файле описания, а не во всех других файлах, находящихся на сервере. Это делается, чтобы определить, должен ли кэш быть обновлен. Иными словами, чтобы кэш обновился, необходимо внести изменения в файл описания. Обновления всех остальных ресурсов будет недостаточно. Учтите это в дальнейшем, при изучении раздела «Обновление кэша».

Загрузив весь сайт из кэша, браузер получает файл описания с сервера. Если файл описания изменился с момента последнего посещения страницы, то браузер повторно скачивает все ресурсы и повторно кэширует их. Если браузеру не удается повторно скачать все ресурсы, он продолжает использовать старый кэш. Однако если все необходимые файлы удалось скачать успешно, то в ходе данного сеанса все равно продолжает использоваться старый кэш. Новый кэш начинает применяться лишь после того, как пользователь в следующий раз зайдет на сайт.

Файл описания кэша

Файл .appcache — текстовый. В нем перечисляются те ресурсы, которые браузеру необходимо кэшировать для обеспечения офлайнового доступа к вашему приложению. Файл описания должен начинаться со строки CACHE MANIFEST. За этой обязательной строкой следуют список файлов, которые должны быть кэшированы, опциональные комментарии и заголовки разделов.

Файл .appcache должен относиться к MIME-типу text/cache-manifest. Добавьте код:

```
AddType text/cache-manifest .appcache
```

в файле .htaccess или в файле конфигурации сервера. Обычно это требование является обязательным. Файл описания постоянно хранится в кэше браузера. .appcache может иметь примерно следующий вид:

```
CACHE MANIFEST
#version01

# явно кэшируемые файлы
CACHE:
index.html
css/styles.css
scripts/application.js
# ресурсы, требующие сетевого соединения
NETWORK:
signin.php
dosomething.cgi

FALLBACK:
/ 404.html
```

Обратите внимание: файлы, перечисленные в файле описания кэша, указаны относительно этого файла описания.

Для создания комментария начните строку с символа #. В таком случае программа игнорирует эту строку.

Заголовки разделов позволяют полнее управлять тем, как кэш приложений будет интерпретировать ваши веб-файлы и ресурсы. Раздел может иметь один из четырех следующих заголовков: `CACHE`, `FALLBACK`, `SETTINGS` и `NETWORK`. За каждым из них следует двоеточие.

Файлы, идущие за заголовком `CACHE`, кэшируются явно. Если никакой заголовок не определен либо файлы перечислены до заголовка, то они кэшируются так, как если бы сопровождались заголовком `CACHE`. Обратите внимание: файлы, передаваемые по защищенному соединению (HTTPS), могут кэшироваться лишь в случае, если они берутся из того же источника, что и файл описания.

Тот файл, в элементе `<html>` которого содержится файл описания, всегда добавляется в кэш независимо от того, указан ли он под заголовком `CACHE`.

Не включайте сюда сам файл описания, иначе сайт может никогда не обновиться.

Файлы, идущие за заголовком `NETWORK`, явно *не* кэшируются. Соответственно, они доступны лишь при работе онлайн.

Файлы, следующие за заголовком `FALLBACK`, предоставляются попарно: те файлы, которые должны отображаться, и их резервные варианты — на случай, если основной файл отобразить не удастся. Если первый файл из пары недоступен, то будет использоваться второй. При включении заголовка `SETTINGS` он должен идти последним и включать в себя однострочное значение `prefer-online`.

ВНИМАНИЕ

Не указывайте cache.appcache в числе файлов для кэширования (то есть в файле описания), так как при его наличии ваш сайт, возможно, никогда не обновится.

Обновление кэша

Браузерный кэш не обновляется и не перезаписывается до тех пор, пока в файл описания не будут внесены изменения или пока не будут применены методы

JavaScript applicationCache. Недостаточно внести изменения в один из файлов, перечисленных в файле описания, — например, на языке JavaScript, HTML или CSS. Каждое такое изменение должно быть отражено в самом файле описания.

Как правило, в файл описания добавляют комментарий, чтобы принудительно выполнить обновление файла. Так, если в предыдущем фрагменте кода заменить комментарий #version01 на #version02, то браузер будет проинформирован о том, что кэш требуется обновить. Если использовать вместо номера версии отметку времени, то такая информация будет более интуитивно понятной. Обратите внимание: неважно, какую именно информацию вы внесете в комментарий, — важны сам факт внесения изменения и то, что эта операция будет зафиксирована браузером.

В целом, номер версии для нашего кэша — это и есть комментарий во второй строке. Когда пользователь обращается к этой веб-странице, браузер сначала загружает сайт из кэша — при условии наличия такого кэша. Затем с сервера скачивается файл .appcache и сравнивается с файлом .appcache, находящимся в памяти. Если в файле описания обнаруживаются изменения — например, изменился номер версии, — то будет скачана и остальная часть кэша. Именно поэтому мы и добавляем комментарий. Изменить его гораздо проще, чем, например, имя файла (и всех файлов, которые на него ссылаются). Изменение комментария с указанием номера версии или отметкой времени уже стало стандартным способом информирования браузера о том, что файл описания следует считать обновленным.

Как только приложение оказывается в офлайне, оно остается в кэше до тех пор, пока пользователь сам не удалит в браузере данные, касающиеся вашего сайта, либо файл .appcache не будет изменен, либо кэш приложений не будет изменен программно.

Сначала браузер загружает сайт из кэша. Только после этого он проверяет, внесены ли какие-либо изменения в файл описания. Как только обнаруживается изменение, скачиваются все файлы, перечисленные в манифесте. Если сам файл описания или указанный в нем ресурс загрузить не удается, то обрывается весь процесс обновления кэша. Соответственно, вы рискуете получить частично обновленную версию веб-приложения.

Можно принудительно обновить кэш, не изменяя файл описания программно. Для того чтобы явно обновить кэш, вызовите метод applicationCache.update(). Когда получите статус «готово» (UPDATEREADY), замените старый кэш новым:

```
var appCache = window.applicationCache;

if (appCache.status == appCache.UPDATEREADY) {
  appCache.swapCache();
}
```

Статус может иметь одно из следующих значений: UNCACHED, IDLE, CHECKING, DOWNLOADING, UPDATEREADY и OBSOLETE. Если не удается загрузить сам файл описания или какой-либо указанный в нем ресурс, то процесс обновления кэша обрывается и браузер продолжает работать со старым кэшем приложений.

Хотя такие шаги и обеспечивают обновление кэша, не обновляется та информация, которую пользователь просматривает в настоящий момент. Пользователь продолжит работать с той версией сайта, которая была кэширована ранее, до тех пор, пока в следующий раз не обратится к веб-приложению. Таким образом, чтобы получить новый контент, пользователю фактически требуется дважды скачать с вашего сайта нужную информацию.

Если вы хотите обязательно предоставить пользователю новый контент, то можете запрограммировать перезагрузку сайта при срабатывании обработчика события `updateready`. Однако перезагрузка может вклиниться прямо посреди действий, выполняемых пользователем на сайте. Пользователи воспринимают это негативно, поэтому тщательно обдумывайте ситуацию перед реализацией данной функции.

В контексте CubeeDoo мы можем указать все нужные файлы в файле описания. Если бы у нас была форма для входа с паролем, то как раз ее мы бы здесь пропустили. Кроме того, потребуется включить комментарий с номером версии (или датой, то есть данными, которые для вас информативны). Как только мы будем вносить изменения в какие-либо файлы, перечисленные под заголовками `CACHE:` или `FALLBACK:`, одновременно с этим будем менять номер версии:

```
CACHE MANIFEST
#version01

CACHE:
index.html
css/cubeedoo.css
scripts/cubeedoo.js
assets/matched.mp3
assets/notmatched.mp3
images/shapes.svg

NETWORK:
login.html

FALLBACK:
/ 404.html
```

Когда мы будем готовы развернуть наше приложение, добавим атрибут `manifest` к тегу `<html>` страницы индекса. *Но сейчас этого делать еще не надо.* Ничто не раздражает так сильно, как разработка и тестирование веб-приложения, полностью кэшированного в браузере:

```
<html lang="en-us" manifest="cubeedoo.appcache">
```

Кроме того, если браузеру не удастся найти кэш-файл описания на сервере, он просто очистит кэш. Если браузер встретит ссылку на несуществующий файл и получит отклик с кодом 404 («Не найдено»), он очистит кэш.

Локальное и сеансовое хранение данных

С помощью кэша приложений мы можем сохранять наши веб-приложения на устройствах, чтобы они были доступны и в офлайновом режиме. В кэше приложений вы можете хранить файлы, но иногда приходится хранить и данные. Например, когда пользователь CubeeDoo находится в офлайне (и в онлайне), мы хотим вести таблицу рекордов, в которой будут указаны результат каждого из лидеров, его имя и затраченное им время. Кроме того, когда пользователь ставит игру на паузу, на этом же экране отображается текущее состояние игры. Для реализации этих возможностей мы можем воспользоваться `LocalStorage`, IndexedDB или базой данных Web SQL, которая уже считается нежелательной.

Мы собираемся применить `LocalStorage` для постановки игры на паузу и использовать базу данных Web SQL для сохранения рекордов (она хоть и считается нежелательной, но по-прежнему широко распространена). Можно было бы сделать и наоборот. Однако нам совершенно не подойдет база данных IndexedDB, поскольку она пока не поддерживается на iOS и Android (однако поддержка этой базы данных появилась в последних релизах IE10, Blink и Firefox).

`LocalStorage` и `SessionStorage` — это удобные в использовании хранилища ключей/значений. Вы могли бы спросить: «Но есть же cookie, зачем вся эта катавасия?» Дело в том, что у cookie есть ряд недостатков, которые легко устраняются с помощью `LocalStorage`.

Сравнение cookie

Cookie применяются в основном для управления сеансами, персонализацией и отслеживанием. Серверные cookie — это строки (последовательности символов), отсылаемые с веб-сервера в браузер и обратно вместе с каждым запросом и откликом HTTP. Браузер может вернуть на сервер неизмененный cookie. В таком случае HTTP-транзакция приобретает состояние, тогда как при любых других условиях состояние в ней не сохраняется. Клиентские cookie — это последовательности символов, генерируемые с помощью JavaScript. Их можно использовать для включения состояния, передачи информации на сервер и получения ее обратно, а также обычного ведения значений на клиенте — например, перечисления элементов в корзине заказов.

Браузер может хранить 300–400 cookie, причем максимальный размер cookie может составлять 4 Кбайт, а каждому серверу или домену могут соответствовать не более 20 cookie. С каждым HTTP-запросом отсылаются сразу все cookie. Хотя автоматическая отправка информации и может быть вам полезна, она неудобна при работе с мобильными устройствами, так как требует значительного расширения полосы передачи данных. Кроме того, курсирование cookie может негативно сказаться на безопасности.

Хотя максимальный объем cookie на любом домене не может превышать 80 Кбайт (не более 20 cookie-файлов, максимальный размер файла — 4 Кбайт), новые стандарты локального и сеансового хранения информации позволяют использовать и больший объем. Размер зависит от браузера, но, как правило, речь идет о мегабайтах, а не о килобайтах.

LocalStorage применяется для долговременного хранения больших объемов данных для конкретного домена в рамках одного браузера. Данные LocalStorage сохраняются и после того, как закрывается окно браузера или весь браузер. Данные LocalStorage доступны сразу во всех браузерных окнах.

Данные SessionStorage доступны лишь в том окне браузера, в котором они были созданы, и удаляются с завершением сеанса. Информация SessionStorage доступна для любой страницы, открытой в конкретном окне, при условии, что все такие страницы относятся к одному и тому же источнику. Если открыть окно браузера и переходить в нем со страницы на страницу одного и того же сайта, то каждая страница, на которую вы попадете таким образом, будет иметь доступ к общему SessionStorage. Если у пользователя открыто сразу много окон — например, он просматривает ваш сайт одновременно в трех разных окнах браузера, — то каждое такое окно будет использовать собственную отдельную копию SessionStorage, но все пары «ключ/значение» из LocalStorage будут использоваться совместно.

При каждом HTTP-запросе на сервер отсылаются как долговременные, так и сеансовые cookie. Если вам требуется отправить информацию на сервер, то cookie, пожалуй, будут верным решением. Однако до появления HTML5 cookie использовались и для сохранения состояния. В результате с сервера и на сервер передавалась масса бесполезной информации, впустую расходовалась полоса соединения. При использовании LocalStorage и SessionStorage мы экономим эту полосу.

В табл. 6.1 описаны пять методов и одно свойство. Ими обладают как SessionStorage, так и LocalStorage.

Таблица 6.1. Методы и свойство SessionStorage и LocalStorage

Метод или свойство	Описание
setItem(key, value)	Устанавливает значение заданного ключа. Например, так определяется сеансовая переменная: SessionStorage.setItem('keyname', 'data value') LocalStorage.setItem('keyname', 'data value')
getItem(key)	Получает значение заданного ключа. Возвращает null, если ключ не существует: SessionStorage.getItem('keyname') SessionStorage.keyname LocalStorage.getItem('keyname') LocalStorage.keyname
removeItem(key)	Удаляет ключ и ассоциированное с ним значение. Следующий код сбрасывает ранее установленное значение: SessionStorage.removeItem('keyname') LocalStorage.removeItem('keyname')
clear()	Удаляет все пары «ключ/значение». Очистка этой информации выполняется с помощью кода: SessionStorage.clear() LocalStorage.clear()

Продолжение ⇗

Таблица 6.1 *(продолжение)*

Метод или свойство	Описание
key(position)	Возвращает ключ для значения, имеющего заданную числовую позицию: SessionStorage.key(position) LocalStorage.key(position)
length	Свойство length доступно только для чтения. Указывает, сколько пар «ключ/значение» в настоящее время хранится в SessionStorage: SessionStorage.length LocalStorage.length

Работать с `SessionStorage` и `LocalStorage` исключительно просто. Мы обращаемся с ними как с обычными объектами, имеющими заданное имя: `SessionStorage` и `LocalStorage` соответственно.

Ведутся дискуссии о том, какой из этих API хранения данных работает быстрее. Конечно, обращение к жесткому диску для считывания данных требует больше времени, чем обращение к значению JSON в браузере. Однако на мобильном устройстве обращение к жесткому диску — более эффективная операция, нежели отправка HTTP-запроса.

Использование LocalStorage для повышения производительности мобильных устройств

На некоторых сайтах, например http://m.bing.com/, `LocalStorage` применяется для уменьшения количества HTTP-запросов, выполняемых при загрузке страницы. Ранее в пункте «Элемент <style> и производительность мобильных устройств: антипаттерн» (подраздел «<style>», глава 2) было написано, что при первом обращении к серверу HTTP-запрос содержит сценарии и стили, которые затем JavaScript и CSS извлекают и разносят в разные пары «ключ/значение» в `LocalStorage`. Каждый сценарий имеет уникальный идентификатор в виде названия пары «имя/значение». Эта информация хранится в cookie.

Когда пользователь делает запрос к новой странице, в этом запросе отправляется cookie-файл, сообщающий серверу, какие файлы уже сохранены в пользовательском браузере. Затем сервер отсылает только остальные файлы, которые еще нужны. Таким образом, при обращении к странице удается ограничиться единственным HTTP-запросом.

Хотя при первом запросе размер файла может быть довольно велик, последующие запросы очень малы, поскольку большинство необходимых ресурсов уже хранится локально в `LocalStorage`. Помещение сценариев и стилей на страницу является антипаттерном с точки зрения производительности и стандартов, но такая техника положительно влияет на производительность некоторых мобильных сайтов и приложений[1].

[1] Подробнее о sessionStorage рассказано на сайте http://www.nczonline.net/blog/2009/07/21/introduction-to-sessionstorage/

Сохраняемость данных и пользовательских настроек не только положительно сказывается на пользовательском восприятии, но и помогает в работе тем пользователям, чьи данные могут быть несогласованными, а также при сбоях Wi-Fi, перегруженности ближайших сотовых вышек, отсутствии данных, а также в случаях, когда пользователь стремится ограничить использование данных.

Поскольку кэш приложений — не та панацея, на которую многие, возможно, надеялись, разработчики сформулировали собственные рекомендации по созданию офлайнового хранилища данных. Как правило, организуется смешанное использование кэша приложений и `LocalStorage`. В Financial Times есть хорошая статья об их работе, обоснованиях и коде.

CubeeDoo

В игре CubeeDoo мы используем `LocalStorage` для сохранения состояния на тот период, пока игра стоит на паузе. С помощью `SessionStorage` сохраняем имя пользователя и значения, задаваемые в игре по умолчанию. Можно применять `SessionStorage` и `LocalStorage` для сохранения всех трех фрагментов информации либо для одновременного сохранения любых двух составляющих этой тройки. Я решила в качестве примера воспользоваться здесь обоими хранилищами.

В данном случае мы прибегаем к API хранения данных и таким образом снижаем необходимость в сохранении состояния на сервере. На самом деле серверного интерфейса базы данных в CubeeDoo просто нет. Нашему серверу требуется лишь сохранять и предоставлять статические файлы. Все возможности, которые обычно реализуются в базе данных и находятся в облаке, например таблица рекордов, в данном случае запрограммированы на пользовательском устройстве.

`LocalStorage` используется для сохранения состояния в тот период, когда игра стоит на паузе. Когда пользователь приостанавливает игру таким образом, мы применяем специальные атрибуты данных и API множества данных для установки и получения значений и местоположений каждой из карт. Значения карт сохраняются в `LocalStorage` вместе со всеми остальными релевантными данными, в частности оставшимся временем, актуальным уровнем, текущим количеством очков и т. д. Вся эта информация требуется для возобновления игры с того момента, в который она была поставлена на паузу. Если бы мы использовали `SessionStorage`, то с постановкой игры на паузу также не возникало бы никаких проблем. Но стоит пользователю закрыть окно браузера — и пары «ключ/значение» `SessionStorage` удаляются, поскольку завершается сеанс браузера.

`SessionStorage` применяется для временного сохранения имени пользователя. Смысл использования `SessionStorage` вместо `LocalStorage` для хранения имени пользователя заключается в том, что отдельное имя пользователя может сохраняться на соседней вкладке того же браузера. Недостаток (или достоинство) заключается в том, что когда пользователь закрывает браузер, то имя пользователя, использовавшееся в списке рекордов, также удаляется.

Мы сохранили исходное состояние игры — значения, задаваемые по умолчанию, — с помощью `SessionStorage`. `SessionStorage` применяется для хранения стандартных настроек игры, которые начинают действовать сразу после ее загрузки.

Когда пользователь проходит игру уровень за уровнем, мы не обновляем страницу, а извлекаем оригинальные значения из SessionStorage. Таким образом, чтобы начать новую игру, не требуется перезагружать страницу для доступа к настройкам игры, которые вполне могли измениться с момента последнего запуска. Мы могли бы сохранить эти переменные как свойства глобального объекта, используемого в сценарии, но при перезагрузке все значения будут сброшены к заданным по умолчанию — тем, которые мы запрограммировали на JavaScript.

Я решила сохранить эти задаваемые по умолчанию значения в SessionStorage просто потому, что HTML5 позволяет мне это сделать! Я применила SessionStorage, а не localStoragr, поэтому данные значения не сохраняют состояния между сеансами.

Я включила в код (среди прочих) следующие функции:

- storeValue — сохраняет игровые значения, задаваемые по умолчанию;
- alterValue — обновляет сохраненные значения, задаваемые по умолчанию;
- pauseGame — ставит игру на паузу, сохраняет актуальное состояние в LocalStorage;
- playGame — возвращает игру в состояние, которое имело место до постановки на паузу. Все карты возвращаются на места, таймер перезапускается;
- reset — очищает хранилище LocalStorage от значений, сохраненных там, когда игра была на паузе. Соответственно, удаляется сохраненное состояние игры.

Обратите внимание: qbdoo — это пространство имен верхнего уровня для игры CubeeDoo. В нем есть несколько свойств, которыми вы можете управлять:

```
1  var qbdoo = {
2    // игровые настройки
3    currentLevel: 1,
4    currentTheme: "numbers",
5    gameDuration: 120,
6    score: 0,
7    matchedSound: 'assets/match.mp3',
8    failedMatchSound: 'assets/notmatch.mp3',
9    mute: true,
10    cardCount: 16,
11    iterations: 0,
12    iterationsPerLevel: 2,
13    possibleLevels: 3,
14    maxHighScores: 5, ...
```

Можно задать значения, которые используются по умолчанию, например, количество карт, количество раздач на каждом уровне, исходную длительность раунда и т. д. Вы как разработчик можете изменять любые из этих стандартных значений. В процессе игры некоторые из них также изменяются. Мы сохраняем исходные значения и восстанавливаем их по мере того, как они изменяются в SessionStorage.

Функция storeValues() сохраняет исходные игровые значения:

```
1    storeValues: function(newgame) {
2      var currentState = {};
3      // берем значения для игры
4      currentState.currentTheme = qbdoo.currentTheme;
```

```
5       currentState.timeLeft = qbdoo.timeLeft;
6       currentState.score = qbdoo.score;
7       currentState.cardCount = qbdoo.cardCount;
8       currentState.mute = qbdoo.mute;
9       currentState.iterations = qbdoo.iterations;
10
11       // получаем все значения карт и их позиции
12       // используем множество данных для получения значений всех карт
13       if (newgame == 'newgame') {
14         currentState.currentLevel = qbdoo.currentLevel;
15         currentState.score = 0;
16         currentState.gameDuration = qbdoo.gameDuration;
17         SessionStorage.setItem('defaultvalues',
           JSON.stringify(currentState));
18         return;
19       } else {
20         return currentState;
21       }
22     },
```

Функция `storeValues()` вызывается в случае, когда игра инициализируется, и применяется для сохранения стандартных значений, установленных в файле JavaScript. В ходе игры некоторые из этих значений меняются. Когда пользователь запускает новую игру, нажимая соответствующую кнопку, мы сохраняем данные значения, и для этого нам не требуется перезагружать страницу. Значения, задаваемые по умолчанию, снимаются в строках 4–9 и обновляются в строках 14–16, если пользователь начинает новую игру (страница при этом не перезагружается).

При первом вызове функции мы задаем эти значения в объекте `currentState`, имеющем локальную область видимости. В строке 17 мы используем метод JSON `stringify()` для преобразования данного объекта в строку JSON. Затем сохраняем эту строку в `SessionStorage` с ключом `defaultvalues`. Для этого применяется метод `setItem()` из `SessionStorage`. Мы используем ключ `defaultvalues` для получения значения методом `getItem()`. Данная операция выполняется в функции `playGame()`.

Мы также включили в код функцию `alterAValue()` для обновления или возврата стандартных значений, установленных с помощью `storeValues()`, на случай, если пользователь изменит настройки или они изменятся в результате игровых событий:

```
23     alterAValue: function(item, value) {
24       var currentState =
       JSON.parse(SessionStorage.getItem('defaultvalues'));
25       if (value) {
26         currentState[item] = value;
27       } else {
28         qbdoo[item] = currentState[item];
29       }
30       SessionStorage.setItem('defaultvalues',
       JSON.stringify(currentState));
31       return value;
32     },
```

Параметр функции alterAValue() — это элемент, который должен быть получен или установлен. Если элемент устанавливается, то здесь может быть сообщено также опциональное значение для него. Когда пользователь изменяет тему оформления карт или отключает/включает звуковое сопровождение, элемент и его значения отправляются в качестве аргументов при вызове функции. Функция alterAValue() выбирает элемент из SessionStorage, изменяет интересующее нас свойство объекта, а затем пересохраняет в SessionStorage значения игры, задаваемые по умолчанию, так, чтобы отразить новое значение.

Функция получает стандартную настройку из SessionStorage с помощью метода getItem(), это происходит в строке 24. Возвращаемое значение — это строка JSON, которую мы сохранили в SessionStorage ранее, тогда мы применяли метод setItem() с функцией storeValues(). Поскольку мы сохранили строку JSON, именно строка JSON возвращается при получении значения методом getItem(). Выполняем синтаксический анализ этой строки с помощью метода JSON.parse() и таким образом определяем наш объект currentState с локальной областью видимости.

Если два значения передаются функции alterAValue(), то ее первый параметр — это игровое свойство, которое требуется изменить. Объект currentState обновляется так, чтобы отразить это изменение. Если передается всего один параметр, то функция alterAValue() возвращает значение соответствующего игрового свойства.

Мы включили в игру функции pauseGame() и playGame() для постановки игры на паузу и ее последующего возобновления, а также ассоциированные с ними функции:

```
33    pauseGame: function(newgame) {
34
35      var currentState = {}, i, cardinfo = [];
36      if (qbdoo.game.classList.contains('paused')) {
37        qbdoo.playGame();
38        return false;
39      }
40
41      qbdoo.pauseOrPlayBoard('pause');
42      currentState = qbdoo.storeValues();
43      currentState.currentLevel = qbdoo.currentLevel;
44
45      for (i = 0; i < qbdoo.cardCount; i++) {
46        cardinfo.push(qbdoo.cards[i].dataset);
47      }
48
49      currentState.cardPositions = JSON.stringify(cardinfo);
50      LocalStorage.setItem('pausedgame',
      JSON.stringify(currentState));
51
52      qbdoo.clearAll();
53
54    },
```

Функция pauseGame() вызывается при нажатии на соответствующую кнопку (Pause/Game), расположенную в правом верхнем углу экрана. Эта кнопка позволяет переключаться между игрой и паузой. Текущее состояние игры — активное или пауза — определяется по классу в игре. Так, в строках 36–38 отмечаем, что, если игра уже стоит на паузе (поскольку присутствует класс paused), вызывается функция playGame(), описанная в следующем разделе. Если игра еще не поставлена на паузу, то в строке 41 вызывается метод pauseOrPlayBoard(), переключающий игровой класс и сбрасывающий интервал таймера.

Чтобы поставить игру на паузу, необходимо сохранить ее текущее состояние. При этом мы должны сохранить оставшиеся значения открытых карт и их положения, а также состояние игры. Мы получаем некоторые значения, присущие сохраненному состоянию игры, вызывая метод storeValues() в строке 42 (об этом методе мы говорили ранее). Добавляем информацию о текущем уровне к объекту состояния в строке 43 с помощью свойства currentLevel.

Затем мы перебираем все карты, записывая пары «ключ/значение» из множества данных в массив cardinfo. Это происходит в строках 45–47. Этот массив затем преобразуется в строку JSON, созданную нами в браузере как значение ключа pausedgame. Это было сделано в строке 50 в LocalStorage.

В последней строке мы очищаем игровое поле, вызывая метод clearAll(). Эта функция убирает все карты с экрана, для чего значение специального атрибута данных data-value устанавливается в 0 для всех карт. Мы скрываем все карты, значение data-value которых равно нулю. В главе 7 мы поговорим о селекторах и о том, как выбирать элементы по атрибутам.

classList. Обратите внимание: в строке 36 используется термин classList. Объект classList, сопровождающий здесь все узлы DOM, позволяет добавлять, удалять, переключать классы и справляться об их существовании на любом узле DOM. classList возвращает список маркеров атрибута class данного элемента:

- node.classList.add(class) добавляет класс к узлу;
- node.classList.remove(class) удаляет класс из списка классов на данном узле, если этот класс там был. Если класса не было, то ошибка не выдается;
- node.classList.toggle(class) добавляет класс к списку классов узла, если этого класса на данном узле еще не было, и удаляет класс, если он там был;
- node.classList.contains(class) возвращает логическое значение: true, если в списке классов узла DOM содержится конкретный класс, false — в противном случае;
- classList поддерживается в iOS 5 и выше, Android 3 и выше, IE10.

Когда пользователь ставит игру на паузу, кнопка Pause превращается в Play. Данное изменение выполняется с помощью CSS на основе класса paused, добавленного нами в функции pauseGame(). Когда пользователь вновь нажмет эту кнопку, условный оператор в строке 36 вернет true, вызвав функцию playGame():

```
55    playGame: function(newgame) {
56      var cardsValues, cards, i, currentState = {};
```

```
57
58      if (newgame == 'newgame') {
59      currentState =
      JSON.parse(SessionStorage.getItem('defaultvalues'));
60      qbdoo.timeLeft = qbdoo.gameDuration = currentState.gameDuration;
61    } else {
62      // получаем состояние на основании данных из локального хранилища
63      currentState = JSON.parse(LocalStorage.getItem('pausedgame'));
64
65      if (qbdoo.game.classList.contains('paused')) {
66        qbdoo.game.classList.remove('paused');
67      }
68      qbdoo.timeLeft = currentState.timeLeft;
69    }
70    qbdoo.reset('pausedgame');
71
72    qbdoo.currentTheme = currentState.currentTheme;
73    qbdoo.mute = currentState.mute;
74    qbdoo.currentLevel = currentState.currentLevel;
75    qbdoo.score = currentState.score;
76    qbdoo.cardCount = currentState.cardCount;
77    qbdoo.iterations = currentState.iterations;
78
79    qbdoo.setupGame(currentState.cardPositions);
80  },
```

Функция `playGame()` вызывается, когда пользователь возобновляет игру после постановки на паузу, а также когда начинает новую игру, проиграв предыдущую. Если пользователь перезапускает игру, то она должна начаться именно с того момента, в который была поставлена на паузу. Если мы начинаем новую игру, то должны вернуться к ее исходным значениям. Функция `playGame()` обрабатывает обе эти операции.

Когда мы начинаем новую игру (строки 58–60), функция получает стандартные значения для начала игры из хранилища `SessionStorage`. Это делается с помощью метода `getItem()`. Затем производится синтаксический анализ строки, после разбора она присваивается объекту `currentState`. Мы также сбрасываем `gameDuration` к значению, задаваемому по умолчанию, и в это же значение устанавливаем `timeLeft`.

В противном случае, если игра возобновляется после постановки на паузу, мы получаем сохраненное состояние со всеми положениями карт и значениями, взятыми из `LocalStorage`. Для этого используется метод `getItem()`, в котором мы передаем ключ `pausedgame`, а не `defaultvalues`. Ключ `pausedgame` относится к сеансовому хранилищу данных. Так мы получаем `pausedstate` и разбираем строку JSON в объект `currentState`. Функция также изменяет класс игрового поля для сброса класса `paused`.

В конце функции устанавливаются игровые свойства. Они основываются либо на `defaultvalues`, полученном из `SessionStorage`, либо на значениях, актуальных на момент постановки на паузу и полученных из `LocalStorage`. Значения берутся по состоянию перед вызовом метода `setupGame`, запускающего игру.

В строке 70 мы вызываем функцию `reset()`, удаляющую значения, которые соответствовали состоянию игры при постановке на паузу и хранились в `LocalStorage`. Для этого применяется метод `removeItem()`, принимающий в качестве единственного аргумента имя ключа той пары «ключ/значение» из `LocalStorage`, которую мы хотим удалить:

```
81    reset: function(item) {
82      LocalStorage.removeItem(item);
83    }
```

Ставя игру CubeeDoo на паузу, мы используем специальные атрибуты данных и специальное множество данных для извлечения положений карт и их значений. Также здесь задействуется метод `JSON.stringify()`, преобразующий игровой объект в строку JSON, после чего эта строка сохраняется в `LocalStorage`.

Для работы с `SessionStorage` здесь наиболее важна строка 30. Мы создали запись `SessionStorage`, ключом которой служит `defaultvalues`, а значением — строка JSON, представляющая наш текущий объект состояния. В строках 6–25 из фрагмента, приведенного ранее, мы создаем свойства этого объекта, в строках 18–25 используем специальные атрибуты данных и API множества данных, рассмотренный в главе 2. В ходе этих манипуляций мы получаем пары «ключ/значение», определяющие положение и достоинство каждой из карт.

Перезапуская игру, мы получаем элемент из `LocalStorage`:

```
gameState = LocalStorage.getItem('cubeedoo');
```

после чего удаляем сохраненное состояние из памяти с помощью следующего кода:

```
LocalStorage.removeItem('cubeedoo');
```

Элементы, хранимые в `LocalStorage` и `SessionStorage`, видны различным отладчикам (рис. 6.1).

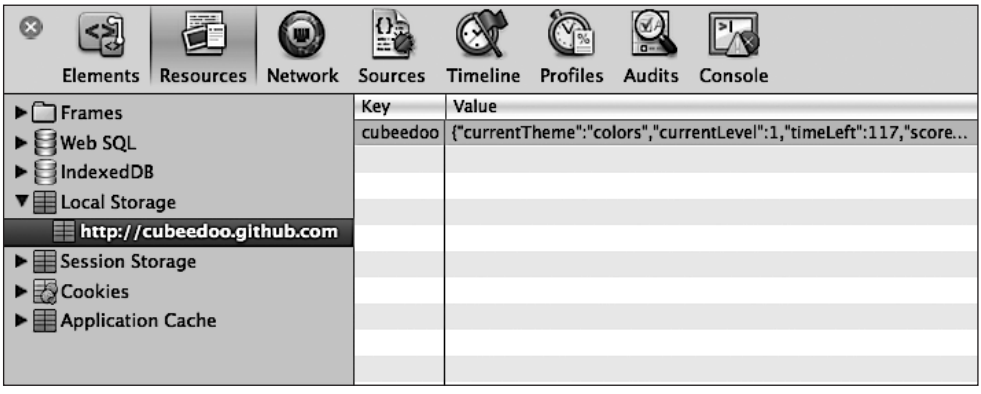


Рис. 6.1. Содержимое LocalStorage и SessionStorage доступно в браузерных отладчиках и веб-инспекторах

Обратите внимание: хотя вы и можете использовать элементы `LocalStorage` и `SessionStorage` в вашем отладчике, в некоторых отладчиках не поддерживается

автоматическое обновление вида. Возможно, потребуется закрыть отладчик и повторно открыть его, чтобы просмотреть текущее состояние ваших ресурсов.

Работа с SessionStorage также очень проста.

Если имени пользователя не существует, мы предлагаем пользователю его ввести, после чего добавляем эту информацию как к ограниченному пространством имен свойству player, так и к SessionStorage:

```
if (!player || player == 'UNKNOWN') {
      player = qbdoo.player = prompt('Enter your name') || 'UNKNOWN';
      SessionStorage.setItem('user', player);
    }
```

Имя пользователя присваивается при загрузке страницы с помощью кода:

```
player: SessionStorage.getItem('user') || '',
```

Если пользователь обновляет страницу, то его имя берется из SessionStorage. В противном случае имя по умолчанию является пустым. Конечно, в данном случае мы могли бы воспользоваться cookie, но эту информацию не требуется отсылать на сервер и получать оттуда обратно. Я также могла бы воспользоваться LocalStorage, но в примере, который мы здесь рассматриваем, я хочу добиться определенной безопасности. Она заключается в том, что, когда пользователь закрывает браузер, его имя должно оттуда удаляться — как при работе с SessionStorage.

Именно из-за этой причудливой черты CubeeDoo недотягивает до полнофункциональной игры. Если пользователь начинает игру и ставит ее на паузу, то и карты, и их актуальное положение сохраняются в LocalStorage. Поскольку имя пользователя находится в SessionStorage, стоит нам закрыть браузер с игрой, поставленной на паузу, как имя пользователя, сохраненное в SessionStorage, будет утеряно. Когда пользователь хочет возобновить игру, которая стояла на паузе, он может снова ввести свое имя. Чтобы предотвратить ситуацию, в которой игру начинает один пользователь, а продолжает уже другой, даже если двумя именами пользуется один и тот же человек, состояние игры на момент постановки на паузу необходимо хранить в SessionStorage, а не в LocalStorage. Если вы не хотите так поступать с именем пользователя, даже если он не играл уже месяц или два, храните это имя в LocalStorage, а не в SessionStorage. Если вас это не волнует — реализуйте любой вариант по собственному выбору. В конце концов, это же так интересно!

Хранение информации в SQL/базе данных

Базы данных для Сети являются новинкой для HTML5 и хорошо поддерживаются. Кстати, по-прежнему хорошо поддерживается и база данных Web SQL, особенно это касается мобильного пространства. Однако ее спецификация уже отвергнута и не будет поддерживаться в тех браузерах, где такая поддержка отсутствовала всегда (например, в IE и Firefox). Поскольку альтернативная база данных IndexedDB пока не вполне готова, а база данных Web SQL полностью поддерживается в браузерах WebKit и Opera Mobile, я расскажу о ней подробно. Однако учитывайте, что

Web SQL устаревает и является лишь временным решением до тех пор, пока в iOS и Android не появится поддержка IndexedDB.

Итак, что же такое базы данных для Сети? Такие базы данных располагаются и долговременно сохраняются в пользовательском браузере на устройстве. Клиентская база данных SQL обеспечивает структурированное хранение данных в виде таблиц со строками и столбцами, а не просто в виде пар «имя/значение». Такая база данных может применяться для хранения электронной корреспонденции на локальном устройстве в почтовом приложении или для хранения заказов в приложении для электронной коммерции. API для взаимодействия с такой базой данных является асинхронным — таким образом гарантируется отсутствие блокировок в пользовательском интерфейсе. `LocalStorage`, в свою очередь, является синхронным. Поскольку взаимодействие с базой данных может одновременно происходить во многих окнах браузера, этот API поддерживает транзакции.

Как и SQL, Web SQL обладает рядом методов и свойств, которые подробно рассмотрены в следующих разделах.

Методы Web SQL

Метод openDatabase. Метод `openDatabase()` объекта окна принимает четыре параметра: имя базы данных, версию, отображаемое имя и размер базы данных. `openDatabase()` создает объект базы данных. Прежде чем вы сможете обратиться к базе данных, ее необходимо открыть. Требуется определить имя, версию, описание и размер базы данных:

```
window.openDatabase(database_name, database_version, display_name, db_size);
```

Этот метод возвращает ссылку на базу данных, которая действует на протяжении всех транзакций этой базы данных.

В CubeeDoo список рекордов можно вести двумя способами: в `LocalStorage` или в Web SQL. Сначала мы проверяем, поддерживается ли на устройстве Web SQL, и если это так — используем ее. Если она не поддерживается, то мы пишем сценарий для работы с `LocalStorage`. Ветвление кода для этой цели организуется на базе свойства `qbdoo.storageType`:

```
storageType: (!window.openDatabase)? "WEBSQL": 'local',
```

Поскольку в настоящий момент база данных Web SQL все еще поддерживается, но в будущем навсегда устареет, обязательно следует сначала проверять ее поддержку, а только затем работать с ней.

Чтобы вести счет рекордов в базе данных, такую базу данных сначала необходимо создать:

```
var dbSize = 5 * 1024 * 1024; // переменная 5 Мбайт для dbSize
if (!qbdoo.db) {
  if (window.openDatabase) {
    qbdoo.db = openDatabase("highscoresDB", "1.0", "Scores", dbsize);
  }
}
```

Метод транзакций. Метод `transaction()` объекта базы данных принимает до трех аргументов: функции обратного вызова, соответствующие всей транзакции, ошибке и успешному обратному вызову. Метод `transaction()` относится к тому объекту базы данных, который мы создали с помощью метода `openDatabase()`, а не к объекту окна, как метод `openDatabase()`, с помощью которого мы создали базу данных. Вы передаете его объекту SQL-транзакции, с которым можно использовать метод `executeSQL()`:

```
db.transaction(transaction_callback, error_callback, success_callback)
```

Метод executeSQL. Метод `executeSQL()` принимает от одного до четырех аргументов: SQL-оператор, аргументы, обратный вызов SQL-оператора и вызов SQL-оператора, срабатывающий при ошибке. Обратный вызов SQL-оператора получает объект транзакции и результирующий объект SQL-оператора, предоставляющий доступ к строкам:

```
db.transaction(function(trnactn) {
  trnactn.executeSql('SELECT * FROM scores', [], callbackFunc,
    db.onError);
  });
```

В CubeeDoo мы комбинируем два метода для установки и получения очков:

```
saveHighScores: function(score, player) {
  qbdoo.db.transaction(function(tx) {
    tx.executeSql("INSERT INTO highscoresTable (score, name, date)
    VALUES (?, ?, ?)", [score, player, new Date()], onSuccess,
    qbdoo.onError);
  });
  function onSuccess(tx, results){
    // не требуется
  }
},
```

Работая с базой данных SQL в качестве офлайнового хранилища данных, можно создавать таблицы, удалять строки и, в принципе, выполнять любые команды SQL, которые поддерживаются на сервере базы данных. Клиентское хранилище данных, использующее SQL (база данных Web SQL), поддерживается в браузерах Safari, Chrome и Opera, но никогда не будет поддерживаться в Firefox и Internet Explorer. Однако, поскольку такая база данных по-прежнему поддерживается в WebKit и Opera, ее можно использовать в мобильных веб-приложениях для оптимизации производительности. В качестве резервного варианта будем сопровождать ее `LocalStorage` до тех пор, пока не будет повсеместно поддерживаться база данных IndexedDB.

Код для таблицы рекордов CubeeDoo

Как было указано ранее, в игре CubeeDoo мы будем использовать для хранения таблицы рекордов базу данных Web SQL. Если Web SQL не поддерживается на устройстве, будет использоваться резервный вариант — `LocalStorage`. Я запрограммировала обнаружение возможностей и установила `storageType` в объекте `qbdoo`:

```
storageType: (window.openDatabase)? "WEBSQL": 'local',
```

Затем добавляю функции для создания таблицы, сохранения рекордов, загрузки рекордов, их отображения и удаления из базы данных. Включаю в код метод для сортировки рекордов, работающий вместе с методом локального хранилища, который получает рекорды. Мне не требуется сортировать результаты из Web SQL при сохранении, поскольку я получаю отсортированные рекорды, используя ASC или DESC в том столбце, сортировка по которому меня интересует.

Мы создали таблицу в нашей базе данных с помощью SQL-оператора create:

```
createTable: function() {
  var i;
  qbdoo.db.transaction(function(tx) {
    tx.executeSql("CREATE TABLE highscoresTable (id REAL UNIQUE, name
    TEXT,
    score NUMBER, date DATE )", [],
      function(tx) {console.log('highscore table created'); },
      qbdoo.onError);
  });
},
```

Для сохранения рекордов используется SQL-оператор insert либо мы сохраняем эту информацию в LocalStorage с помощью метода setItem() (если браузер не поддерживает Web SQL):

```
saveHighScores: function(score, player) {
  if (qbdoo.storageType === 'local') {
    LocalStorage.setItem("highScores",
    JSON.stringify(qbdoo.highScores));
  } else {
    qbdoo.db.transaction(function(tx) {
      tx.executeSql("INSERT INTO highscoresTable (score, name, date)
      VALUES (?, ?, ?)", [score, player, new Date()],
        onSuccess,
        qbdoo.onError);
      });
    function onSuccess(tx,results){
    // не требуется
    }
  }
}
```

У нас есть две функции для загрузки рекордов. Функция выбирается в зависимости от того, работаем мы с LocalStorage или с Web SQL. Мы применяем SQL-оператор select для выборки рекордов из базы данных и сортируем их в порядке убывания:

```
loadHighScoresLocal: function() {
  var scores = LocalStorage.getItem("highScores");
  if (scores) {
    qbdoo.highScores = JSON.parse(scores);
  }
  if (qbdoo.storageType === 'local') {
```

```
    qbdoo.sortHighScores();
  }
},

loadHighScoresSQL: function(){
  var i, item;
  qbdoo.db.transaction(function(tx) {
    tx.executeSql("SELECT score, name, date FROM highscoresTable
    ORDER BY score DESC", [], function(tx, result) {

    for (i = 0, item = null; i < result.rows.length; i++) {
      item = result.rows.item(i);
      qbdoo.highScores[i] = [item['score'], item['name'],
      item['date']];
    } // конец for
  }, onError); // конец выполнения
  function onError(tx, error) {
    if (error.message.indexOf('no such table')) {
      qbdoo.createTable();
    } else {
      console.log('Error: ' + error.message);
        }
      }
    qbdoo.renderHighScores();
  }); // конец транзакции
},
```

Функция `renderHighScores()` создает список рекордов:

```
// выводим рекорды на экран
renderHighScores: function(score, player) {
  var classname, highlighted = false, text = '', i;
  for (i = 0; i < qbdoo.maxHighScores; i++) {
    if (i < qbdoo.highScores.length) {
      if (qbdoo.highScores[i][1] == player && qbdoo.highScores[i][0]
      == score) {
        classname = ' class="current"';
      } else {
        classname = '';
      }
        text += "<li" + classname + ">" +
        qbdoo.highScores[i][1].toUpperCase() +
        ": <em>" + parseInt(qbdoo.highScores[i][0]) + "</em></li> ";
    }
  }
  qbdoo.highscorelist.innerHTML = text;
},
```

SQL-оператор `drop` используется для удаления таблицы, если пользователь хочет очистить список рекордов. Если база данных Web SQL не поддерживается, то функция `reset()` использует метод `removeItem()`, относящийся к `LocalStorage`:

```
eraseScores: function() {
  if (qbdoo.storageType === 'local') {
    qbdoo.reset("highScores");
  } else {
    qbdoo.db.transaction(function(tx) {
      tx.executeSql("DROP TABLE highscoresTable", [],
        qbdoo.createTable,
        qbdoo.onError);
    });
  }
  qbdoo.highscorelist.innerHTML = '<li></li>';
},

onError: function(tx, error) {
  console.log('Error: ' + error.message);
},

reset: function(item) {
  LocalStorage.removeItem(item);
}
```

IndexedDB

В нашем распоряжении вскоре появится база данных IndexedDB для хранения структурированных данных на клиенте. Когда эта база данных будет окончательно доработана и станет полностью поддерживаться, она позволит осуществлять высокопроизводительный поиск данных, опираясь на индексные списки. Хранилище данных DOM удобно использовать с небольшими объемами данных, но IndexedDB представляет собой гораздо более мощное асинхронное решение для хранения гораздо большего количества структурированных данных. Поскольку в настоящее время она еще не слишком широко поддерживается в мобильных браузерах, мы не будем здесь подробно ее рассматривать. Если вы предпочитаете работать с API, спецификации которых пока находятся в разработке, попробуйте полизаполнение, позволяющее использовать синтаксис IndexedDB в браузерах, поддерживающих Web SQL.

Улучшенное пользовательское восприятие

HTML5 не только предоставляет возможности офлайновой работы с веб-приложениями и унифицированную поддержку медиа, но и включает несколько API, с помощью которых пользователь может значительно улучшить пользовательское восприятие ресурса. Так, в HTML5 есть API геолокации, позволяющий браузеру определять местоположение пользователя (разумеется, с согласия пользователя). HTML5 предоставляет веб-работники, оптимизирующие сценарную среду выполнения веб-приложений, микроданные для уточнения семантики веб-контента. API для междокументного обмена сообщениями призван обеспечить безопасную коммуникацию документов друг с другом независимо от того, какой домен является источником каждого из документов. Наконец, существует стандарт ARIA, позволяющий разработчикам создавать доступные активные интернет-приложения.

Геолокация

С помощью геолокации пользователь может по желанию предоставить приложению информацию о своем физическом местоположении. Эта возможность особенно полезна при работе в социальных сетях, геотегинге и картографировании (но, в принципе, может применяться в любых приложениях). Геолокация позволяет улучшить пользовательское восприятие ресурса, создавать контент, социальные графы, а также подбирать рекламу, актуальную в том регионе, где находится пользователь.

Для доступа к геолокационной информации браузеру необходимо разрешение пользователя (рис. 6.2). Возможность геолокации выбирается по принципу явного согласия (opt-in): когда ваше веб-приложение запрашивает геолокационную информацию, браузер должен получить у пользователя разрешение на это. Такое предложение выводится на экран в виде баннера или информационного окна. Пользователь может дать такое разрешение или не дать его; кроме того, можно запомнить выбор для каждого конкретного сайта. Если пользователь разрешил задействовать геолокацию, то геолокационная информация будет доступна вашим сценариям, а также любым сторонним сценариям, присутствующим на странице. Приложение сможет определять, где находится пользователь, а также обновлять геолокационную информацию в соответствии с передвижениями пользователя.

Рис. 6.2. Пользователь должен дать разрешение на то, чтобы информация о его местоположении была доступна для функций геолокации

Информация о местоположении является приблизительной и собирается на основании IP-адресов, контактов с сотовыми вышками, работы в сетях Wi-Fi, GPS и даже на базе тех данных, которые пользователь вводит вручную. При всей приблизительности этих методов вы иногда с удивлением заметите, что такая информация чертовски точна.

Геолокационный API не регламентирует способа, который должен применяться на клиенте для сбора информации о местоположении, — важно лишь то, что данные должны поступать в систему стандартным образом. Геолокационный API является асинхронным.

Чтобы определить, поддерживается ли в браузере геолокация, пользуйтесь следующим кодом:

```
if (navigator.geolocation) {
    // геолокация поддерживается
}
```

Геолокационный объект предоставляет методы `getCurrentPosition()` и `watchCurrentPosition()`, которые асинхронно возвращают актуальное местоположение пользователя. Такое определение местоположения может происходить как однократно, так и непрерывно. Метод `watchCurrentPosition()` может применяться в приложениях, которые активно занимаются определением местоположения, — например, в GPS-навигаторах. В наших веб-приложениях не требуется информация о направлении движения, поэтому нас вполне устроит метод `getCurrentPosition()`. Мы не будем истощать заряд батареи слишком частыми геолокационными запросами. Для функционирования CubeeDoo геолокационная информация не требуется, но мы тем не менее можем ее узнать:

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(handle_success,
    handle_errors);
}
```

В случае успеха функция обратного вызова возвращает актуальную позицию с объектом `coords`, содержащим широко известные свойства `latitude` и `longitude`, а также свойства `altitude`, `accuracy`, `altitudeAccuracy`, `heading` и `speed`. Следующий сценарий выдает информационное сообщение с актуальными широтой и долготой.

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(handle_success,handle_errors);

function handle_success(position) {
    alert('Latitude: ' + position.coords.latitude + '\n Longitude: '
    + position.coords.latitude);
  }

function handle_errors(err) {
  switch(err.code) {
    case err.PERMISSION_DENIED:
      alert("User refused to share geolocation data");
```

```
      break;
    case err.POSITION_UNAVAILABLE:
      alert("Current position is unavailable");
      break;
    case err.TIMEOUT:
      alert("Timed out");
      break;
    default:
      alert("Unknown error");
      break;
    }
  }

}
```

В случае успеха обратные вызовы от методов `getCurrentPosition()` и `watchCurrentPosition()` возвращают объект местоположения с объектом `coords`, обладающим следующими свойствами:

- `position.coords.latitude`;
- `position.coords.longitude`;
- `position.coords.altitude`;
- `position.coords.accuracy`.

Метод `watchCurrentPosition()` возвращает также следующие свойства:

- `position.coords.heading`;
- `position.coords.speed`.

Эти свойства вполне понятны без объяснения. Геолокация поддерживается во всех браузерах, кроме Kindle и Opera Mini, и не так давно была реализована на ПК (в IE9 и выше).

В игре CubeeDoo не требуется геолокационная информация, но мы включим в игру возможность указания местоположения на карте:

```
1  function getLocation() {
2    if (navigator.geolocation) {
3      navigator.geolocation.getCurrentPosition(success, error);
4      console.log('got position');
5      } else {
6      error('not supported');
7      }
8    }
9    function error(text) {
10     text = text || 'failed';
11     console.log(text);
12   }
13   function success(location) {
14     var lat = location.coords.latitude;
15     var long = location.coords.longitude;
16     var url = "http://maps.google.com/maps?q=" + lat + "," + long;
17  }
```

Функция `getLocation()` в строке 2 проверяет, поддерживается ли в браузере геолокация. Обратите внимание: геолокация внедряется в объекте `navigator` (а не в объекте окна документа, в отличие от большинства используемых нами методов и свойств). Мы получаем актуальную позицию с помощью метода `getCurrentPosition()` объекта `geolocation` в строке 3. В случае успеха обратный вызов использует свойства `latitude` и `longitude` возвращенного объекта `coords` для того, чтобы в строках 14–16 установить метку на карте Google. В строке 9 мы поставили функцию обратного вызова, срабатывающую при ошибке. При ошибках обычно выдается сообщение о задержке, отказе в доступе или недоступности информации о местоположении.

Веб-работники

Весь код JavaScript, в частности отвечающий за пересчет положения элементов на странице и их перерисовку, работает в единственном потоке пользовательского интерфейса. В этом же потоке происходит перерисовка, связанная с пользовательскими взаимодействиями, анимация, не снабженная аппаратным ускорением и т. д. Если приходится выполнять задачу, связанную с интенсивным использованием сценариев, то работа браузера может очень сильно замедлиться, что плохо сказывается на пользовательском восприятии. Веб-работники — это технология JavaScript, позволяющая делегировать сложные задачи другим процессам так, чтобы сценарии работали параллельно. Таким образом, главный поток может заниматься наиболее интересными задачами пользовательского интерфейса, а веб-работник в это время справится со всей сложной работой, освободив от нее главный сценарный поток. Веб-работники полезны, когда в вашем коде требуется выполнять интенсивные вычисления, нагружающие процессор, и не блокировать при этом поток пользовательского интерфейса.

Благодаря работникам веб-контент может запускать сценарии в фоновых потоках — это касается даже AJAX. Рабочий поток выполняет все задачи, совершенно не вмешиваясь в функционирование пользовательского интерфейса.

Все мы представляем себе, как веб-приложения иногда притормаживают и на экране возникает изображение песочных часов или радужного круга. Продолжать работу со страницей можно только после того, как это изображение исчезнет. Веб-работники — решение такой проблемы. Они выполняют весь JavaScript в фоновых потоках, оставляя главный поток пользовательского интерфейса исключительно для манипуляций с DOM и при необходимости для перерисовки страницы. Если действия, выполняемые в фоновом потоке, приводят к изменениям в объектной модели документа, то для внесения таких изменений веб-работники должны послать своим элементам-создателям соответствующие сообщения. Например, можно задействовать метод `postMessage()`.

Если ваш JavaScript содержит какие-либо ресурсозатратные вычисления, то этот код можно передать веб-работнику, чтобы главный поток мог свободно продолжать работу. Перед тем как создавать веб-работник, убедитесь, что такая технология поддерживается в браузере. Веб-работники могут брать на себя очень

сложные вычисления, и если они не поддерживаются, то браузер может просто рухнуть под грузом такой работы:

```
if (window.Worker) {
  // браузер поддерживает веб-работники
}
```

Чтобы создать веб-работник, необходимо вызвать конструктор `Worker()` и указать URI того сценария, который должен выполняться в рабочем потоке. URI относителен к файлу, вызывающему сценарий:

```
if (window.Worker) {
  var webWorker = new Worker('subcontractor.js'); // создаем его
}
```

Коммуникация с веб-работниками выполняется с помощью метода `postMessage()`. Создав веб-работник, установите в качестве значения его свойства `onmessage` соответствующую функцию, которая будет получать уведомления от веб-работника. Для завершения работника применяется метод `terminate()` или `close()`. Метод `terminate()` срабатывает немедленно, останавливая все текущие процессы:

```
if (window.Worker) {
  var webWorker = new Worker('subcontractor.js'); // создаем его
  webWorker.postMessage(some_message);
}
```

В работнике (в данном случае `subcontractor.js`) мы получаем сообщение от главного потока и оперируем этим сообщением:

```
// в файле subcontractor.js
self.onmessage = function(event) {
    // обрабатываем сообщение
    var stuff = event.data;
    // и отправляем его обратно в главный поток
    postMessage(stuff);
};
```

Работники могут использовать задержки и интервалы — точно как и главный поток. Это бывает удобно, например, если вы желаете запускать работник в коде периодически, а не использовать его безостановочно. Для управления работником можно задействовать методы `setTimeout()`, `clearTimeout()`, `setInterval()` и `clearInterval()`.

Рабочие потоки обладают доступом к глобальной функции `importScripts()`, с помощью которой могут импортировать сценарии или библиотеки в свою область видимости. В качестве параметров эта функция принимает ноль или более URI или ресурсов для импортирования:

```
/* импортируем два сценария */
importScripts('scripts/jquery-min.js', 'application.js');
```

Веб-работники не имеют доступа ни к DOM, ни к консоли. Кроме того, вы можете получить ошибку безопасности при тестировании на локальной машине.

Поэтому разработка с применением веб-работников бывает несколько сложнее, чем на обычном языке JavaScript.

В CubeeDoo у нас нет никакого сложного кода на JavaScript, нет и фоновых процессов AJAX. Но я включила в код веб-работник, который выполняет функцию сортировки рекордов. Разумеется, отсортировать пять чисел можно и без помощи веб-работника. Однако если бы мы сохраняли в таблице рекордов 1 000 000 наилучших результатов, то веб-работник нам очень пригодился бы (хотя хранить такое количество информации в `LocalStorage` нерационально).

Если бы мы использовали веб-работники для сортировки рекордов, то могли бы заменить сортировочную функцию вызовом веб-работника:

```
var webWorker = new Worker('js/sort.js');
  webWorker.postMessage(qbdoo.highscores);
  webWorker.onmessage(function(event) {
      qbdoo.highscores(event.data);
  });
```

Сценарий веб-работника, в свою очередь, должен ожидать и принимать значение `highscores` через свойство `onmessage`, а также отсылать отсортированные результаты обратно с помощью `postMessage`:

```
self.onmessage = function(event) {
    var sortedScores = sortScores(event.data);
    self.postMessage(sortedScores);
};
```

Микроданные

Еще одна новая возможность HTML5 называется *микроданными*. Микроданные никак не отражаются на видимых характеристиках вашего сайта, однако они исключительно важны при поисковой оптимизации и извлечении данных.

Микроданные позволяют обходиться без *микроформатов*. Микроформаты — это стандартизированные множества словарей, которые пригодны для чтения как человеком, так и машиной. При создании веб-страниц применяются соглашения, используемые для описания распространенных типов информации — мероприятий, обзоров, данных из адресной книги, а также календарных событий. Такое описание выполняется с помощью атрибутов `class`. Каждая сущность, например персона, событие или компания, обладает собственными свойствами, в которых могут записываться имя, адрес и номер телефона.

Микроданные позволяют вам создавать собственные словари, не ограниченные возможностями HTML5, и расширять веб-страницы с помощью собственной семантики. С микроданными используются некоторые атрибуты, появившиеся только в HTML5, — `itemscope`, `itemprop`, `itemref` и `itemtype`.

Атрибут `itemscope` применяется для создания компонента и указывает, что область применения компонента начинается с открывающего тега, сопровождаемого данным атрибутом, и заканчивается закрывающим тегом соответствующего элемента. Атрибут `itemprop`, означающий свойство компонента, используется

для добавления свойства к компоненту. Если itemprop связан со свойствами, не являющимися потомками данного itemprop, то их можно ассоциировать с нужным компонентом itemprop с помощью itemref — атрибута ссылки на компонент. Сущность, имеющая атрибут itemscope, также обладает itemref, который принимает в качестве значения список разделенных пробелами ID-сущностей. Эти сущности должны быть просмотрены поисковым роботом наряду с потомками itemprop.

Микроданные особенно полезны в таких контекстах, где авторы и читатели могут сотрудничать и находить новые возможности применения элементов разметки. Вы можете создавать собственные типы микроданных или использовать готовые словари данных. Некоторые подобные словари предоставляются на сайте http://www.data-vocabulary.org/.

Сравнение микроданных и микроформатов

Микроформаты очень похожи на микроданные. Фактически микроданные можно считать развитием старой идеи микроформатов. Микроданные призваны справиться с недостатками микроформатов, но не такими сложными способами, которые применяют альтернативные системы, в частности RDF. Вместо использования новых атрибутов itemscope, itemprop, itemtype и т. д. микроформаты позволяют по-новому задействовать атрибут class и предоставляют для данных семантику, читаемую как человеком, так и машиной. В сущности, такая семантика и есть микроданные.

Как правило, микроформаты используют атрибут class с открывающими HTML-тегами (зачастую или) для присваивания сущностям и их свойствам кратких информативных наименований. В отличие от микроданных, микроформаты *не входят* в состав спецификации HTML5.

Вот небольшой пример разметки HTML5, в котором представлена моя контактная информация:

```
<ul>
  <li><img src="http://standardista.com/images/estelle.jpg"
    alt="photo of Estelle Weyl"/></li>
  <li><a href="http://www.standardista.com">Estelle Weyl</a></li>
  <li>1234 Main Street<br />San Francisco, CA 94114</li>
  <li>415.555.1212</li>
</ul>
```

А вот та же информация, записанная с применением микроформата hCard (для персоналий):

```
<ul id="hcard-Estelle-Weyl" class="vcard">
  <li><img src="http://standardista.com/images/estelle.jpg"
    alt="photo of Estelle Weyl" class="photo"/></li>
  <li><a class="url fn" href="http://www.standardista.com">Estelle
  Weyl</a></li>
  <li class="adr">
  <span class="street-address">1234 Main Street</span>
  <span class="locality">San Francisco</span>, <span
   class="region">CA</span>,
  <span class="postal-code">94114</span>
```

```
  <span class="country-name hidden">USA</span>
</li>
  <li class="tel">415.555.1212</li>
</ul>
```

В первой строке код `class="vcard"` указывает, что HTML в элементе `<ul>` описывает персону, в данном случае меня. Этот микроформат для предоставления информации о людях называется hCard, но в HTML обозначается как vCard. Да, немного запутанно, но это не опечатка.

В оставшейся части примера описаны свойства персоны: фотография, полное имя, адрес, URL и телефон. У каждого свойства есть атрибут класса, описывающий это свойство. Например, `fn` описывает полное имя (full name).

Свойства могут содержать другие свойства. Например, в свойстве `adr` заключены все компоненты моего вымышленного адреса: название улицы, квартала, района, почтовый код. Применив немного CSS, мы можем скрывать элементы с помощью класса `hidden` и добавить разрыв строки между адресом и названием квартала. Для создания своего hCard зайдите на сайт http://microformats.org/code/hcard/creator.

Аналогичный контент можно записать с помощью микроданных:

```
<ul id="hcard-Estelle-Weyl" itemscope
    itemtype="http://microformats.org/profile/hcard">
  <li><img src="http://standardista.com/images/estelle.jpg"
    alt="photo of Estelle Weyl" class="photo"/></li>
  <li><a href="http://www.standardista.com" itemprop="fn">Estelle
     Weyl</a></li>
  <li itemprop="adr">
    <span itemprop="street-address">1234 Main Street</span>
    <span itemprop="locality">San Francisco</span>,
    <span itemprop="region">CA</span>,
    <span itemprop="postal-code">94114</span>
    <span class="hidden" itemprop="country-name">USA</span>
  </li>
  <li itemprop="tel">415.555.1212</li>
</ul>
```

Или совместить два варианта:

```
<ul id="hcard-Estelle-Weyl" class="vcard" itemscope
    itemtype="http://microformats.org/profile/hcard">
  <li><img src="http://standardista.com/images/estelle.jpg"
    alt="photo of Estelle Weyl" class="photo"/></li>
  <li><a class="url fn" href="http://www.standardista.com"
    itemprop="fn">Estelle Weyl</a></li>
  <li class="adr" itemprop="adr">
    <span class="street-address" itemprop="street-address">1234 Main
     Street</span>
    <span class="locality" itemprop="locality">San Francisco</span>,
    <span class="region" itemprop="region">CA</span>,
    <span class="postal-code" itemprop="postal-code">94114</span>
```

```
    <span class="country-name hidden" itemprop="country-name">USA</span>
</li>
  <li class="tel" itemprop="tel">415.555.1212</li>
</ul>
```

Микроданные не изменяют внешнего вида документа, а просто оптимизируют его семантику. Поисковики не отображают контент, который должен быть невидимым для пользователя. Полезно предоставлять поисковикам более детальную информацию, даже если вы не собираетесь показывать эту информацию посетителям вашей страницы. Чтобы посодействовать превращению Сети в единую глобальную базу данных, требуется возможность преобразования доступных данных в значимые фрагменты информации. Микроданные и микроформаты помогают разъяснить непонятные данные синтаксическим анализаторам.

API микроданных

API объектной модели документа для работы с микроданными пока поддерживается не очень хорошо, но с его помощью уже можно получить доступ к компонентам микроданных. Метод `document.getItems(itemType)` возвращает список nodeList, содержащий компоненты с заданными типами либо все типы, если аргумент не указан. Метод `document.getItems()` возвращает список nodeList, содержащий все компоненты микроданных с конкретной страницы, если аргумент не указан. Можно указать в качестве аргумента URL на интересующий вас тип компонентов, чтобы вернуть компоненты лишь этого типа.

Вернув список nodeList, вы можете получить доступ к различным свойствам с помощью атрибута `properties`:

```
var allMicrodata = document.getItems();
var firstItemName = allMicrodata.properties['name'][0].itemValue;
```

Каждый компонент представлен в DOM тем элементом, в котором находится соответствующий атрибут `itemscope`.

Междокументный обмен сообщениями

С помощью междокументного обмена сообщениями обеспечивается коммуникация между отдельными документами независимо от их исходного домена. Обмен информацией строится так, чтобы защитить ресурс от атак с применением межсайтового скриптинга.

Зачастую в веб-приложениях присутствуют сервисы, относящиеся к различным доменам. Сегодня встраиваемые приложения-мэшапы создаются способом, который сопряжен с множеством рисков. Если вы включаете в сайт сторонний код JavaScript, то не можете контролировать эти внешние сценарии. Они же обладают доступом к cookie вашего домена, могут подделывать запросы, которые выглядят так, как будто исходят от пользователя. Встроенные фреймы не решают эту проблему, поскольку ваш документ не может обмениваться информацией с содержимым такого встроенного фрейма, если данный фрейм и ваша страница относятся к разным доменам.

API HTML5 для междокументного обмена сообщениями призван решить обе эти проблемы. Он позволяет регистрировать обработчики событий на получение входящих сообщений с других доменов, а также обеспечивает отправку сообщений на другие домены.

Чтобы убедиться, что сообщение пришло с того домена, откуда мы его ожидаем, выполним код:

```
window.addEventListener('message', function(e) {
  if (e.origin == 'http://the_domain.com') {
    // Источник сообщения верифицирован. Перед использованием проверяем,
    // имеет ли оно правильный формат
}, false);
```

Отсылаем сообщение на другой домен:

```
var theFrame = document.getElementById("myIFrame").contentWindow;
theFrame.postMessage("The message", "http://www.the_domain.com");
```

CORS: обмен ресурсами с запросом происхождения

Поскольку CubeeDoo — небольшое самодостаточное приложение с ограниченным количеством пользователей, которое не предполагает интеграции со сторонними приложениями, мы не используем в нем междокументный обмен сообщениями или обмен ресурсами с запросом происхождения. Если вы создаете приложения для более широкой аудитории, то, вероятно, будете пользоваться сетями доставки контента (CDN) или интегрировать сторонние программы. Например, если вы храните шрифт в сети доставки контента, то понадобится задействовать технологию CORS, чтобы сообщить браузеру Firefox или Internet Explorer: да, разрешено отображать шрифты, взятые с другого домена.

Безопасность

Безопасность — одна из важнейших проблем при междокументном обмене сообщениями. Всегда проверяйте свойство `origin`, чтобы гарантированно принимать сообщения лишь с тех доменов, с которых вы ожидаете их получить. Убедившись, что сообщение пришло с нужного сервера, необходимо проверить также, правильный ли формат имеют данные. Чтобы избежать взлома, никогда не полагайтесь на чужие, непроверенные серверы.

Доступные активные интернет-приложения (ARIA)

HTML5, как и более ранние версии этого языка, можно сделать полностью доступным. Просто требуется спланировать работу. ARIA (стандарт доступности активных интернет-приложений) — первая часть HTML5, которая поддерживается во всех современных браузерах. В самых популярных библиотеках JavaScript также предоставляется поддержка для реализации ARIA. Кроме применения ARIA HTML5

делает еще один важный шаг к повышению доступности: в этой версии языка удается обойтись без технологии Flash и встроенных объектов. Да, ведутся разговоры о том, как бы сделать Flash доступнее, но ничего для этого не делается. В HTML5 используются элементы <video>, <audio>, <svg> и <canvas>, которые, за исключением <canvas>, по природе своей доступны. На них очень удобно нацеливаться в рамках DOM, благодаря чему доступность значительно повышается.

По мере того как мы создаем все более динамические веб-приложения, контент становится все менее доступным, особенно для пользователей, возможности которых могут быть по-разному ограничены. Можно использовать свойства ARIA для предоставления экранным дикторам и другим вспомогательным технологиям информации о базовом типе, состоянии и изменениях, создаваемых с помощью JavaScript-виджетов. Например, при проверке стоимости авиабилетов пользователь может выбрать вариант «только рейсы без пересадок» — и получить динамический отклик, в результате которого стоимость билетов будет пересчитана без перезагрузки страницы. Если пользователь плохо видит, он может и не заметить такого обновления. Другой пример. Пользователь просматривает страницу с финансовой информацией. Как он узнает, что биржевой тикер постоянно обновляется? API ARIA предоставляет ненавязчивые (а также навязчивые) способы донесения такой информации до пользователя.

Как было указано ранее, аббревиатура ARIA расшифровывается как «стандарт доступности активных интернет-приложений». С распространением интернет-приложений на все большем количестве сайтов активно используется JavaScript, а страницы могут обновляться без перезагрузки. В результате возникают проблемы с доступностью веб-страниц, которые не были затронуты в документе «Руководство по обеспечению доступности веб-контента» (WCAG 1). Дело в том, что эта спецификация «решала» проблемы доступности всего одним разумным примечанием: «Работа сайта не должна зависеть от JavaScript».

Когда в Интернете появилось довольно много веб-приложений, а не просто сайтов, обязательно требующих JavaScript для реализации вспомогательных технологий, возникли новые проблемы с доступностью. Стандарт ARIA призван справиться с некоторыми из этих проблем. В нем задействуются роли, состояния и свойства, помогающие обеспечить доступность вашего контента для вспомогательных технологий. Кроме того, эти дополнительные усовершенствованные семантические подсказки помогают повысить и доступность статического контента.

Реализуя на сайте возможности улучшения доступности, предоставляемые в ARIA, вы значительно упрощаете работу с сайтом или приложением. ARIA, оперируя ролями, состояниями и свойствами, позволяет разработчику делать код семантически насыщенным, что особенно удобно для тех пользователей, которые прибегают к вспомогательным технологиям. ARIA обеспечивает семантическое описание элемента или поведения виджета, содержит информацию о группах и относящихся к ним элементах. Состояния и свойства ARIA доступны на уровне объектной модели документа.

ARIA напоминает атрибут `title` в том отношении, что является чистой оптимизацией и никоим образом не вредит вашему сайту. Иными словами, нет никаких

разумных причин не пользоваться этими возможностями! В большинстве библиотек JavaScript, в частности в jQuery и Dojo, стандарт ARIA уже поддерживается. Современные браузеры, в том числе IE8, также поддерживают ARIA.

Наиболее важные и в то же время наиболее удобные для включения функции ARIA связаны с использованием атрибута `role`, а также состояний и свойств.

- Используйте роли, атрибуты и свойства ARIA только в тех случаях, когда обычная HTML-разметка не поддерживает всей необходимой семантики.
- Используйте атрибут `role` из ARIA в случаях, когда разметку требуется семантически усовершенствовать, а также в ситуациях, когда элементы используются в нетипичных для них семантических значениях. В частности, это требуется делать при установке взаимосвязей между родственными элементами (группировании).
- Устанавливайте свойства и исходное состояние для динамических элементов и таких элементов, которые могут быть изменены по инициативе пользователя. Состояния, например «отмечено», — это свойства, которые могут часто изменяться. Вспомогательные технологии, поддерживающие ARIA, реагируют на такие изменения состояний и свойств.
- Поддерживайте полную, удобную клавиатурную навигацию. Все элементы должны при необходимости получать фокус с клавиатуры.
- Пользовательский интерфейс нужно создавать так, чтобы он визуально совпадал с определенными состояниями и свойствами в тех браузерах, которые поддерживают псевдоклассы ARIA CSS

С помощью атрибута `role` разработчик может создавать семантическую структуру на базе элементов, назначение которых специально переопределил. Например, пользователь с хорошим зрением может и не заметить, что элемент `<span>` используется как флажок, но атрибут `role` позволяет сделать эту, казалось бы, несемантическую разметку доступной, удобной и пригодной для взаимодействия со вспомогательными технологиями. Однако если атрибут `role` уже установлен, его значение не должно динамически изменяться, так как это может запутать вспомогательную технологию.

Например, дизайнер настаивает на том, что все флажки на странице должны выглядеть определенным образом. Вы говорите: «Это невозможно». Вы знаете, что с помощью CSS можно сделать `<span>` неотличимым от флажка и пользователь с хорошим зрением даже не заметит, что вы используете `<input type="checkbox"...`. Но вы задумываетесь и об обеспечении доступности, поэтому понимаете, что экранный диктор не поймет, что перед ним флажок, а не `<span>`. Если в коде будет присутствовать атрибут ARIA `role` и как браузер, так и экранный диктор будут поддерживать ARIA, то вы сможете обеспечить доступность переопределенного элемента следующим образом:

```
<span role="checkbox" aria-checked="true" tabindex="0"/>
```

В предыдущем примере недостаточно просто задействовать атрибут `role`. Если вы применяете элементы `<span>`, оформленные как флажки, то вам понадобится включить эквивалентные, но ненавязчивые события касания, клавиатуры и мыши

для соответствующих взаимодействий. Рекомендуется всегда применять для решения задачи такой элемент, который семантически наиболее точно ей соответствует, поэтому вышеуказанный код на практике использовать нежелательно.

В настоящее время существует более 60 ролей ARIA, в частности:

alert	dialog	listitem	option	spinbutton
alertdialog	directory	log	presentation	status
application	document	main	progressbar	tab
article	form	marquee	radio	tablist
banner	grid	math	radiogroup	tabpanel
button	gridcell	menu	region	textbox
checkbox	group	menubar	row	timer
columnheader	heading	menuitem	rowgroup	toolbar
combobox	img	menuitemcheckbox	search	tooltip
complementary	link	menuitemradio	scrollbar	tree
contentinfo	list	navigation	separator	treegrid
definition	listbox	note	slider	treeitem

Можно обойтись и без использования `role`, если элемент задействуется строго по назначению. Так, незачем указывать `role="checkbox"` для `<input type="checkbox"/>`. Однако если вы хотите, чтобы `<span>` выглядел и функционировал как флажок, то укажите ролевой атрибут ARIA `<span role="checkbox">`. Выберите тип роли из приведенного ранее списка, причем роль должна максимально полно соответствовать назначению элемента, который вы используете несемантическим образом.

Если вы хотите подробнее познакомиться с веб-инициативой по разработке стандарта доступности активных интернет-приложений WAI-ARIA, можете начать с сайта http://www.w3.org/WAI/intro/aria.php.

Резюме

В этой главе я хотела дать вам представление об API, вошедших в состав HTML5. Каждый из разделов этой главы можно было бы расширить до целой книги. На самом деле уже написаны целые книги, посвященные отдельным микроформатам. Другие технологии, например междокументный обмен сообщениями, появились слишком недавно, и писать о них книги пока рано.

7 Переход на новый качественный уровень с помощью CSS3

Спецификация CSS3 разрабатывалась более 10 лет. Некоторые рекомендуемые этой спецификацией свойства в течение длительного времени поддерживались WebKit/Blink, Opera и Firefox. Теперь почти все они поддерживаются и браузерами IE10 и IE11! Настало время и нам воспользоваться некоторыми весьма впечатляющими новыми (а иногда и не такими уж новыми) свойствами. Особенно актуальной для мобильных браузеров является поддержка селекторов CSS.

Если не приходится волноваться за устаревшие версии Internet Explorer (IE8 и более ранние), то можно воспользоваться любыми CSS3-селекторами. Все они поддерживаются всеми современными браузерами, а также всеми браузерами смартфонов и планшетных компьютеров.

Спецификация CSS2 была принята в 1998 году. И с того времени велась разработка CSS3, которая продолжается до сих пор и, наверное, никогда не закончится.

Спросите почему? Спецификация CSS 2.1 и более ранних версий имела цельный характер. А CSS3 является охватывающим понятием для всех спецификаций, появившихся после CSS 2.1. Вместо монолитности она теперь состоит из модулей для каждого CSS-компонента, и к ней продолжают добавляться новые модули и новые свойства. Некоторые из этих спецификаций, например относящиеся к цветовому оформлению и селекторам, относятся к уровню 3 (level 3), в то время как началась работа над уровнем 4. Все остальные спецификации находятся на уровне 1. Все эти модули независимо от уровня, на котором они находятся, объединяются общим понятием CSS3.

WebKit/Blink, Opera и Mozilla не стали дожидаться выпуска законченных модульных спецификаций. А начиная с Internet Explorer 9 к игре наконец-то присоединилась и компания Microsoft. Большинство рекомендуемых свойств в течение многих лет были частью черновой спецификации. В подавляющей массе областей рекомендуемые спецификации приобрели довольно стабильное состояние, что позволило основной массе разработчиков браузеров приступить к реализации имеющихся в них свойств CSS3.

В то время как веб-разработчикам по-прежнему придется обеспечивать постепенное упрощение наших сайтов на настольных системах для различных версий

IE, повсеместное распространение поддержки CSS3 и HTML5 на рынке мобильных устройств и на операционных системах, отличных от Windows (на PlayStation, Wii и т. д.), означает, что мы не просто развлекаемся с этими технологиями, а можем их реализовать. Благодаря поддерживающим HTML5 и CSS3 браузерам, являющимся браузерами по умолчанию в большинстве мобильных телефонов, планшетных компьютеров и почти всех настольных устройств, работающих под операционными системами, отличными от Windows, наша аудитория может достичь миллионов, если не миллиардов пользователей.

В данной главе будут рассмотрены селекторы CSS и порядок их использования в отношении целевых DOM-узлов в JavaScript (без использования jQuery). Но сначала будет дан краткий обзор синтаксиса CSS3, который аналогичен синтаксису предыдущих CSS-рекомендаций. Затем, прежде чем погрузиться в селекторы, определяющие наши запросы, углубленно рассмотрим селекторы CSS3, которые позволяют применять семантическое нацеливание на элементы HTML-документов, включая нацеливание на любой элемент в вашем документе, без обращения к HTML или без добавления атрибутов класса или идентификатора.

CSS: определение и синтаксис

Прежде чем углубиться в CSS3, нужно освоиться с основами создания набора правил CSS. Кратко расскажем о синтаксисе тем, кто, может быть, еще не силен в CSS, а затем углубимся в некоторые усовершенствованные технологии нацеливания на элементы с использованием CSS-селекторов.

Прежде всего нужно узнать, что означает CSS.

Каскадные таблицы стилей (Cascading Style Sheets (CSS)) составляют презентационный уровень веб-информации. С помощью CSS можно определить внешний вид вашего сайта в одном месте, и этот один файл может влиять на внешний вид всего сайта. При изменении дизайна технология CSS позволяет вам вносить изменения в один презентационный файл, и эти изменения тут же отразятся на всем сайте. Когда ваш начальник или клиент говорит: «Знаете, мое мнение изменилось, давайте сделаем эти ссылки не зелеными, а фиолетовыми» — и при этом сайт запрограммирован правильно — с использованием внешнего CSS-файла, — на эти изменения может уйти всего одна минута. И неважно, сколько страниц в вашем сайте, одна или миллион. Внесите изменения всего в одну строку своего CSS-файла, и это позволит успешно обновить внешний вид всех ваших веб-страниц.

В ранних версиях HTML существовал ряд презентационных элементов, таких как `<font>` и `<center>`, которые использовались с целью предоставления веб-мастерам возможности определять дизайн сайтов. Но соответствующие веб-стандарты предписывают отделение уровня содержимого (HTML) от презентационного уровня (CSS) и поведенческого уровня (JavaScript). При использовании таких презентационных элементов, как `<font color="green">`, выполнить запрос на изменение цвета на фиолетовый было бы весьма непросто. Для этого пришлось бы обновлять каждое появление кода `color="green"` на каждой веб-странице.

Использование элементов в целях презентации отбросит нас в далекий 1996 год! Фактически многие из презентационных элементов, широко использовавшихся в 1996 году, являются устаревшими или превратились в устаревшие из-за использования для презентации технологии CSS. Не стоит в целях получения презентационного эффекта пользоваться , <b>, <i>[1], <font>, <tt> или другими презентационными элементами, даже если они все еще не считаются устаревшими. Вместо них нужно использовать CSS.

Вот как это делается...

Синтаксис CSS

Перед реализацией правил с помощью селекторов, свойств и значений нужно изучить синтаксис. Таблицы стилей состоят из правил, представляющих собой селекторы, за которыми следуют блоки инструкций со свойствами и значениями. Большинство CSS-правил имеют примерно следующий вид:

```
селектор {
  свойство1: значение1;
  свойство2: значение2;
}
```

Селектор сообщает браузеру, какому элементу (или элементам) это правило соответствует. *Свойство* относится к тому свойству элемента, на которое нужно оказать воздействие, а *значение* является тем самым значением, которое нужно установить для данного свойства данного элемента.

Свойства поддерживают определенные типы значений и/или ключевые слова значений, которые рассматриваются в главе 8.

CSS-селекторы позволяют указывать на те элементы разметки, к которым будут применяться стили, определяемые в *блоке определения стилей*. Этот блок состоит из свойств и значений, заключенных в фигурные скобки. Все свойства и значения для того или иного правила заключаются в фигурные скобки, производя тем самым блок объявления:

```
p {
  color: blue;
  margin-bottom: 12px;
}
```

Инструкция гласит: «Текст абзаца должен быть синим, а ниже его должно быть поле размером 12 пикселов».

Заметьте, что в этом примере значения и свойства отделены друг от друга двоеточием, а каждое объявление заканчивается точкой с запятой. В последнем объявлении блока точка с запятой официально считается необязательной, но ее не стоит опускать! Можно, конечно, сэкономить несколько символов, не включая

[1] Как отмечалось в главе 3, в HTML 5 элементы <i> и <b> получили новое семантическое значение и их нужно использовать ограниченно, только там, где это семантически оправданно.

необязательную завершающую точку с запятой, но, поскольку единственная пропущенная обязательная точка с запятой, круглая или фигурная скобка может привести к сбою всей таблицы стилей, я настоятельно рекомендую всегда включать в код завершающую необязательную точку с запятой. Влияние ее включения на объем передаваемых данных ничтожно по сравнению с тем временем, которое придется затратить на поиск и устранение в сбойной таблице CSS ошибок, вызванных пропущенным символом.

Стили CSS оказывают влияние на тот элемент, на который нацеливает селектор. Нацеливание на элементы может происходить обычным способом, по имени элемента, или с конкретной точностью путем определения элементов на основе их связей с другими элементами, их позиции в потоке документа, их атрибутов, значений их атрибутов, текущего состояния или через уникальные идентификаторы (ID). Все селекторы будут рассмотрены в следующем разделе.

Стили, подобные ранее показанным, могут находиться в трех местах: быть встроенными непосредственно в код страницы, помещаться во внедренные стили или во внешние таблицы стилей.

Стили могут быть объявлены как *встроенные*, в виде составной части открывающего тега `<p>` с использованием атрибута `style`:

```
<p style="color:blue; margin-bottom: 12px;">
```

Стили могут быть *внедренными* в заглавную часть документа:

```
<style>
  p {
    color: blue;
    margin-bottom: 12px;
  }
</style>
```

ПРИМЕЧАНИЕ

Учтите, что в приводимых здесь примерах тег стиля записывается как <style>, а не как <style type="text/css">. В HTML5 выражение type="text/css" подразумевается, и поэтому его можно опустить.

Стили могут и *должны* быть включены в виде *внешней таблицы стилей*, связанной с документом с помощью элемента `<link>`:

```
<link rel="stylesheet" src="styles.css"/>
```

Рекомендуется использовать внешние таблицы стилей. При их использовании все свои веб-страницы можно привязать к единой таблице стилей, гарантируя тем самым единый внешний вид всех страниц и сокращая объем обслуживания сайта за счет того, что изменения дизайна могут быть сделаны для всего сайта путем редактирования одного-единственного файла. Для параллельной загрузки стилей, ускоряющей загрузку сайта, нужно вместо директивы `@import` использовать элемент `<link>`.

При использовании внешних таблиц стилей необходимость загрузки стилей возникает только один раз — при начальном посещении пользователем любой страницы внутри сайта. Файл CSS, как правило, кэшируется клиентским браузером.

Поэтому байты CSS загружаются единожды. Кроме того, хранение стилевой информации в отдельном документе отделяет содержимое от его презентации. Отделение содержимого от презентации и от поведения является основным принципом веб-стандартов.

При всей настоятельности рекомендаций по использованию внешних таблиц стилей из-за проблем, связанных с задержками в мобильных сетях, нерекомендуемая (anti-pattern) схема включения внедренных стилей и хранения их в локальном хранилище имеет свои преимущества, связанные с сокращением количества HTTP-запросов. Эта тема уже обсуждалась в главе 2 при рассмотрении элемента `<style>`. При выборе между привязкой стилей и использованием нерекомендуемой схемы их внедрения в качестве составной части документа нужно убедиться в сокращении количества поисков и HTTP-запросов.

Использование внешних таблиц стилей: повторное обращение к <link>

Для включения внешних таблиц стилей используется элемент `<link>`. Тег `<link>` уже рассматривался в главе 2, поэтому я не хочу снова перечислять все его атрибуты и значения. Давайте лучше посмотрим на атрибуты и значения, имеющие отношение к CSS. При этом следует обратить особое внимание на атрибуты, встречающиеся в ссылках на внешние таблицы стилей:

```
<link type="text/css" rel="stylesheet" src="styles.css" media="all"/>
```

В XHTML обязательным был атрибут, определяющий MIME-тип в виде `type="text/css"`. Предположительно, тем самым браузер был информирован о том, что привязываемый файл является текстовым (а не файлом приложения или двоичным файлом) с CSS-разметкой. Как упоминалось ранее, применять этот атрибут в HTML5 не требуется, пока не используется что-либо отличное от CSS, чего вы, скорее всего, делать никогда не будете. Когда отношение (`relation` или `rel`) имеет значение `"stylesheet"`, предполагается, что тип (`type`) имеет значение `test/css`, если не указано иное, и значением по умолчанию для среды применения (`media`) является `"all"`:

```
<link rel="stylesheet" src="styles.css"/>
```

Не забудьте включать атрибут `rel`, записанный в виде пары «атрибут — значение» `rel="stylesheet"`. Без этого атрибута браузер не будет знать предназначения вашего файла и не станет его интерпретировать как CSS. Если произойдет сбой интерпретации CSS, убедитесь в том, что этот атрибут включен. Его отсутствие доставляет кучу неприятностей, причину которых трудно заметить, но легко устранить. Атрибут `rel` сообщает браузеру, какое отношение к данному файлу имеет привязываемый файл.

Если этот атрибут присутствует и имеет значение `"stylesheet"`, значение *типа* данных подразумевается и браузер знает, что содержимое файла нужно рассматривать как `text/css`.

Атрибут «источник» — src должен в качестве значения иметь URL-адрес, указывающий на внешнюю таблицу стилей, содержащую нужный код CSS.

Элемент <link> относится к пустым элементам. Если используется разметка в стиле XHTML, то он должен быть самозакрытым с помощью слеша перед закрывающей скобкой.

Атрибут media. Атрибут media, если он не указан, по умолчанию получает установку media="all", что означает предназначение для всех сред применения. Атрибут, который является частью CSS уже многие годы, может получать следующие основные значения:

- All — представляет все устройства, включая все перечисленные далее типы;
- braille — представляет только устройства с тактильной обратной связью, использующие систему Брайля;
- embossed — код предназначен для постраничных брайлевских принтеров;
- handheld — код предназначен для портативных устройств, обычно имеющих небольшой экран и ограниченную полосу пропускания. Следует заметить, что, хотя смартфоны и iPad тоже можно причислить к портативным устройствам, на них установлены полноценные браузеры и они относятся к средам screen и all, но никак не к среде handheld;
- print — представляет режимы большинства браузеров для вывода на печать, отображения PDF-документов и предварительного просмотра перед выводом на печать;
- projection — код предназначен для проекторов и других проецируемых презентаций;
- screen — код предназначен для цветных компьютерных экранов, включая экраны браузеров ноутбуков, настольных компьютеров и смартфонов, к которым причисляются такие устройства, как телефоны, планшеты и фаблеты;
- speech — код предназначен для речевых синтезаторов. Следует учесть, что в CSS 2 для этой цели имеется подобный тип среды применения под названием aural;
- tty — код предназначен для сред, использующих моноширинную символьную сетку (например, для телетайпов, терминалов или переносных устройств с ограниченными возможностями отображения данных);
- tv — код предназначен для устройств телевизионного типа, у которых есть звук, но нет возможности прокрутки данных.

Как отмечалось ранее, у смартфонов имеются полноценные браузеры, и поэтому на них реализуются привязанные таблицы стилей, имеющие установки атрибута media="screen", media="all" и не имеющие объявления среды применения, поскольку по умолчанию используется значение all.

Можно использовать единую таблицу стилей без объявления среды применения и нацеливать их на различные типы среды с помощью инструкции @media:

```
@media screen {
  p {
    color: blue;
  }
}

@media print {
  p {
    color: red;
  }
}
```

Медиазапросы

В CSS3 значения атрибута `media` не ограничиваются предыдущим списком. Медиазапросы позволяют нам нацелить CSS на устройство или на браузер на основе высоты, ширины, разрешения и ориентации окна браузера или устройства или — в случае использования масштабируемой векторной графики (SVG) — родительского контейнера. Если нужно использовать одну и ту же HTML-страницу, но разные таблицы стилей для браузеров смартфонов, планшетных компьютеров и настольных компьютеров, вы можете использовать атрибут `media` для указания того, какой именно CSS-файл должен быть исполнен на экранах разных размеров:

```
<link media="only screen and (max-device-width: 480px)"
  href="mobile.css" rel="stylesheet"/>
```

Наиболее часто используемые свойства, помогающие нацелить устройства и браузеры на стиль, показаны в табл. 7.1.

Таблица 7.1. Наиболее важные @media-свойства в сфере мобильных устройств

Свойство	Свойство минимум	Свойство максимум	Описание
width	min-width	max-width	Ширина окна просмотра
height	min-height	max-height	Высота окна просмотра
device-width	min-device-width	max-device-width	Ширина экрана
device-height	min-device-height	max-device-height	Высота экрана
orientation	—	—	Книжная (высота больше ширины). Альбомная (ширина больше высоты)
aspect-ratio	min-aspect-ratio	max-aspect-ratio	Соотношение ширины и высоты
device-aspect- ratio	min-device-aspect-ratio	max-device-aspect-ratio	Соотношение ширины и высоты, присущее устройству

Можно указать конкретное нацеливание устройства на стиль. Например, нацелить на стиль iPhone в книжном режиме можно с помощью следующей инструкции:

```
<link media="only screen and (width: 320px) and (orientation: portrait)"
  href="iphone.css" rel="stylesheet"/>
```

Но назвать это решение удачным нельзя. Мобильные устройства имеют различные формы и размеры. И вместо того, чтобы определять отдельные таблицы стилей для каждого возможного устройства и для каждых возможных ширины и высоты экрана в пикселах, следует создавать медиазапросы с диапазонами размеров, разбивая эти диапазоны там, где изменение макета может иметь смысл. Например, для того, кто пользуется планшетным компьютером, имеет смысл поместить расширенную панель навигации в верхней части, а на очень маленьких устройствах содержимое лучше расположить над панелью расширенной навигации, чтобы пользователю для просмотра важного содержимого не пришлось применять прокрутку.

Медиазапросы могут использоваться для предоставления различных значений CSS-свойств, основанных на размере и ориентации устройства и окна просмотра. Например, медиазапросы могут (и зачастую должны) использоваться для того, чтобы различные виды информации работали на экранах разных размеров. Изображение шириной 1400 пикселов нет смысла показывать на экране телефона[1] шириной 320 пикселов:

```
@media screen and (min-width: 440px) {
   #content { background-image: url(/images/small/bg.jpg);
}
@media screen and (min-width: 1000px) {
  #content { background-image: url(/images/large/bg.jpg);
}
```

Заметьте, что в этих `@media`-блоках используются два значения ширины, которые могут и не отражать стандартные показатели ширины устройств. При выборе в своих дизайнерских решениях и макетах контрольных точек не следует основываться на размерах популярных устройств. Нужно выбирать такие контрольные точки, которые имеют смысл с точки зрения дизайна вашего пользовательского интерфейса. Быстрее всего контрольные точки можно выбрать путем их тестирования в браузерах. Для этого нужно медленно увеличивать или уменьшать размер экрана настольного компьютера. Контрольную точку лучше всего выбрать там, где дизайн начинает терять привлекательность.

Рассмотрение правила `@media` и адаптивных свойств будет продолжено в главе 11.

Кроме нацеливания на основе размера и ориентации можно выполнять нацеливание также на основе поддержки браузером анимации, переходов и трехмерных преобразований:

---

[1] Если нужно выяснить ширину и высоту имеющегося окна просмотра с помощью JavaScript, то перед тем, как вмешаться в структуру макета, это можно сделать, выполнив следующий код:
width = window.innerWidth;
height = window.innerHeight;

```
@media screen and (transform-3d) {
  .transforms {}
}
```

Здесь свойство в круглых скобках может нуждаться в указании префикса производителя в тех браузерах, которые по-прежнему требуют использования такого префикса с упомянутыми тремя свойствами.

Со временем браузеры станут поддерживать `@supports`-правила:

```
@supports (display: table-cell) and (display: list-item) {
  .query .supports { display: block; }
}
```

Хотя запросы о поддержке (supports query) похожи на медиазапросы, они могут нацеливать CSS на различные устройства на основе поддержки браузерами тех или иных CSS-свойств. Но на момент написания данной книги это свойство поддерживалось некоторыми браузерами настольных устройств, но не поддерживалось браузерами мобильных устройств.

window.matchMedia. Объектная модель CSS, CSSOM предоставляет расширения оконного интерфейса. Расширение `window.matchMedia`[1], если оно поддерживается, возвращает новый объект списка медиазапросов — `MediaQueryList` (`mql`), представляющий собой результаты анализа строки указанного медиазапроса на предмет наличия соответствующего свойства:

```
var mql = window.matchMedia(mediaquery);
if (mql.matches) {
  //если имеется соответствие медиазапросу
}
```

Здесь `mediaquery` означает медиазапрос.

Например, можно протестировать окно просмотра на ширину, составляющую менее 500 пикселов:

```
var mqobj = window.matchMedia('(orientation: portrait)');
if (mqobj.matches) {
  document.querySelector('body').classList.add('portrait');
}

if (window.matchMedia("(max-width: 500px)").matches) {
  // окно просмотра имеет ширину, не превышающую 500 пикселов
} else {
  // окно просмотра имеет ширину, превышающую 500 пикселов
}
```

Также предоставляется способ отследить события изменения среды применения. Можно проверить, является ли текущая ориентация экрана книжной или альбомной, и отследить изменения:

---

[1] Начиная с iOS 5 и Android 3, window.matchMediais поддерживается всеми мобильными браузерами, за исключением IE.

```
var mqobj = window.matchMedia('(orientation: portrait)');
mqobj.addEventListener('orientationchange', bodyOrientationClass);

function bodyOrientationClass() {
  if (mqobj.matches) { // книжная ориентация
    document.querySelector('body').classList.remove('landscape');
    document.querySelector('body').classList.add('portrait');
  } else {
    document.querySelector('body').classList.remove('portrait');
    document.querySelector('body').classList.add('landscape');
  }
}
```

Сначала создается объект списка медиазапросов и включается метод отслеживания `addEventListener`, имеющийся в объекте списка медиазапросов, который в ответ на событие вызывает функцию. Я включила в код функцию, проверяющую на соответствие медиазапросу и выполняющую надлежащую обработку.

Удалить отслеживатель можно с помощью следующего кода:

```
myobj.removeEventListener('orientationchange', bodyOrientationClass);
```

Рекомендуемые методы использования CSS

Рассмотрим пять советов (или правил!), позволяющих создавать наиболее рациональный код CSS, которые следует взять на вооружение для обеспечения более высокого качества вашего сайта и повышения скорости его загрузки.

1. Сведите к минимуму количество HTTP-запросов. Для повышения скорости загрузки нужно минимизировать количество отдельных таблиц стилей, чтобы минимизировать количество HTTP-запросов. Издержки на HTTP-запросы могут быть слишком высокими. Сокращение количества запросов может существенно снизить время загрузки страниц.

HTTP-запросы с точки зрения загрузки зачастую являются самыми большими потребителями времени, особенно если дело касается мобильных сетей. В связи с этим предпочтительнее добавить одну большую таблицу стилей, чем несколько небольших, каждая из которых задает стиль какого-нибудь компонента вашего сайта.

Хотя, может быть, и выгоднее собирать свои стили в таблице стилей из группированных вместе блоков, задающих стили для каждого модуля вашего сайта, но для готового изделия лучше собрать все CSS-таблицы в один большой файл и вместо `style.css`, `home.css`, `about.css`, `footer.css`, `sidebar.css` и т. д. использовать один-единственный файл `all.css`.

Загрузка и кэширование одного крупного CSS-файла обычно лучше отражается на пользовательском восприятии, чем принуждение клиента загружать таблицы стилей для тех или иных страниц, даже если эти таблицы имеют меньший размер. Цена дополнительного HTTP-запроса чаще всего превышает цену наличия нескольких строк незадействуемого кода CSS. Кроме того, за счет использования одного CSS-файла все ваши стили для всего сайта кэшируются при загрузке со-

держимого первой страницы, при этом устраняются задержки на загрузку дополнительных CSS-файлов в том случае, когда пользователь путешествует по сайту.

Следует также отметить, что мобильная память имеет ограничения по объему в сравнении с памятью настольных устройств, поэтому увлекаться особенно длинными файлами не стоит. Хотя для упрощения и ускорения написания кода CSS мною рекомендованы такие препроцессоры серверной стороны, как Sass, если вы не ориентируетесь в происходящем, ваши CSS-файлы могут превысить необходимые размеры. Если же вы разбираетесь в том, что делаете, подобные средства могут помочь вам создать модульную структуру кода и минимизировать объем кода CSS, существенно сокращая количество необходимых байтов. Применяйте эти средства благоразумно.

2. Используйте внешние таблицы стилей. Используйте внешнюю таблицу стилей, ссылаясь на нее из раздела `<head>` своих файлов. Включение этой единственной внешней таблицы стилей дает следующие преимущества.

- Стили для всего сайта можно менять в одном месте.
- Пользователи загружают и кэшируют таблицу стилей только один раз при первом посещении сайта (и не нуждаются в ее повторной загрузке при посещении второй, третьей и четвертой страниц).
- Пользователям нужно лишь единожды загрузить вашу таблицу стилей, экономя на HTTP-запросах при повторных посещениях страницы.
- Сохраняется отделение содержимого от презентации.

Хотя правило 1 (минимизировать количество HTTP-запросов) может навести на мысль, что лучше будет внедрить код CSS и сэкономить HTTP-запрос, но цена одного HTTP-запроса обычно полностью покрывается преимуществами, получаемыми от внешних таблиц стилей (хотя, как говорилось в главе 2, существуют исключения, вынуждающие прибегать к нерекомендуемым решениям). Браузер может кэшировать внешнюю таблицу, на которую ссылаются все ваши страницы, поэтому она должна быть загружена только один раз. А внедренные стили загружаются с каждой страницей.

Хотя цена времени загрузки, связанная с одним HTTP-запросом, как правило, меньше цены, связанной с загрузкой байтов внедренного в страницу кода CSS, при загрузке второй, третьей и четвертой страниц вашего сайта HTTP-запросы через сети 3G могут проходить с большой задержкой. Для сайтов, доступных через ограниченные сети, дополнительный HTTP-запрос на внешнюю таблицу стилей стоит того. Но для мобильных устройств это не всегда актуально.

Как уже говорилось в главе 2, некоторые мобильные сайты используют нерекомендуемые решения. Они внедряют CSS и даже JavaScript в первый серверный ответ. Затем они используют JavaScript для извлечения внедренных сценариев и помещения их в локальное хранилище с использованием API-функций объекта `LocalStorage`. При снабжении каждого сценария уникальным идентификатором можно хранить и извлекать сценарий, а также ссылаться на него с использованием этого уникального ключа. Идентификатор сценария также добавляется в cookie-файл. При загрузке последующих страниц, как и при использовании всех HTTP-запросов, cookie-файлы отправляются вместе с HTTP-запросом, информируя сервер, какие сценарии уже есть

у сервера, и позволяя серверу решать, какие сценарии, если таковые имеются, отправлять в ответ на последующие запросы. Эта нерекомендуемая практика может привести к очень долгой загрузке первой страницы и чуть более коротким последующим ответам на запросы. Хотя у объекта `LocalStorage` имеются некоторые недостатки, например время, затрачиваемое устройством на обращение к данным `LocalStorage`, это хранилище может стать действенным инструментом в минимизации количества HTTP-запросов, что будет способствовать решению проблем задержек в мобильных устройствах. Хранилище `LocalStorage` рассматривается в главе 6.

3. Выполняйте нормализацию браузеров путем сброса CSS или использования нормализатора. Браузеры поставляются с собственной таблицей стилей, которая называется таблицей стилей агента пользователя — user agent (UA) stylesheet. В этой принадлежащей браузеру таблице стилей имеются исходные настройки стилей, например курсив для содержимого тега `<em>`, применение полужирного шрифта и установка размеров шрифтов для содержимого тегов `<h1>` — `<h6>`, отступы и маркеры для содержимого тегов `<li>`. К сожалению, не все браузеры и не все версии браузеров поставляются с одними и теми же UA-таблицами. Например, поля абзацев и тела документа от браузера к браузеру изменяются. Поэтому сначала рекомендуется сбросить[1] или нормализовать CSS-файл, чтобы установить для всех браузеров единое поведение: удалить или нормализовать многие из исходных стилей, чтобы унифицировать стили для всех браузеров.

Для нормализации нужно начать свою таблицу с установки основных стилей низкой конкретизации, чтобы убрать различия между браузерами в UA-таблицах. Установка исходных настроек позволяет устранить несовместимость как имеющихся на данный момент, так и будущих браузеров.

Но даже если вы выполняете разработку только под один браузер, например под одну версию WebKit (чего я, конечно же, не рекомендую делать), я все же настаиваю на использовании сброса CSS или применении нормализатора[2]. В применяемый мною сброс в отношении большинства элементов я включаю такие объявления, как `margin: 0;`, `padding: 0;` и `background-repeat: no-repeat;`. Включая эти три строки кода в разметку в моем коде сброса, я экономлю сотни строк кода за счет исключения необходимости повторять какую-либо из них.

ПРИМЕЧАНИЕ

Использовать в коде сброса символ * нужно осмотрительно, поскольку удалять исходные установки стилей для некоторых элементов, например для полей формы, вам вряд ли захочется. Кроме того, его использование увеличивает расход памяти и время выполнения[3].

---

[1] Yahoo! предоставляет великолепный файл сброса CSS по адресу http://developer.yahoo.com/yui/reset/. Добавьте к его содержимому строку background repeat: no-repeat;, и все будет в порядке.

[2] Normalize.css — небольшой CSS-файл, предоставляемый для установки межбраузерного соответствия исходных стилей HTML-элементов (вместо сброса). Он создан Джонатаном Нилом и Николасом Галлахером и нацелен только на те стили, которые требуют нормализации.

[3] В документации по исходному коду HTML5BoilerPlate на сайте GitHub можно найти массу полезных советов.

4. Чтобы упростить внесение изменений, используйте как можно более слабую конкретизацию. Рекомендуется также использовать элементы и классы, а не идентификаторы (ID). Снижение степени конкретизации сокращает количество селекторов, необходимых для перезаписи правила. Чем слабее конкретизация, особенно при создании собственного кода сброса или исходного шаблона, тем проще будет переписать значение для одного из стилей. Нужно начинать с определения стилей с помощью селекторов основных HTML-тегов.

Избегайте использования идентификаторов, поскольку они имеют более весомое значение в каскаде. Хотя ID-селекторы могут выполняться немного быстрее, экономия времени получается незначительной. Использование идентификаторов ограничивает нацеленность CSS-правила одной областью подмножества, и перезапись стилей, указанных с помощью ID-селекторов, требует более строгой конкретизации идентификаторов. Поэтому, несмотря на небольшие потери в производительности, использование меньшей конкретизации нужно в селекторе для создания более общих правил и лучше подходит для перезаписи значения свойства.

Затем можно создавать конкретные стили для перезаписи исходных установок.

Например, на абзац:

```
<p id="myP" class="blue">Это
  <strong class="warning">важно</strong></p>
```

можно нацелиться с помощью кода:

```
body p#myP.blue strong.warning {
  color: red;
}
```

или просто с помощью кода:

```
.warning{
  color:red;
}
```

В CSS-таблицах часто встречаются селекторы, подобные первому из показанных, но лучше всего использовать второй.

При этом не только экономятся байты кода, но и упрощаются написание, обслуживание и, что важнее, изменение кода. Представим, что дизайнер высказывает еще одно уточнение: «Если этот абзац находится на боковой панели, нужно, чтобы в красном цвете было больше зеленовато-желтых составляющих». Возникают две проблемы: во-первых, я не понимаю, что такое зеленовато-желтые составляющие (chartreuse). Но здесь хоть можно заглянуть в Google. Главной же проблемой станет то, что для изменения цвета придется конкретизировать цель! Либо так:

```
body aside.sidebar p#myP.blue strong.warning {
  color: #7FFF00;
}
```

Либо, если с самого начала код был с меньшей конкретизацией, так:

```
aside .warning{
  color: #7FFF00;
}
```

Если используется Sass или другой компилятор, то могут обнаружиться селекторы, имеющие глубину в 10 правил. Я же обычно не создаю селекторы глубже трех правил, что мне представляется весьма хорошим балансом между производительностью, конкретизацией и простотой обслуживания.

Дополнительные сведения о конкретизации CSS даны в приложении и разделе «Другие селекторы: теневая DOM-модель» данной главы.

5. Не используйте встроенные стили или модификатор !important. Здесь можно обойтись и без объяснений, поскольку пользоваться встроенными стилями и ключевым обозначением `!important` (важное) действительно не стоит. Не пользуйтесь ими (кроме как для создания прототипов).

Если же требуется объяснение, то вот оно: ключевое слово `!important` отменяет каскадность для свойства, в которое это объявление включено. Значение свойства для селектора со слабой конкретизацией, который включает в объявление модификатор `!important`, превалирует над всеми остальными значениями того же самого свойства и не может быть переписано. Например:

```
p {color: green !important;}
p#myP {color: blue;}

<p style="color: red" id="myP">
```

В данном случае текст абзаца все равно будет зеленым, потому что использовался модификатор `!important`.

Модификатор `!important` был добавлен к CSS, чтобы позволить пользователям изменять авторские веб-стили. Но какие бы намерения и цели ни преследовались, ключевое обозначение `!important` должны применять ваши опытные пользователи, но никак не вы как разработчик.

Лично я использую `!important` только для отладки. Модификатор `!important` добавляю временно, чтобы увидеть, нацеливается ли вообще мой селектор на мой элемент. Если добавление `!important` не меняет внешний вид элемента так, как это предполагалось, стало быть, я допустила опечатку в селекторе или же `<span>`-элемент был `<a>`-ссылкой или какой-нибудь похожей ошибкой.

ПРИМЕЧАНИЕ

Неважно, насколько строга конкретизация даже с добавлением !important, переписать указание модификатора !important в свойстве, объявленном в UA-таблице стилей, невозможно. В большинстве UA-таблицах стилей их немного, но те значения свойств, которые их включают, не могут быть переписаны вашими собственными стилями.

Перечисляемые здесь советы исходят из практического опыта и упрощают написание CSS. При создании легких в обслуживании таблиц стилей можно исполь-

зовать и другой положительный опыт, например группировку селекторов по разделам, добавление комментариев для облегчения последующего понимания кода и установку отступов для улучшения читаемости, но мы не будем углубляться в приемы достижения наилучшего восприятия кода[1], поскольку, как говорится, на вкус и цвет товарищей нет.

Наилучшим будет совет накапливать собственный положительный опыт и придерживаться его в дальнейшем. Например, мне все равно, что вы будете использовать для отступов, пробелы или табуляцию, но, что бы вы ни выбрали, не меняйте свой выбор — будьте постоянными!

Селекторы CSS

Если вы уже знакомы с CSS, переходите к разделу «Дополнительные селекторы CSS3». Если же вы не уверены в своих знаниях, давайте рассмотрим основные положения. И даже если вы считаете себя профессионалом, не пропускайте раздел «Дополнительные селекторы CSS3». Возможно, рассмотренное там повышение эффективности селекторов CSS станет для вас сюрпризом.

Селекторы являются CSS-комбинациями, используемыми для определения того, какие правила стилей к каким элементам дерева документа применить. Они начинаются с простых селекторов типов элементов и доходят до контекстных комбинаций, нацеливающих на DOM-узлы на основе атрибутов, исходного порядка или отношений в семействе дерева элементов. Если все условия комбинации для какого-то элемента или псевдоэлемента выполняются, то указанные правила применяются к этому элементу.

ПРИМЕЧАНИЕ

Все мобильные браузеры поддерживают CSS 2.1 и селекторы CSS3, рассматриваемые в данной главе, то же самое касается и всех браузеров настольных компьютеров, исключая IE8 и более ранние версии этого браузера.

Имеющие некоторый опыт работы с CSS, наверное, знают, как нацеливаться на элементы, используя селекторы элементов, классов и идентификаторов (ID) или их комбинации. Наиболее часто используются селекторы типа, класса и идентификатора — основные CSS-селекторы, предоставляемые исходными версиями CSS.

Вам предстоит узнать, что при использовании CSS3 можно задавать невероятную точность прицеливания практически в любом узле вашего документа, как правило, даже без необходимости добавления класса или идентификатора. Но сначала нужно убедиться в полном понимании построения блоков CSS.

[1] Оптимизации CSS с целью повышения производительности посвящена вся глава 9 книги Advanced CSS, написанной Московичем и Левисом и выпущенной издательством Friends of Ed.

Селектор типа

Селектор типа, или селектор элемента, будет нацеливать на все элементы конкретного типа:

```
a {
  color: blue;
}
p {
  color: pink;
}
strong {
  color: green;
}
```

Этот код CSS требует, чтобы ссылки имели синий цвет, абзацы — розовый, а сильно выделенный текст — зеленый.

```
<p>Это абзац с
  <a href="..."><strong>выделенной </strong> ссылкой</a>
</p>
```

В этом примере благодаря вложенности слово «выделенной» будет зеленым, слово «ссылка» — синим, а весь остальной абзац будет розовым, если только какой-либо из этих элементов не унаследует объявление CSS, являющееся более конкретным и меняющим цвета.

В группе селекторов можно объявить *несколько элементов*, отделяя их друг от друга запятой и создавая тем самым список селекторов типа:

```
p,li {
  text-transform: uppercase;
}
```

Элементы-потомки объявляются путем разделения их пробелом:

```
p strong {
  color: pink;
}

li a {
  color: black;
}
```

Если бы я включил этот CSS, то текст в этом абзаце, а также текст в любых других абзацах и элементах списка был бы показан в верхнем регистре, а слово «выделенной» было бы розовым, поскольку элемент `strong` является потомком элемента `<p>`. Но слово «ссылка» по-прежнему оставалось бы синим, а не черным, поскольку тег `<a>` находится внутри тега `<p>` и не имеет в качестве предка тега `<li>`.

ВНИМАНИЕ

Старые версии Internet Explorer не поддерживают селектор типа в отношении незнакомых им элементов, включая все новые элементы HTML5. Все мобильные браузеры, которые вы, возможно, попытаетесь нацелить, будут выводить нераспознаваемые ими элементы и понимать все селекторы CSS3.

Селектор класса

Селектор класса будет нацеливать на все элементы с конкретным классом, в названии которого учитывается регистр символов:

```
.copyright {
  font-size: smaller;
}
.urgent {
  font-weight: bold;
  color: red;
}
<p class="copyright">Это абзац с
  <a href="..."><strong class="urgent">выделенной</strong> ссылкой</a>
</p>
```

С добавленными классами весь абзац будет выведен более мелким шрифтом, а слово «выделенной» будет выведено полужирным шрифтом красного, а не зеленого цвета. Причина смены цвета с зеленого на красный в том, что селектор класса сильнее или конкретнее в понятиях каскадности.

Для элемента HTML можно использовать более одного класса: нужно просто в значении атрибута `class` отделить имена классов друг от друга пробелами. Следует учесть, что порядок задания классов в атрибуте `class` любого элемента неважен: приоритет определяется порядком следования классов в таблице стилей. Следующие две строки по смыслу равны друг другу:

```
<p class="class1 class2">некий текст</p>
<p class="class2 class1">некий текст</p>
```

В понятиях каскадности отдельный селектор класса имеет больший вес, чем любое количество селекторов типа.

Мы могли бы написать:

```
p.copyright {
  font-size: xx-small;
}
```

Затем, поскольку между `p` и указанием класса пробел отсутствует, размер шрифта `xx-small` получат только те абзацы, у которых есть класс `copyright`. Содержимое тега `<li class="copyright">` будет выведено более мелким шрифтом, но не шрифтом размера `xx-small`, поскольку на него нацелен общий селектор `.copyright`, а не более конкретизированный селектор `p.copyright`.

В разделе, посвященном рациональному использованию CSS, рекомендовалось использовать как можно меньшую конкретизацию при нацеливании на элемент. В данном случае рекомендуется вместо `p.copyright` использовать `.copyright`. Более короткий селектор менее конкретен, потенциально нацелен на большее количество элементов — на все элементы с классом `.copyright`. Пока не предпринимается попытка переписать стиль именно тех абзацев, у которых имеется этот класс, нужно использовать селектор с меньшей конкретизацией.

ВНИМАНИЕ

Имена классов чувствительны к регистру символов: copyright не является эквивалентом Copyright или copyRight.

Селектор идентификатора

Селектор идентификатора будет нацеливать на отдельный элемент документа, имеющий именно такой идентификатор (ID). Следует помнить, что идентификаторы чувствительны к регистру символов и должны иметь для документа уникальный характер:

```
#divitis {
  color: orange;
  font-size: larger;
}
<p class="copyright" id="divitis">Это абзац с
  <a href=""><strong class="urgent">выделенной</strong> ссылкой</a>
</p>
```

В данном примере абзац будет выведен оранжевым и более крупным, а не более мелким шрифтом, поскольку селекторы идентификаторов имеют более конкретный характер по сравнению с селекторами классов, и поэтому имеющиеся в них правила переписывают правила, имеющиеся в селекторах `.copyright` и `p.copyright`.

Селектор `#divitis a{}` конкретнее селектора `.copyright a{}`, который, в свою очередь, конкретнее селектора `p a{}`, а тот конкретнее селектора `a {}`. Чтобы представить себе конкретизацию значений селекторов типа, класса и идентификатора, посмотрите данные табл. 7.2 и ознакомьтесь с разделом «Другие селекторы: теневая DOM-модель» данной главы.

В понятиях каскадности отдельный селектор идентификатора имеет больший вес или конкретизацию, чем любое количество селекторов класса или селекторов типа, поэтому ими нужно пользоваться крайне редко или не пользоваться вообще. Уменьшить их конкретизацию весьма непросто. Для этого нужно создать еще более детализированное правило, использующее такое же или большее количество селекторов идентификатора.

ВНИМАНИЕ

Селекторы класса и селекторы идентификатора, в отличие от селектора типа, чувствительны к регистру символов.

Дополнительные селекторы CSS3

Следующий материал будет полезен даже тем, кто уже знаком с CSS: в нем содержится множество таких тонкостей, о которых вы не знали или на которые просто не обращали внимания.

Селекторы соответствуют тому элементу, к которому будут применены объявления CSS. Во времена использования CSS 2 и поддержки таблиц на настольных

компьютерах понятие элемента ограничивалось селекторами типа, класса и идентификатора, небольшим количеством связанных со ссылками псевдоклассов и, возможно, имеющимся, хотя и не обязательно четким представлением об универсальном селекторе, задаваемым символом звездочки (*).

Давайте при нацеливании на элементы в нашей таблице стилей прекратим думать о типе элемента, сфокусировавшись вместо этого на модели документа. CSS3 дает возможность упростить нацеливание стилей на основе позиции элемента в документе и его отношения к другим элементам и даже на основе атрибутов элемента и состояния пользовательского интерфейса.

Спецификация CSS3 существенно расширила наши горизонты и возможности точного нацеливания с помощью новых селекторов атрибутов, структурных селекторов, псевдоклассов и комбинаторов (рассматриваемых в следующем разделе) наряду с некоторыми селекторами атрибутов, имевшимися в CSS 2, но Internet Explorer стал полностью поддерживать селекторы CSS 2.1, только начиная с версии IE8. Браузеры Opera, Chrome, Safari, Firefox и Internet Explorer (начиная с версии IE9) поддерживают все селекторы CSS 2.1 и CSS3 с одной оговоркой: из соображений безопасности некоторые браузеры прекратили полную поддержку псевдоклассов `:link` и `:visited` элемента `<a>`.

ПРИМЕЧАНИЕ

Все браузеры мобильных (и настольных) систем, включая Opera, Chrome, Safari, Firefox и Internet Explorer, начиная с версии IE9, поддерживают все селекторы CSS 2.1 и CSS3.

Основные селекторы

Основные селекторы, включая универсальный селектор, селекторы типа, класса и идентификатора, вошли в обиход еще в прошлом тысячелетии.

Универсальный селектор: *

Этот селектор был добавлен в CSS 2 и соответствует любому элементу на странице. В его синтаксисе применяется символ звездочки (*):

```
* {
  color: blue;
}
footer * {
  color: white;
}
```

Отдельно указанный универсальный селектор относится к любому элементу, от корневого элемента документа до последнего дочернего элемента. Вместо использования данного селектора в качестве глобального лучше сузить его область действия с помощью комбинатора: нацелить его на все элементы, имеющие какого-нибудь известного предка, например на все элементы, содержащиеся в контейнере `<footer>`. А вот нацеливания на весь документ нужно избегать, поскольку непредвиденным последствием этого может стать удаление исходной стилизации полей формы.

Селектор типа: E

Селектор элемента, или селектор типа, соответствует всем элементам данного типа. Для представления того или иного элемента в селекторе нужно указывать в таблицах стилей только имя тега без каких-либо специальных символов:

```
section, aside, p {
  color: red
}
```

Селектор класса: .class

Ранее рассмотренный селектор класса соответствует любому элементу конкретного класса с учетом регистра символов в названии этого класса. В таблице стилей указывается название класса, предваряемое символом точки (.):

```
.myClass {
  color: green;
}
```

Селектор идентификатора: #id

Селектор идентификатора соответствует любому элементу, имеющему именно такой идентификатор, с учетом регистра символов в его названии. Селектор идентификатора имеет наивысшую среди всех типов селекторов степень конкретизации (рассматриваемую в следующем разделе). В таблице стилей указывается название идентификатора, предваряемое символом решетки (#):

```
#myId {
  color: black;
}
```

Использование селекторов

У элемента может быть более одного класса, но только один идентификатор. Включенный в элемент идентификатор должен быть уникален для документа:

```
<p class="firstclass secondclass" id="myparagraph">
```

На этот элемент абзаца можно нацеливаться несколькими способами в порядке конкретизации, показанной в табл. 7.2.

Таблица 7.2. Комбинации селекторов, нацеленных на абзац, в порядке увеличения конкретизации

Селектор	Объяснение	Конкретизация
p	Все абзацы	0-0-1
.firstclass .secondclass	Все элементы, имеющие данный класс	0-1-0
p.firstclass p.secondclass	Все абзацы, имеющие данный класс	0-1-1
.firstclass.secondclass	Любые элементы, имеющие оба класса	0-2-0

Селектор	Объяснение	Конкретизация
p.firstclass.secondclass	Все абзацы, имеющие оба класса	0-2-1
#myparagraph	Уникальный элемент, имеющий данный идентификатор	1-0-0
p#myparagraph	Элемент с этим идентификатором, если это абзац. В противном случае селектор не будет иметь никаких соответствий	1-0-1
p#myparagraph. firstclass p#myparagraph. secondclass	Абзац с этим идентификатором, если у него имеется данный класс. В противном случае селектор не соответствует элементу с таким идентификатором	1-1-1
p#myparagraph.first class.secondclass	Абзац с этим идентификатором, если у него имеются оба этих класса. В противном случае селектор не соответствует элементу <p> с таким идентификатором	1-2-1

Используйте как можно меньшую степень конкретизации! Если можно нацелиться на элемент с использованием только селектора типа, используйте только селектор типа. Если нужно использовать класс, используйте по возможности только один класс. Менее конкретизированные селекторы нацеливаются на большее количество элементов, что обеспечивает единообразие сайта, упрощение внесения изменений путем использования каскадности CSS и/или минимальное увеличение конкретизации.

При допустимости применения всех селекторов, перечисленных в табл. 7.2, последние несколько селекторов должны использоваться как можно реже или не использоваться вообще. Из-за высокой степени конкретизации я стараюсь не применять селекторы идентификатора, включая атрибут `id`, и использую их, как правило, только в метке, чтобы нацеливать на нее с помощью кода JavaScript и превращать ее в указатель. Если вы начнете придерживаться принципа наименьшей конкретизации, последние пять-шесть селекторов, показанные в табл. 7.2, вам никогда не понадобятся.

Реляционные селекторы: правила, основанные на порядке следования кода

Рассмотренная ранее классификация базировалась на типе, классе и идентификаторе одного элемента. Спецификация CSS позволяет производить нацеливание также на основе отношений элементов к другим элементам.

В предыдущих селекторах при обсуждении и изучении каскадности мы, как правило, использовали только один элемент. Но в реальности страница не состоит лишь из одного элемента. Все элементы по отношению к другим элементам являются либо родительскими, либо дочерними (за исключением корневых элементов и текстовых узлов, но они являются соответственно родительскими и дочерними). Большинство элементов одновременно являются и теми и другими. У многих из них в разметке будут родительские, дочерние и смежные элементы того же уровня. Фактически множество элементов составлено из предков, потомков и смежных элементов.

Чтобы помочь нам нацелиться на элементы на основе их взаимоотношений, CSS предоставляет несколько реляционных селекторов. Для наших примеров будет использоваться следующий код:

```
<div id="myParent">
  <p class="copyright" id="divitis">Это абзац с <a href="...">
    <strong class="urgent">сильно выделенной</strong> ссылкой</a></p>
  <p class="second classitis">А это еще один абзац с
    <a href="..."><em>выделенной</em> ссылкой</a></p>
</div>
```

После прочтения данной главы обнаружится, что вы можете удалить из этого фрагмента кода все классы и идентификаторы, не утратив возможности стилизации каждого элемента по отдельности.

Комбинатор потомков: E F

Комбинатор потомков условно записывается как `E F` (с одним или несколькими пробелами), где селектор для элемента `F` является дочерним или более отдаленным потомком элемента `E`. В нашем предыдущем примере:

- `p strong {}` выполняет нацеливание на элемент `<strong>`, являющийся потомком элемента абзаца, даже если он не является непосредственным дочерним элементом;
- `#myParent a{}` выполняет нацеливание на обе ссылки в тегах `<a>`. Не являясь непосредственными дочерними элементами, они все же являются потомками элемента с `id="myParent"`;
- `.copyright .urgent {}` выполняет нацеливание на элементы с классом `urgent`, которые являются потомками элемента с классом `copyright`;
- `li strong {}` не нацеливает на наш текст, поскольку в примере элемент `<strong>` не является потомком элемента списка.

Вопрос: Можете определить разницу между следующими двумя селекторами?

```
#myparagraph.myclass { }
#myparagraph .myclass { }
```

Ответ: У первого из них нет пробела между `#myparagraph` и `.myclass`, а у второго такой пробел есть.

Если перевести на обычный язык, первый читается так: «Нужно выбрать элемент, имеющий идентификатор `myparagraph`, а также имеющий класс по имени `myclass`».

А второй селектор читается так: «Нужно выбрать все элементы с классом по имени `myclass`, являющиеся потомками элемента с идентификатором `myparagraph`».

Следует помнить, что пробелы и знаки пунктуации играют в CSS весьма важную роль!

Комбинатор дочерних элементов: E > F

Комбинатор дочерних элементов условно записывается как `E > F`. Это селектор элемента `F`, который является для `E` непосредственным дочерним элементом. Он отличается от ранее рассмотренного более общего комбинатора потомков `E F`, который

позволяет элементу F быть не только дочерним, но и более отдаленным по прямому родству элементом (относиться к внукам и правнукам). Комбинатор дочерних элементов с символом «больше» (>) требует, чтобы дочерний элемент F был *непосредственным* дочерним элементом родительского элемента E:

- `div > p {}` соответствует обоим абзацам, поскольку оба абзаца являются непосредственными дочерними элементами родительского элемента ;
- `p > strong {}` не соответствует ничему в представленном примере, поскольку является непосредственным дочерним элементом для <a>, а не для <p>.

Хотя селекторы дочерних элементов уже давно поддерживаются всеми мобильными браузерами, ими из-за отсутствия поддержки на действительно устаревших браузерах настольных систем пользовались очень немногие разработчики. Все современные браузеры мобильных устройств поддерживают все селекторы CSS3, поэтому пора отбросить сомнения и начать их использовать! Следует учесть, что при всем удобстве использования символа > для большей конкретизации при нацеливании на нужный элемент комбинация `E F` имеет при конкретизации такой же вес, как и комбинация `E > F`.

Комбинатор дочерних элементов использовался нами в CubeeDoo. По мере выхода на более высокие уровни размер карт уменьшается. Чтобы заставить -элементы, являющиеся непосредственными дочерними элементами `#board` и никаких других узлов, изменять высоту на основе класса панели, можно нацелить их непосредственным образом, не применяя HTML для добавления классов:

```
#board > div {
  position: relative;
  width:23%;
  height:23%;
  margin: 1%;
  float: left;
  transform-style: preserve-3d;
  transition: 0.25s;
  box-shadow: 1px 1px 1px rgba(0,0,0,0.25);
  cursor: pointer; /* for desktop */
}
#board.level2 > div {
  height: 19%;
}
#board.level3 > div {
  height: 15%;
}
```

Эти значения свойства будут воздействовать на контейнеры карт, но не на их лицевую или тыльную части. Нам нужно определить свойства CSS для контейнера карт, но не нужно, чтобы эти свойства применялись к лицевой и тыльной частям карты: карта должна иметь ширину 23% от ширины панели, являющейся окном просмотра. И нам не нужно, чтобы ширина лицевой часть карты составляла 23% от ширины отдельной карты.

Но у нас все объявлено в виде -контейнеров. Тогда как мы можем нацелиться на -контейнер не затрагивая те -контейнеры, которые являются его потомками? Путем использования комбинатора дочернего элемента > для указания конкретного отношения. Комбинация #board > div соответствует только тем -узлам, которые являются непосредственными дочерними узлами для панели, определяемой селектором #board. CSS-пары «свойство — значение» не будут добавлены к более отдаленным прямым потомкам, хотя некоторые свойства, такие как цвет, могут быть унаследованы.

Высота всех карт составляет 23 %, потому что на уровне 1 имеется четыре строки карт. Но когда у панели, определяемой селектором #board, имеется класс level2 или level3, значение высоты переписывается на 19 и 15 % соответственно, поскольку тогда у игры имеются соответственно четыре и пять строк. Если бы мы не включили комбинатор дочернего элемента, объявление высоты было бы нацелено также на лицевую и тыльную части карт. Высота этих вложенных -контейнеров составляла бы 23, 19 или 15 % от высоты их родительского -контейнера, а это не позволило бы понять, карта это или что-то совсем крошечное.

Селектор дочернего элемента также использовался для нацеливания на элемент <footer>, являющийся непосредственным дочерним элементом для элемента <article>, что дало возможность задать стиль основного колонтитула и потомков этого колонтитула, не нацеливаясь заодно и на колонтитулы разделов:

```
article > footer,
article > footer ul {
  text-align: center;
  margin: auto;
  width: 100%;
}
article > footer li {
  float: none;
  display: inline-block;
}
article > footer p {
  clear: both;
}
```

Комбинатор смежного элемента того же уровня: E + F

Комбинатор смежного элемента того же уровня нацеливается на второй элемент в селекторе, если оба элемента разделены символом «плюс» (+), *имеют общий родительский элемент* и целевой элемент (F) появляется в разметке *сразу же* после первого элемента селектора (E).

p:first-of-type + p {} выполняет нацеливание исключительно на второй абзац, поскольку оба абзаца являются непосредственными дочерними элементами для , но только если второй абзац следует сразу же после первого и между ними нет никаких других элементов.

Общий сборный комбинатор элементов одного уровня: E ~ F

Общий сборный комбинатор элементов одного уровня похож на комбинатор смежных элементов одного уровня тем, что нацеливается на второй элемент селектора, если оба элемента разделены символом «тильда» (~) и имеют общий родительский элемент. Но, в отличие от комбинатора смежных элементов, здесь целевой элемент `F` не должен появляться в разметке сразу же после элемента `E`. Он должен быть любым элементом `F` того же уровня, который появляется после первого элемента `E`. В отличие от комбинатора смежного элемента того же уровня, здесь общий сборный комбинатор может соответствовать более чем одному узлу.

Краткая справка по реляционным селекторам с предоставлением ряда примеров дана в табл. 7.3.

Таблица 7.3. Определения и примеры реляционных селекторов

Название	Синтаксис	Пример	Краткое объяснение примера
Комбинатор потомков	E F	.content p	Любой абзац, являющийся потомком (дочерний и более отдаленного прямого родства) элемента с классом content
Комбинатор дочерних элементов	E > F	ul.main > li	Элемент списка, являющийся непосредственным дочерним элементом ненумерованного списка с классом main. Не будет нацелен на вложенные элементы списка, касаясь только непосредственных дочерних узлов DOM родительского узла , имеющего класс main
Комбинатор смежного элемента того же уровня	E + F	h1 + p	Любой абзац, у которого общий родительский элемент с h1 и который следует в разметке непосредственно после h1
Общий сборный комбинатор элементов одного уровня	E ~ F	h1 ~ p	Любой абзац, у которого общий родительский элемент с h1 и который появляется в разметке после h1

Селекторы атрибутов

В главе 2 рассматривалось добавление атрибутов к элементам. Сильной стороной CSS является то, что вы можете использовать CSS-селекторы для нацеливания на элементы на основе этих атрибутов и даже их значений. Ряд весьма полезных селекторов атрибутов предоставлялся в CSS 2. А в CSS3 был добавлен ряд дополнительных селекторов атрибутов, позволяющих нацеливаться на соответствие подстроке значения атрибута.

Селектор атрибутов CSS 2 включает целевые элементы, у которых имеется конкретный атрибут с любым значением, атрибуты, имеющие конкретное значение, атрибуты, чьи значения содержат конкретное, отделенное пробелом слово, и атрибуты языка.

Обратите внимание на повторяющееся употребление слова «конкретный» в предыдущем абзаце. CSS позволяет очень точно нацелиться на конкретный элемент на основе его атрибутов.

Для ознакомления с селектором атрибутов спецификации CSS 2.1 мы зададим соответствие двум строкам кода HTML с помощью четырех типов селекторов атрибутов, представленных в табл. 7.4:

```
<ul>
  <li><a href="http://x.com/selectors.pdf" hreflang="en-us" rel="nofollow"
    title="CSS-селекторы и поддержка браузером">Специфика</a></li>
  <li>
    <input type="checkbox" name="spec" id="spec" value="web workers rock"/>
    <label for="spec">Есть в спецификации веб-работники?</label>
  </li>
</ul>
```

Таблица 7.4. Селектор атрибутов CSS 2 позволяет задавать соответствие по наличию атрибута, по значению атрибута, по атрибуту языка и по наличию в значении атрибута слов, разделенных пробелами

Селектор	Пример	Описание
E[attr]	a[rel] input[type] label[for]	Соответствует при наличии атрибута с любым значением
E[attr=val]	a[rel="nofollow"] input[type="checkbox"]	Соответствует при наличии атрибута с конкретным значением val[1]
E[attr|=val]	a[hreflang|="en"]	Соответствует при наличии значения, в точности равного значению val или начинающегося со значения val, после которого тут же следует символ –
E[attr~=val]	a[title~="browser"] input[value~="workers"]	Соответствует любому отделенному пробелами полному слову в значении атрибута

С внедрением в практику CSS 2.1 появилась возможность задавать соответствие селекторов на основе простого наличия атрибута, наличия атрибута с конкретным значением, наличия атрибута, значение которого содержит конкретное, отделенное пробелами слово, и наличия атрибута с конкретным значением или начинающегося с конкретного значения, за которым следует дефис.

Учтите, что кавычки в значении атрибута в этих примерах ставить не обязательно. Они понадобятся только в том случае, когда в значении селектора атрибута будет пробел. Я считаю, что их лучше ставить всегда, поскольку иногда в них возникает потребность, поэтому будьте последовательны и заключайте свои атрибуты в кавычки. Также учтите, что значение атрибута в селекторе атрибутов чувствительно к регистру символов в том случае, если значение атрибута в HTML также было чувствительно к регистру символов.

Селектор атрибута субкода языка является малоизвестным и редко используемым свойством CSS. Оно полезно при добавлении каких-нибудь визуальных знаков, например при добавлении в качестве фонового изображения флагов, симво-

[1] Чувствительность к регистру символов зависит от языка и от чувствительности к регистру символов значения атрибута. Кавычки нужны в том случае, когда значение атрибута в элемента должно быть в кавычках.

лизирующих конкретный язык, или при выделении курсивом содержимого на иностранном языке.

Используя такой селектор атрибутов, как `a[hreflang|=fr]`, мы можем нацелиться на соответствующие ссылки и снабдить их небольшим французским флагом, показывающим, что ссылка ведет на страницу, написанную по-французски. Простой пример предоставления визуального знака на основе атрибута показан на рис. 7.1:

```
<style>
a[hreflang] {
  padding-right: 18px;
  background-position: 100% 0;
  background-repeat: no-repeat;
}
a[hreflang|="en"] {
  background-image: url(img/usa.png);
}
a[hreflang|="es"] {
  background-image: url(img/esp.png);
}
a[hreflang|=fr] {
  background-image: url(img/fra.png);
}
</style>

<ul>
  <li><a href="/us/index.html" hreflang="en-us">English</a></li>
  <li><a href="/fr/index.html" hreflang="fr-fr">Français</a></li>
  <li><a href="/fr/index.html" hreflang="es-es">Español</a></li>
</ul>
```

Рис. 7.1. Использование селекторов атрибутов для обозначения языка

В теле страницы только ссылки на файлы на французском языке будут иметь атрибут `hreflang` со значением `fr`. Используя селектор атрибутов для нацеливания

на язык, вам не нужно знать, где находится сам элемент, каковы его родительские элементы и т. д. При использовании селекторов атрибутов неважно, кем и как обновляется сайт, необходимые элементы все равно могут получить нужный для них стиль. Селекторы атрибутов позволяются нацеливаться на элементы по их атрибутам, и в таких случаях нет необходимости засорять свой код HTML какими-то дополнительными классами.

В CSS3 добавлены еще более эффективные селекторы атрибутов, включая значения атрибутов, начинающиеся с конкретной подстроки и имеющие конкретную подстроку, в том числе и те значения, в которых искомая подстрока содержится в любом месте значения атрибута.

Разобраться в этих селекторах лучше на примерах. Поэтому вместо описания каждого из селекторов атрибутов мы продолжим задавать соответствия строкам HTML с помощью различных типов селекторов атрибутов. Новые селекторы атрибутов, введенные в CSS3, показаны в табл. 7.5:

```
<ul>
  <li><a href="http://x.com/selectors.pdf" hreflang="en-us" rel="nofollow"
    title="CSS-селекторы и поддержка браузером">Специфика</a></li>
  <li>
    <input type="checkbox" name="spec" id="spec" value="web workers
     rock"/>
    <label for="spec">Есть в спецификации веб-работники?</label>
  </li>
</ul>
```

Таблица 7.5. Селекторы атрибутов CSS3 позволяют задавать соответствие по подстроке в начале значения, в конце значения или в любом месте значения атрибута

Селектор	Пример	Описание
E[attr^=val]	a[href^="http"] input[value^="web"]	Значение val соответствует началу значения атрибута
E[attr$=val]	a[href$=".pdf"] input[name$="spec"]	Значение val соответствует окончанию значения атрибута
E[attr*=val]	a[href*=":"]	Значение val соответствует любому месту значения атрибута

Селекторы атрибутов позволяют нацеливать CSS на элементы на основе их атрибутов и значений атрибутов. В предыдущем перечне есть ряд полезных примеров. Например, `a[href^=http]` нацеливает CSS на любой полностью указанный URL-адрес[1], а `a[href$=".pdf"]` свидетельствует о том, что ссылка, скорее всего, указывает на файл формата PDF.

[1] a[href^=http] соответствует любой полностью указанный URL, использующий протокол HTTP или HTTPS, а также другие ссылки с путевым именем, начинающимся с HTTP. Можно конкретизировать цель, написав a[href^="http://"], a[href^="https://"]. Селекторы атрибутов можно объединять, например, так: a[href^=http][href$=pdf], что будет соответствовать внешним ссылкам на PDF-файлы и другим ссылкам, начинающимся на HTTP и заканчивающимся на PDF, но запись a[href^="http://"] [href^="https://"] не будет соответствовать ничему, поскольку ссылка не может начинаться как с http:, так и с https:.

Возможно, в таблице стилей потребуется обозначить неродственную ссылку со значком, указывающим на то, что ссылка ведет на другой домен, или обозначить, что ссылка ведет к файлу в PDF-формате, а не к веб-странице, а можно даже добавить текст к ссылке, чтобы обозначить тип ссылки. Например, пользователям, возможно, захочется получить предупреждение перед щелчком на ссылках, приводящих к загрузке файлов или открытию новых окон. Можно обозначить тип файла (рис. 7.2), воспользовавшись селекторами атрибутов, задающими соответствие строкам:

```
<ul>
  <li><a href="file.zip">Link 1</a></li>
  <li><a href="file.pdf">Link 2</a></li>
  <li><a href="file.html">Link 3</a></li>
  <li><a href="file.html" target="_blank">Link 3</a></li>
</ul>

<style>
  a[target="_blank"]::after {content: " (opens in new window)";}
  a[href$=".zip"]::after {content: " (.zip file)";}
  a[href$=".pdf"]::after {content: " (.pdf file)";}
</style>
```

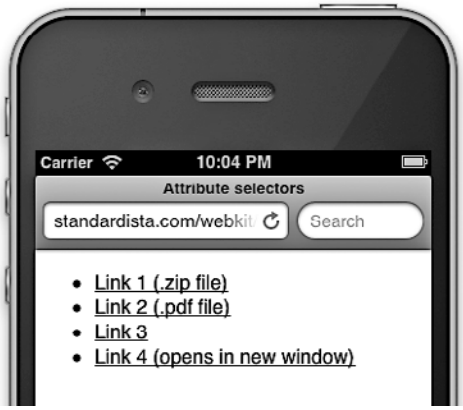


Рис. 7.2. Содержимое, сгенерированное на основе атрибутов элементов

Заключение значений атрибутов в ссылки в ряде случаев является необязательным, но требуется, когда в них включены символы, не относящиеся к буквам и цифрам, например, пробелы и двоеточия. Имя атрибута нечувствительно к регистру символов, но если значение не является значением HTML, то значение атрибута к нему чувствительно. Например, оба выражения, `[type=CHECKBOX]` и `[type=checkbox]`, будут указывать на флажки независимо от того, как они фигурируют в разметке, а вот выражения `a[href^=http]` и `a[href^=HTTP]` будут указывать лишь на протоколы, записанные в нижнем и верхнем регистрах соответственно[1].

[1] CSS-селекторы уровня 4 разрешат нечувствительное к регистру символов соответствие. См. приложение.

Можно даже на основе ширины окна просмотра задать с помощью медиазапросов разные стили разным типам ссылок. Например, если устройство имеет экран достаточной ширины, можно включить перед ссылкой фоновое изображение, а после ссылки — тип документа, опуская это усовершенствование, если окно просмотра будет слишком узким:

```
@media screen and (min-width: 480px) {
  a[href^="mailto:"] {
    padding-left: 30px;
    background: url(emailicon.png) no-repeat left center;
  }
}
@media print, screen and (min-width: 640px) {
  a[href^="mailto:"]::after {
    content: " (" attr(data-address) ")";
    opacity: 0.7;
  }
}
```

В предыдущем фрагменте кода при ширине окна браузера, равной или больше 480 пикселов, к ссылке добавляется значок ссылки на электронную почту, а также адрес электронной почты в том виде, в котором он указан в атрибуте данных адреса, если ширина окна браузера равна или больше 640 пикселов или если данные выводятся на печать.

CSS 2.1 предусматривает создание части содержимого страницы с помощью CSS. В предыдущих примерах создавался текст, информирующий пользователя о типе файла, который будет загружен по ссылке, или о том, что это ссылка на адрес электронной почты, при этом тип ссылки определялся на основе значения URL-адреса ссылки. При всей полезности свойства, позволяющего создавать часть содержимого страницы, цель вообще-то состояла не в том, чтобы включить в страницу текст как таковой, а в том, чтобы включить только тот текст, который необходим для понимания ситуации на странице. Текст нужно создавать не для реального содержимого страницы, а для последовательного усовершенствования ее внешнего вида.

В данном примере использовались два свойства CSS 2.1. Мы воспользовались селекторами атрибутов вместе с псевдоэлементом `::after` и создали часть содержимого страницы с помощью свойства CSS 2.1 `content`. Селекторы атрибутов могут использоваться не только для нацеливания на элементы, но и для повышения удобства работы со страницей путем добавления созданной части содержимого с помощью соответствующих свойств CSS 2.1 (рассматриваемых в подразделе «Псевдоклассы»).

В понятиях конкретизации нацеливания все селекторы атрибутов независимо от степени их конкретизации имеют такой же вес, что и селектор класса.

CubeeDoo. Вся игра CubeeDoo основывается на атрибуте `data-value`. Нацеливание внешнего вида лицевой части карты основано на значении атрибута `data-value`. Цветовая тема игры полностью определяется изменением класса панели на `colors`[1].

[1] Есть также классы numbers и shapes. Класс numbers будет рассмотрен в данной главе чуть позже, а класс shapes рассматривался в главе 5 при изучении SVG.

Цвет фона содержимого контейнера изменяется в зависимости от значения атрибута data-value родительского элемента:

```
.colors div[data-value="0"] .back {background-color:transparent;}
.colors div[data-value="1"] .back {background-color:#F00;}
.colors div[data-value="2"] .back {background-color:#090;}
.colors div[data-value="3"] .back {background-color:#FF0;}
.colors div[data-value="4"] .back {background-color:#F60;}
.colors div[data-value="5"] .back {background-color:#00F;}
.colors div[data-value="6"] .back {background-color:#909;}
.colors div[data-value="7"] .back {background-color:#F0F;}
.colors div[data-value="8"] .back {background-color:#633;}
.colors div[data-value="9"] .back {background-color:#000;}
.colors div[data-value="10"] .back {background-color:#fff;}
.colors div[data-value="11"] .back {background-color:#666;}
.colors div[data-value="12"] .back {background-color:#ccc;}
```

Псевдоклассы

Псевдоклассы похожи на классы, но при их определении разработчик не помещает атрибут class в открывающий тег элемента HTML. Псевдоклассы определяются на основе позиции в объектной модели документа — DOM или на основе текущего состояния пользовательского интерфейса. С точки зрения конкретизации и каскадности псевдокласс имеет такой же вес, как и обычный класс, что показано в приложении и сказано в разделе «Другие селекторы: теневая DOM-модель».

Существуют два псевдокласса ссылки: :link и :visited. Псевдокласс :link соответствует непосещенным ссылкам, а :visited — посещенным. Хотя эти два псевдокласса могут применяться для повышения удобства использования, они являются источником риска для системы безопасности. Браузер Safari версии 5 исключил их поддержку, и этому примеру последовали другие поставщики браузеров. Задание стилей для этих двух представителей псевдоклассов, в отличие от остальных, носит весьма ограниченный характер. Это единственные два типа классов, на которые распространяются подобные ограничения.

С глобальным атрибутом tabindex применение псевдоклассов пользовательской активности :hover, :active и :focus больше не ограничивается только ссылками и формами. Как ссылки, так и элементы форм всегда могли иметь фокус и/или быть активными. Поэтому :focus и :active были актуальны для всех интерактивных элементов. С помощью tabindex интерактивным может быть любой элемент. Поэтому псевдоклассы :focus и :active могут применяться к элементам с атрибутом tabindex (см. главу 2).

ПРИМЕЧАНИЕ

Для большего удобства и простоты использования применяйте объявление :focus вместе с :hover.

Как известно, при наличии мыши ее указатель может пройти над любым элементом. И код CSS позволяет реагировать на это. Псевдокласс :hover может быть добавлен к любому элементу. Но на сенсорных устройствах вы не проводите указателем

мыши над элементами — вы прикасаетесь к элементам. Мобильные устройства трактуют прикосновение точно так же, как и проход над элементом указателя мыши, а некоторые браузеры мобильных устройств и операционные системы таких устройств добавляют ряд дополнительных свойств. К числу свойств, которые могут потребоваться для задания стилей, относятся следующие:

- `-webkit-tap-highlight-color`. Позволяет устанавливать цвет фона основы — `background-color`, когда пользователь прикасается к ссылке или к какому-нибудь другому элементу, на котором может быть произведен щелчок кнопкой мыши. Изначально вместо цвета выделения элемента, к которому прикоснулся пользователь, устанавливается режим полупрозрачности основы. Стиль можно изменить, но выключать его не стоит. Возможность браузера показать пользователю, к какому элементу он прикоснулся, улучшает пользовательское восприятие. А сокрытие факта прикосновения это восприятие ухудшает;
- `-webkit/moz/ms-user-select`. На настольных и мобильных устройствах это свойство доступно с префиксом, и при установке его значения в `none` можно лишить пользователя возможности выбирать текст или по крайней мере создать видимость того, что он не может выбрать текст (на самом деле ничто не мешает ему выбрать содержимое страницы). Это свойство имеет экспериментальный характер, и его пока нет в спецификациях, хотя я ожидала его там увидеть. Оно должно иметь префикс производителя и пользуется повсеместной поддержкой, за исключением Opera до перехода на движок Blink, хотя и не является стандартным;
- `-webkit-touch-callout`. Когда для этого свойства установлено значение `none`, оно не дает появляться вызываемой панели инструментов (для выбора, копирования или вставки), когда пользователь удерживает ссылку.

Вдобавок к новому свойству поддержки псевдоклассов `user-action` CSS3 предоставляет еще два псевдокласса `user-interface` (пользовательского интерфейса) и массу псевдоклассов `user-interaction` (взаимодействия с пользователем).

Можно нацеливаться на доступные и недоступные элементы: `:enabled` и `:disabled`. С помощью псевдокласса `:checked` можно нацеливаться на выбранные элементы ввода данных с указаниями типа `type="checkbox"` и `type="radio"`.

В табл. 7.6 перечислены псевдоклассы пользовательского интерфейса и описания элементов, соответствующих их селекторам.

Таблица 7.6. Псевдоклассы CSS и элементы, соответствующие селекторам псевдокласса

Псевдокласс	Что ему соответствует
:link	Непосещенные ссылки. Поддерживаемый со времен CSS 1, этот псевдокласс из соображений безопасности в новых браузерах полностью не поддерживается
:visited	Посещенные ссылки. Поддерживаемый со времен CSS 1, этот псевдокласс из соображений безопасности в новых браузерах полностью не поддерживается
:hover	Не только ссылки, но и любой другой элемент, над которым проходит указатель мыши

Псевдокласс	Что ему соответствует
:active	Текущий активизированный пользователем элемент
:focus	Элемент, имеющий фокус на основе событий прикосновения, с помощью клавиатуры, или мыши, или других средств ввода
:enabled	Доступный элемент пользовательского интерфейса
:disabled	Недоступный элемент пользовательского интерфейса
:checked	Выбранный элемент переключателя или установленный флажок
:indeterminate	Элементы формы в неопределенном состоянии (ни установлены, ни сняты)

Когда элемент получает фокус, псевдокласс `:active` активизируется не на всех устройствах. Исправить положение можно с помощью добавления класса `.active` в том месте, где в вашем CSS установлен псевдокласс `:active`, и добавления и удаления класса `.active` с помощью отслеживателей событий начала и конца прикосновения `touchstart` и `touchend`.

На сенсорных устройствах, не использующих мышь, указатель, проходящий над объектом, отсутствует. Создать аналог событию прохождения указателя можно с помощью событий `touchstart` и `touchend`, а со временем — с помощью событий прихода и ухода указателя `pointerenter` и `pointerleave`. Большинство сенсорных устройств обрабатывают период между событиями `touchstart` и `touchend`[1] как `:hover`, но при этом они более привередливы в смысле определения элементов с псевдоклассом `:active`. Чтобы гарантировать поддержку `.hover`, подобную поддержке `:hover`, и поддержку `.active`, подобную поддержке `:active`, в отношении любого элемента с атрибутом `tabindex` всеми браузерами, поддерживающими сенсорные устройства, можно добавить следующий сценарий:

```
var myLinks = document.querySelectorAll('[tabindex]');
for (var i = 0; i < myLinks.length; i++) {
  myLinks[i].addEventListener('touchstart',
    function() {
        this.classList.add('hover');
        this.classList.add('active');
      }, false);
  myLinks[i].addEventListener('touchend',
    function() {
        this.classList.remove('hover');
        this.classList.remove('active');
      }, false);
}
```

[1] События touchstart и touchend являются нестандартными и запатентованными. События прикосновения запатентованы компанией Apple, но спецификации являются открытыми стандартами. Стандартом прикосновений в W3C являются события указателя — pointer. Хотя эти нестандартные touch-события в данный момент поддерживаются всеми сенсорными устройствами, браузеры скоро станут поддерживать pointer-события, и события touchstart и touchend на этих устройствах, за исключением устройств Apple, могут войти в разряд нерекомендованных. См. главу 13.

Этот код добавляет класс hover ко всем элементам, к которым привлекается внимание или которые активизируются прикосновением пальца, а не проходом над ними указателя мыши. Там, где в вашем коде CSS будет обращение к псевдоклассу hover, нужно добавить класс hover:

```
.hover, :hover {
  /* css для состояния hover */
}
.active, :active[1]{
  /* css для состояния active */
}
```

Вообще-то селектор класса .hover в CSS добавлять не нужно. Большинство браузеров сенсорных устройств будут корректно обрабатывать объявление :hover, если элемент будет вами идентифицирован и если у них есть событие touchStart:

```
<script>
var everything = document.querySelectorAll('a, label, span, input, [tabindex]');
for (var i = 0; i < everything.length; i++) {
  everything[i].addEventListener('touchstart',
    function() {
        // пусто
      }, false);
}
</script>
<style>
a:hover,
label:hover,
span:hover,
input:hover {
  /* css для состояния hover */
}
</style>
```

Если собрать все изученное вместе, в вашем арсенале появится ряд весьма удобных инструментов. Можно будет задавать стиль для элемента label на основе установки флажка!

```
<li>
  <input type="checkbox" name="spec" id="spec" value="web workers rock"/>
  <label for="spec">Are web workers in the specifications?</label>
</li>

input[type=checkbox]:checked + label {color: red;}
```

Теперь понять эту строку будет значительно легче. Она означает: «Надпись, идущая сразу же за установленным флажком, должна быть красной». В этом селекторе используются селектор атрибута, псевдокласс :checked и комбинатор смежного элемента того же уровня.

---

[1] Этот селектор лучше не использовать в качестве глобального (*) подразумеваемого селектора. Его применение нужно конкретизировать.

Псевдоклассы состояния

Псевдоклассы состояния пользовательского интерфейса (UI) пока еще не пользуются общей поддержкой браузеров. UI-модуль спецификации CSS3 представляет несколько псевдоклассов (показаны в табл. 7.7) для определения дополнительных состояний пользовательского интерфейса, которые становятся все более актуальными и лучше поддерживаются HTML5. Хотя псевдоклассы `:required`, `:valid` и `:invalid` использовались приблизительно с 2004 года, они приобрели актуальность не так давно благодаря ставшей присущей браузерам проверке приемлемости данных формы и добавленному в веб-формы HTML5 атрибуту `required` (см. главу 4).

Таблица 7.7. Псевдоклассы состояния UI и элементы, которые им соответствуют

Псевдокласс	Что ему соответствует
:default	Применяется к одному или нескольким UI-элементам, которые изначально входят в набор подобных элементов
:valid	Применяется к элементам, чья приемлемость основана на ожидаемом типе или шаблоне, определяющем семантику приемлемости данных
:invalid	Применяется к элементам, которые не соответствуют семантике приемлемости данных, определяемой типом или шаблоном
:in-range	Применяется к элементам, имеющим ограничения диапазона значений, значение которых не должно выходить за рамки этого диапазона
:out-of-range	Применяется к элементам, имеющим ограничения диапазона значений, значение которых должно выходить за рамки этого диапазона
:required	Применяется к элементам формы, имеющим установленный атрибут required
:optional	Применяется ко всем элементам формы, у которых нет установленного атрибута required
:read-only	Применяется к элементам, чье содержимое не может быть изменено пользователем
:read-write	Применяется к элементам, чье содержимое может быть изменено пользователем, например к полям ввода текста или contentEditable-полям (см. главу 2)

ПРИМЕЧАНИЕ

Псевдоклассы UI, или псевдоклассы состояния, определены в модуле Basic User Interface Module, а не в модуле селекторов CSS3. Они будут включены в спецификацию в виде части CSS-селекторов уровня 4 (CSS Selectors Level 4).

Структурные псевдоклассы

В CSS3 добавлено много селекторов, позволяющих разработчикам нацеливаться на элементы на основе структуры файлов HTML и на основе DOM. В табл. 7.8 перечислены все структурные псевдоклассы и дано краткое описание их значений. Если что-то будет непонятно, переживать не стоит. Математика структурных псевдоклассов *n*-го типа будет рассмотрена в следующем подразделе.

Таблица 7.8. Структурные псевдоклассы и их определения

Псевдокласс	Что ему соответствует
:root	Корневой элемент, которым в документах HTML5 всегда является <html>
:nth-child()	Элемент, являющийся *n*-м дочерним элементом своего родителя
:nth-last-child(n)	*N*-й дочерний элемент своего родителя, высчитываемый от последнего дочернего элемента
:nth-of-type(n)	*N*-й элемент данного типа, стоящий на таком же уровне родства
:nth-last-of-type(n)	*N*-й элемент данного типа, стоящий на таком же уровне родства, высчитываемый от последнего элемента этого же уровня
:first-child	Первый дочерний элемент своего родителя (CSS 2); то же самое, что :nth-child(1)
:last-child	Последний дочерний элемент своего родителя (CSS 2); то же самое, что :nth-last-child(1)
:first-of-type	Первый элемент данного типа, стоящий на таком же уровне родства; то же самое, что :nth-of-type(1)
:last-of-type	Последний элемент данного типа, стоящий на таком же уровне родства; то же самое, что :nth-last-of-type(1)
:only-child	Единственный дочерний элемент своего родителя
:only-of-type	Единственный элемент такого же уровня родства этого типа
:empty	Элемент, не имеющий дочерних элементов (включая текстовые узлы)

Что такое элемент `:root`, долго объяснять не нужно. Это корневой элемент документа, которым в документе HTML5 всегда является `<html>`. А вот псевдоклассы типа *n*-х нуждаются в некотором разъяснении.

Математика *n*-х типов

`:nth-of-type()`, `:nth-child()` и другие структурные псевдоклассы позволяют задавать соответствия элементам на основе их позиции по отношению к их предкам и элементам одного уровня родства. Эти селекторы получают аргумент, позволяющий выявить те элементы, к которым нужно применить стиль. Этот аргумент может быть ключевым словом, числом или числовым выражением.

Even и odd

Существуют два ключевых слова, `odd` (четный) и `even` (нечетный), заставляющие селектор нацеливаться на каждый следующий элемент заданного типа, начиная с первого элемента для `odd` или со второго элемента для `even`.

Например, *n*-е псевдоклассы часто используются с ключевыми словами `odd` и `even` для создания в таблице чередующихся полос, или *зебры*. Таблицы данных, особенно широкие и/или длинные, может быть трудно читать. Задавая для каждой нечетной строки другой фоновый цвет, можно облегчить чтение таблицы:

```
table {
  background-color: #ffffff;
}
tr:nth-of-type(even) {
  background-color: #dedede;
}
```

С помощью этого псевдокласса полосатость таблицы динамически задается кодом CSS. Не нужно добавлять классы непосредственно к элементу <tr>, как мы привыкли делать. И при сортировке данных не нужно переживать за изменение цвета тех или иных строк. Каждая нечетная строка автоматически получит серый цвет фона, даже если строки будут отсортированы или частично удалены.

Код придает полосатость строкам таблицы за счет того, что все <tr>-элементы относятся к одному уровню родства — являются дочерними элементами для <tbody>. Структурные селекторы вычисляют элементы, имеющие того же родителя. Это не «все строки таблиц в документе». Вместо этого вычисляются все <tr>-элементы одного уровня родства, и вычисление начинается снова, когда дело доходит до второй таблицы, будь она вложенной или нет.

Отдельные элементы

Если нужно нацелиться только на один элемент на основе его позиции, в качестве параметра нужно включить целое число. Продолжая работу с предыдущим примером, можно написать:

```
tr:nth-of-type(8) {
  background-color: #ff0000;
}
```

Этот код CSS сделает фон восьмой строки красным. Чтобы этот селектор заработал, его нужно поместить в таблице стилей *после* селектора `tr:nth-of-type(even)` (поскольку оба селектора имеют одинаковую конкретизацию (0-1-1), приоритет остается за последним в каскаде) или в порядке задания разметки.

Учтите, что браузер вычисляет соответствие секторам `nth-child` и `nth-of-type` родительского элемента. Если таблица вложена в ячейку другой таблицы, то будут соответствовать восьмые строки обеих таблиц, и внешней и вложенной.

Сравнение :nth-of-type с :nth-child

Разница между `:nth-of-type` и `:nth-child` весьма незначительна, и применительно к нашему примеру они часто будут нацеливать код на один и тот же узел. И тем не менее разница есть:

```
p:nth-child(3) {color: red;}
p:nth-of-type(3) {color: blue;}
```

Селектор `p:nth-child(3)` вызовет проверку третьего дочернего элемента каждого родительского узла, чтобы узнать, не является ли этот элемент абзацем. В следующем примере элемент 3 является третьим дочерним элементом, но вторым абзацем, и поэтому его содержимое приобретет красный цвет. Селектор `p:nth-of-type(3)` будет подсчитывать только дочерние абзацы элемента, выбирая третий найденный абзац, или в данном случае элемент 4 , содержимое которого окрасится в синий цвет:

```
<article>
  <p>1</p>
```

```
  <div>2</div>
  <p>3</p>
  <p>4</p>
</article>
```

В таблице селектор `:nth-of-type(8)` будет иметь такой же эффект, что и селектор `:nth-child(8)`, поскольку дочерними элементами узла `<tbody>` могут быть только элементы `<tr>`. Если бы мы написали `p:nth-of-type(8)` и `p:nth-child(8)`, то с помощью этих селекторов могли бы не нацелиться на один и тот же абзац. Селектор `:nth-of-type(8)` подсчитывает абзацы в родительском элементе и выбирает восьмой абзац, вложенный в один и тот же родительский элемент, если их не менее восьми. А селектор `:nth-child(8)` будет вести подсчет дочерних элементов (не потомков, а только непосредственных дочерних элементов) родительского элемента, пока не дойдет до восьмого. Если все восемь дочерних элементов будут абзацами, то мы получим полное совпадение в работе селекторов. Если нет, браузер для подсчета дочерних элементов переместится к следующему элементу.

Числовые выражения

И наконец, менее очевидным и определенно более веским является тот факт, что эти селекторы поддерживают числовые выражения.

Числовые выражения записываются как `(xn+y)`, где `x` — коэффициент повторения, а `y` — смещение. Например, вместо использования ключевых слов `even` и `odd` можно использовать `(2n)` для четных элементов и `(2n-1)` — для нечетных.

В качестве пояснения: `(2n)` означает каждый 2-й элемент, начиная с $2 \cdot 0$, затем $2 \cdot 1$, после этого $2 \cdot 2$, то есть 2, 4, 6, 8, 10-й и т. д. `(2n-1)` означает каждый 2-й элемент, начиная с элемента, на единицу меньше 2-го, или каждый нечетный элемент, то есть 1, 3, 5, 7, 9-й и т. д. Среди других примеров можно выбрать выражение `(4n-2)`, которое будет нацеливать селектор на каждый 4-й элемент, начиная со 2-го, то есть 2, 6, 10, 14-й и т. д.

Знак «+» или «–» нужен, только если в выражение включено смещение. Если нужно нацелиться на каждый 5-й элемент, начиная с 5-го элемента, напишите просто `(5n)`. Следует заметить, что если смещение включено в выражение, то оно должно быть последним (после параметра `n`, если таковой имеется), иначе селектор даст сбой.

Если включить в выражение большое смещение, например `2n+9`, то первым целевым элементом будет 9-й вычисленный элемент. Итерации начинаются с `n = 0` и всякий раз увеличиваются на 1. Выражение `:nth-of-type(2n+9)` приведет к нацеливанию на 9, 11, 13-й элементы. Выражение `:nth-last-of-type(2n+9)` заставит браузер найти последний элемент, отсчитать в обратном направлении девять элементов и нацелить код на 9, 11, 13-й и т. д. элементы с конца. Иными словами, последние восемь элементов не будут соответствовать выражению, а четные или нечетные элементы будут соответствовать в зависимости от того, каким было число дочерних элементов у родителя — четным или нечетным.

В CubeeDoo для иллюстрации работы селектора `:nth-of-type()` мы заставили совпадающие карты последовательно исчезать. В коде CSS было предписано второй совпадающей карте начинать постепенно исчезать после истечения 250 мс, а нацеливание на вторую совпадающую карту производилось с помощью следующего кода:

```
#board > div.matched:nth-of-type(2){
  -webkit-animation-delay: 250ms;
}
```

Приблизительно такие структурные селекторы использовались в примере естественного представления информации на экране iPhone, где обеспечивалось удаление нижней границы у последнего языка из списка языков. Нацеливание велось на последний элемент списка, чтобы удалить его нижнюю границу:

```
article ul li:last-of-type {
  border-bottom: none;
}
```

Благодаря использованию структурных псевдоклассов нам не нужно знать, какой из языков последний, и не нужно добавлять класс к последнему элементу. Вместо этого для нацеливания на элемент на основе структуры мы воспользовались текущей структурой документа. Мы использовали выражение `:last-of-type`, но, поскольку дочерними для `<ul>`-элементов могут быть только `<li>`-элементы, могли бы также нацелиться на этот элемент с помощью выражений `:last-child`, `:nth-last-of-type(1)` или `:nth-last-child(1)`.

Следует заметить, что с точки зрения производительности использовать селектор `:first-of-type` выгоднее, чем селектор `:last-of-type`. У нас могла быть и, вероятнее всего, должна была быть включенной верхняя граница первого элемента списка языков, после чего мы написали следующий код:

```
article ul li:first-of-type {
  border-top: none;
}
```

Упражнение

Итак, я знаю, что вам уже не 12 лет (а если 12, то честь вам и хвала за изучение HTML5 и CSS3 в столь раннем возрасте), но, поскольку во всех этих выражениях нетрудно запутаться, давайте выполним несколько промежуточных упражнений, которые помогут увидеть, насколько полезными могут быть эти выражения.

Напишите выражения для нацеливания на следующие элементы.

1. Имеются 30 элементов, и нужно нацелиться на элементы 3, 8, 13, 18, 23 и 28.
2. Нужно нацелиться на элемент 17.
3. Имеются 10 элементов, и нужно нацелиться на элементы 1, 3, 5, 7 и 9.
4. Имеются 50 элементов, и нужно нацелиться на элементы 10, 20, 30, 40, 50.
5. Имеются 30 элементов, и нужно нацелиться на элементы 6, 10, 14, 18, 22, 26, 30.

Ответы

1. `(5n-2)` **или** `(5n+3)`.
2. `(17)`.
3. `(odd)` **или** `(2n-1)` **или** `(2n+1)`.
4. `(10n)`.
5. `(4n+2)`.

ВНИМАНИЕ

При включении псевдоклассов добавляется конкретизация.

ВНИМАНИЕ

Перед открывающей круглой скобкой или между множителем и *n* никаких пробелов ставить не нужно. А смещение следует указывать в последнюю очередь.

Дополнительные псевдоклассы

Несколько дополнительных, еще не рассмотренных нами псевдоклассов перечислено в табл. 7.9.

Таблица 7.9. Псевдоклассы :target, :lang и :not

Псевдокласс	Название	Что ему соответствует
E:target	Псевдокласс цели	Элемент, послуживший целью текущего активного внутристраничного указателя
E:lang(L)	Псевдокласс языка	Элемент в языке, обозначенный двухбуквенной аббревиатурой (L)
E:not(s)	Псевдокласс отрицания	Элемент E, не соответствующий селектору (s) в круглых скобках. Элементы, соответствующие E, за исключением тех, которые также соответствуют s

:target

Псевдокласс `:target` применяется или становится активным, когда элемент является текущей целью документа. Например, если есть `div`-контейнер с идентификатором `ID` и ваш пользователь щелкает на указателе ссылки, делая этот `div`-контейнер активным, все стили, установленные в блоке `:target`, будут применяться до тех пор, пока фокус не будет перемещен с цели в какое-нибудь другое место.

Например, селектор `#main:target` будет нацелен на элемент `<div id="main">`, когда URL читается как `thispage.html#main`. Стиль элементов можно задавать, основываясь на том, что они являются на странице текущей целью. В интернет-ресурсе главы есть пример показа и скрытия содержимого, оформленного в виде закладок с использованием только CSS.

:lang(en)

`E:lang()` или псевдоклассу языка соответствует элемент `E`, если `E` выведен на языке, переданном в виде заключенного в скобки параметра в выражении `:lang()`. Элемент `E` не обязательно должен иметь атрибут `lang`, примененный непосредственно к нему, он должен быть просто потомком того элемента, к которому применен соответствующий язык.

Например, ваш документ HTML с помощью объявления о языке `<html lang="en-us">` позиционирован как написанный на языке US English. Этот документ будет соответствовать любому селектору `E` с `E:lang(en)`, а вот селектору `E:lang(fr)` он соответствовать не будет, пока какой-нибудь подраздел вашей страницы не будет объявлен написанным на французском языке. Например, если внутри документа есть элемент `<blockquote lang="fr-fr">`, то селектору `p:lang(fr)` будет соответствовать

любой абзац, находящийся в блоке цитаты, но селектору `p:lang(en)`, которому соответствуют все остальные абзацы документа, те абзацы, которые находятся внутри блока цитаты, соответствовать не будут.

:not(s), или псевдокласс отрицания

Псевдокласс отрицания `:not(s)` представляет элемент, который не представлен аргументом `s`. Селектору `E:not(s)` будут соответствовать все элементы `E`, которые также не соответствуют аргументу в круглых скобках. Выражение `E:not(F)` обычно читается как «Ему соответствуют все элементы `E`, которые также не соответствуют `F`».

Аргумент в круглых скобках является простым селектором. Простой — не значит несложный, это значит, что в селекторе нет отношений типа «предок — потомок»:

```
input[type=checkbox]:not(:checked)
```

В предыдущем примере селектор соответствует всем элементам ввода данных типа `checkbox` (флажки), которые пока не установлены. Если селекторы CSS3 вам еще в новинку, то псевдокласс `:checked` может показаться сложным. Но он считается простым, поскольку им не определяются взаимоотношения в дереве объектной модели документа — DOM.

Селекторам с псевдоклассом `:not` соответствуют элементы, заданные слева от двоеточия, с последующим исключением из этой группы тех из них, которые также соответствуют тому селектору, который указан справа от двоеточия.

- `p:not(.copyright)`. Соответствуют все абзацы, *за исключением* тех, которые имеют класс `copyright`.
- `:not(a)`. Соответствуют все элементы, кроме ссылок.
- `p a:not(:visited)`. Соответствуют все непосещенные ссылки, найденные в абзаце.
- `li:not(:last-of-type)`. Соответствуют все элементы списка, за исключением последнего.
- `input:not([type=radio]):not([type=checkbox])`. Соответствуют все элементы ввода данных, за исключением переключателей или флажков.
- `h1:not(header > h1):not(#main h1)`. Не работает, поскольку `header > h1` и `#main h1` не являются простыми селекторами, поэтому селектор дает сбой и игнорируется.

Следует учесть наличие возможности совместного применения нескольких псевдоклассов, как показано в ранее приведенном примере с применением псевдокласса `:not` с селектором элементов ввода данных:

```
ul > li:nth-of-type(n+2):nth-last-of-type(n+2)
```

Предыдущий код будет нацеливать на все элементы списка, за исключением первого и последнего элементов неупорядоченного списка, точно так же, как это сделали бы две более простые версии:

```
ul > li:not(:first-of-type):not(:last-of-type)
ul > li:not(:first-child):not(:last-child)
```

В понятиях конкретизации у :not нет никакого веса, но конкретизацию добавляет аргумент, переданный в круглых скобках:

```
li:not(#someID) /* 1-0-1 селектор идентификатора добавляет к конкретизации 1-0-0 */
li:not([title]) /* 0-1-1 селектор атрибута добавляет к конкретизации 0-1-0 */
```

Практический пример

В следующем коде имеются флажок для ввода понятия «другое» (other) и текстовое поле, которое нужно показать только в том случае, когда установлен флажок «другое»:

```
<li>
  <input type="checkbox" value="other" id="other">
  <label for="other"> other: </label>
  <input type="text">
</li>
```

Чтобы скрыть и показать это текстовое поле на основе действий пользователя, можно составить комбинацию из нескольких селекторов:

```
input[type="checkbox"]:not(:checked) ~ input {
  display: none;
}
```

Этот код ищет неустановленные флажки, затем ищет элементы ввода того же родителя и скрывает эти элементы. Если их флажок установлен, то к текстовому полю свойство display: none применяться не будет.

Псевдоэлементы

Псевдоэлементы могут применяться при нацеливании на текст, который является частью документа, но не может быть целью в дереве документа. Например, у всех текстовых узлов есть первая буква. Но пока вы не заключите ее в -контейнер, эта первая буква не будет отдельной, доступной для нацеливания частью DOM.

Псевдоэлементы, как следует из предлагаемого для них имени, создают псевдоэлементы. С помощью псевдоэлемента ::first letter можно обратиться к первой букве элемента, как будто она являлась отдельным элементом DOM (чего нет на самом деле), и придать ей свой стиль. Псевдоэлементы позволяют разработчикам нацеливаться на ту информацию, на которую невозможно нацелиться другим способом, не добавляя к разметке никакой логики, выделяющей первую букву или первую строку.

:first-letter ссылается на первую строку текстового узла элемента. С точки зрения синтаксиса правильнее будет применять запись с двойным двоеточием, ::first-letter, но мы обычно используем систему записи с одинарным двоеточием, поскольку IE не поддерживает двойное двоеточие.

По аналогии с этим :first-line и ::first-line ссылаются на первую строку текста элемента. Хотя более правильной является запись с двойным двоеточием, запись с одинарным двоеточием лучше поддерживается браузерами.

Возможно, новым для вас станет использование псевдоэлемента `::selection`. С его помощью можно нацелиться на выделенный текст. `::selection` был удален из текущей спецификации селекторов CSS3, поскольку сдерживал окончательное согласование, но он поддерживается всеми браузерами (и долго поддерживался в Firefox, но все еще с префиксом `-moz-`).

При создании игры может потребоваться отключить произвольный выбор изображений и текста. Как уже упоминалось, для управления выделением или для его предотвращения можно включить в код несколько свойств:

```
.willNotBeSelectable {
  -webkit-tap-highlight-color: #bada55;
  -webkit-user-select: none;
  -webkit-touch-callout: none;
  -ms-touch-action: none;
}
```

С помощью свойства `tap-highlight` можно управлять фоновым цветом элементов, подвергшихся прикосновению. С помощью свойства `user-select: none;` можно заставить устройство не спрашивать пользователя о том, не хочет ли он скопировать и/или вставить содержимое. Это свойство может пригодиться при разработке игр: если наш пользователь держит карту в CubeeDoo слишком долго, мы не хотим, чтобы его отвлекали всплывающие вопросы, не хочет ли он сохранить карту без содержания. На последнее похоже свойство `touch-callout`, но оно предотвращает появление в отношении изображений диалогового окна. Когда для свойства `touch-action` устанавливается значение `none`, это заставляет операционную систему не показывать всплывающие окна при выполнении в Windows панорамирования.

::before и ::after

Псевдоэлементы `::before` и `::after` задают несколько иное поведение. Вместо нацеливания на текст внутри документа эти два псевдоэлемента предоставляют способ ссылки на содержимое, которое не существует в разметке или в DOM. Псевдоэлементы `::before` и `::after` дают возможность создавать содержимое. Например, можно добавить восклицательный знак к концу каждого элемента, имеющего класс предупреждения — `warning`:

```
.warning::after {content: '!';}
```

Можно не только добавлять содержимое, но и придавать ему стиль. Одним из наиболее частых примеров использования этих псевдоэлементов было решение `.clearfix`, в котором псевдоэлемент `:after` предназначен для четкого обозначения чисел с плавающей точкой. Ранее в данной главе было показано еще одно правильное применение этого псевдоэлемента: включение значка флага на основе данных о языке (см. рис. 7.1) и создание текста на основе типов файлов и типов ссылок (см. рис. 7.2).

При создании генерируемого содержимого *нужно* использовать свойство `content`, даже если это содержимое — пустая строка или в нем нечего выводить на экран. Сгенерированное содержимое появится внутри элемента перед узлами

содержимого или текста того элемента, который приходится текущему элементу родительским, или после последнего дочернего элемента или текстового узла. Хотя сгенерированное содержимое появится на экране так, будто оно было реальным содержимым, в DOM оно добавлено не будет.

Псевдоэлементы `:before` и `:after` поддерживаются всеми браузерами, включая IE, начиная с IE8.

CubeeDoo. В качестве примера в CubeeDoo сгенерированное содержимое используется с целью добавления содержимого для наших тем `numbers` и `shapes`:

```
.numbers div[data-value="1"] .back:after{ content:'1';}
.numbers div[data-value="2"] .back:after{ content:'2';}
.numbers div[data-value="3"] .back:after{ content:'3';}
.numbers div[data-value="4"] .back:after{ content:'4';}
.numbers div[data-value="5"] .back:after{ content:'5';}
.numbers div[data-value="6"] .back:after{ content:'6';}
.numbers div[data-value="7"] .back:after{ content:'7';}
.numbers div[data-value="8"] .back:after{ content:'8';}
.numbers div[data-value="9"] .back:after{ content:'9';}
.numbers div[data-value="10"] .back:after{ content:'10';}
.numbers div[data-value="11"] .back:after{ content:'11';}
.numbers div[data-value="12"] .back:after{ content:'12';}

.shapes div[data-value="1"] .back:after{ content:'★';}
.shapes div[data-value="2"] .back:after{ content:'●';}
.shapes div[data-value="3"] .back:after{ content:'⬣';}
.shapes div[data-value="4"] .back:after{ content:'■';}
.shapes div[data-value="5"] .back:after{ content:'↑';}
.shapes div[data-value="6"] .back:after{ content:'▶';}
.shapes div[data-value="7"] .back:after{ content:'♦';}
.shapes div[data-value="8"] .back:after{ content:'♥';}
.shapes div[data-value="9"] .back:after{ content:'♣';
.shapes div[data-value="10"] .back:after{ content:'♠';}
.shapes div[data-value="11"] .back:after{ content:'☻'}
.shapes div[data-value="12"] .back:after{ content:'↓';}
```

Путем простого изменения класса панели игры мы можем изменить тему. Для цветовой схемы меняем цвета фона на основе значения атрибута `data-value`. В примере спрайта изображения SVG мы просто изменили `background-position` на основе значения атрибута `data-value` и позицию целевого изображения в спрайте. Чтобы изменить тему и добавить числа и образы, мы воспользовались сгенерированным содержимым для фактического создания чисел и образов значков.

Сгенерированное содержимое появляется по умолчанию в текущей строке. Стиль его может быть полностью задан, за исключением анимации, но возможность анимировать сгенерированное содержимое вскоре должно быть реализовано и уже присутствует в Firefox.

Мы уже разбирали пример использования медиазапросов для определения ширины окна и обслуживания постепенного улучшения сгенерированного содержимого для ссылок на основе того, будут ли они помещаться на экране. Постепен-

но улучшающиеся ссылки — лишь один из многих примеров использования сгенерированного содержимого.

Сгенерированное содержимое может использоваться для замены изображения, вывода атрибутов в качестве значений на печать (или на экран), создания упорядоченного списка из любого элемента со счетчиками, придания стиля номерам упорядоченного списка, вывода заключенных в кавычки меток соответствующего языка, создания стилизованных подсказок, а также вынесенных эллипсов с текстом мыслей или слов персонажа и т. д. Ссылку на учебное пособие по генерированию содержимого можно найти в онлайн-ресурсах к главе.

Зачем применяется запись с двумя двоеточиями

Псевдоэлементы начинаются с двойного двоеточия (`::`), за которым следует имя псевдоэлемента. В CSS3 двойное двоеточие заменило для псевдоэлементов одинарное двоеточие с целью обозначить явное отличие псевдоклассов от псевдоэлементов. Для обратной совместимости синтаксис с одинарным двоеточием применим к селекторам, созданным до появления CSS3. Поэтому есть псевдоэлемент `::after` и есть псевдоэлемент `:after`, но с системой записи, применяемой к псевдоклассам, в то же время `:hover` всегда является псевдоклассом, а не псевдоэлементом, и перед ним разрешается ставить только одинарное двоеточие.

Эти два двоеточия, `::` (запись с двойным двоеточием), были введены W3 для того, чтобы «установить различие между псевдоклассами и псевдоэлементами. Для совместимости с существующими таблицами стилей пользовательские агенты должны также признавать для псевдоэлементов предыдущую систему записи с одинарным двоеточием, введенную в CSS уровней 1 и 2 (а именно, `:first line`, `:first-letter`, `:before` и `:after`)».

Другие селекторы: теневая DOM-модель

Вы полагали, что на этом все? Мы тоже! Но есть и другие псевдоклассы и псевдоэлементы, созданные производителями браузеров, но еще не ставшие частью спецификации. Например, для стилевого оформления ошибок, выявленных при проверке данных формы, в WebKit предоставляются четыре псевдоэлемента:

```
::-webkit-validation-bubble {}
::-webkit-validation-bubble-arrow-clipper {}
::-webkit-validation-bubble-arrow {}
::-webkit-validation-bubble-message {}
```

Но этими четырьмя селекторами всплывающих сообщений о неудачной проверке дело не ограничивается. Во всех браузерах есть множество псевдоэлементов, а применение WebKit в данный момент позволяет без особого труда нацелиться на свойства, присущие пользовательскому интерфейсу. Например, есть псевдоэлементы, позволяющие задать стиль индикаторам хода выполнения[1]:

[1] Описания псевдоэлементов и псевдоклассов Mozilla можно найти по адресу http://mzl.la/1cdK4mx

```
::-webkit-progress-bar {}
::-webkit-progress-value {}
```

Чтобы выяснить, на какие псевдоэлементы можно нацелиться и как выглядит правильный синтаксис для таких псевдоузлов, веб-инспектор Chrome позволяет инспектировать теневую DOM (рис. 7.3).

Рис. 7.3. Теневая DOM, инспектируемая в браузерном инструментарии разработчика

На рисунке можно увидеть стрелку, которая следует за полем ввода типа `range`. Поля ввода являются пустыми элементами и поэтому не могут иметь вложенных дочерних элементов. Но щелчок на этой стрелке приведет к показу теневой DOM или компонентов пользовательского агента, которые в данном случае определяют внешний вид ползунка диапазона. Щелкнув на псевдоэлементе `webkit-slider-runnable-track`, можно просмотреть стили пользовательского агента для направляющей. Задать стиль этого элемента можно с помощью использования в качестве селектора псевдоэлемента выражения `::-webkit-slider-runnable-track`. Щелкнув на вложенном `div`-элементе можно увидеть, что дочерний элемент может стать целевым с помощью выражения `-webkit-slider-thumb` и у него есть собственные стили. Применяя веб-инспектор, можно изучить все разнообразные компоненты теневой DOM, которым можно задать стиль:

```
input[type="range"]::-webkit-slider-runnable-track {
  -webkit-flex: 110px;
  min-width: 0px;
  -webkit-align-self: center;
  box-sizing: border-box;
  display: block;
  -webkit-user-modify: read-only;
}
input[type="range"]::-webkit-slider-thumb,
input[type="range"]::-webkit-media-slider-thumb {
  -webkit-appearance: sliderthumb-horizontal;
  box-sizing: border-box;
  display: block;
  -webkit-user-modify: read-only;
}
```

С введением веб-компонентов браузеры на основе Blink, возможно, сократят доступ к заданию стилей некоторым псевдоэлементам теневой DOM, отдав предпочтение веб-компонентам.

Конкретизация превыше каскадности: понятие о конкретизации CSS. CSS-объявления могут противоречить друг другу. Один и тот же элемент может быть объявлен в разных блоках селекторов и более крупным, и более мелким, и розовым, и оранжевым.

Но в спецификации CSS написано, что определить, какие именно значения свойств будут применены, можно всегда и разногласий среди браузеров по этому вопросу быть не должно: всегда можно определить, какое из правил получит преимущество на основе порядка и конкретизации или веса объявлений селекторов.

CSS-каскад представляет собой набор правил, определяющих, какое из них получит преимущество при, казалось бы, противоречивых объявлениях. Более конкретизированные правила переписывают правила общего характера. При равенстве конкретизации более поздние правила переписывают более ранние. При равном весе более близкие (или более поздние в каскаде) правила переписывают более отдаленные (или более ранние в каскаде). Все правила применяются от более общих к более конкретным, и каждое последующее более конкретное или более близкое правило переписывает предыдущие противоречащие ему объявления, относящиеся к свойству.

Какой селектор ни возьми, будь то селектор типа, класса, идентификатора или селектор одного из псевдоклассов, атрибутов и т. д., он определяет вес или конкретизацию правила. Два противоречащих друг другу заявления сравниваются в порядке очередности, только если имеют одинаковый вес.

Общий селектор (*), а также комбинаторы дочернего элемента, смежного элемента того же уровня и общий сборный комбинатор элементов одного уровня (>, + и ~) не добавляют веса конкретизации. Селекторы элемента и превдоэлемента имеют одинаковый вес самого низкого уровня.

У селекторов классов, атрибутов, псевдоклассов, включая структурные селекторы и селекторы элементов пользовательского интерфейса, один и тот же вес, но при этом селектор отдельного класса, атрибута или псевдокласса имеет более высокий вес, чем любое количество селекторов элементов.

Сам по себе псевдокласс отрицания `:not` не имеет значения, но конкретизация его параметра добавляет селектору вес. Когда у элемента в качестве значения атрибута `class` имеется два и более класса, исходный порядок этих классов в коде HTML не имеет никакого значения. В расчет берутся степень конкретизации селекторов и исходный порядок объявлений этих отдельных классов в коде CSS.

Отдельный селектор идентификатора имеет более высокий вес, чем селектор с любым количеством классов.

Общий порядок нарушается двумя обстоятельствами: встроенные стили имеют более конкретный характер, чем внедренные или привязываемые стили, а свойства с ключевым словом `!important` имеют еще более конкретный характер по сравнению со встроенными стилями. Но согласно рекомендуемым методам использования

(см. правило 5 из подраздела «Рекомендуемые методы использования CSS»), предписывающим никогда не использовать встроенные стили или объявление `!important` в готовой продукции (на эксплуатируемых сайтах), нам нужно сконцентрировать внимание только на понимании каскадности.

Если со всем этим так и не удалось разобраться, то определения веса класса, элемента и идентификатора в понятиях каскадности, а также комбинаций селекторов, нацеленных на абзацы в порядке степени их конкретизации, от самой низкой до самой высокой, с привлечением понятий рыб, акул и планктона даются на сайте http://specifishity.com. Список селекторов и их весов в понятиях степени конкретизации приведен в приложении.

Резюме

Вот и все! Мы всего лишь затронули часть имеющихся возможностей, чтобы изучить особенности CSS-каскада, конкретизации, селекторов и синтаксиса, но, надеюсь, этого будет достаточно, чтобы выполнить задуманное. В приложении содержится небольшая шпаргалка, напоминающая обо всех селекторах CSS3, полностью поддерживаемых во всех мобильных устройствах.

8 Расширение вариантов настроек с помощью значений CSS3

Поскольку мы ведем разработку мобильных приложений на современных браузерах на смартфонах, нам не приходится волноваться о том, что старые браузеры не поддерживают селекторы, свойства и значения CSS3. Такие устройства, как iPhone, iPod, iPad, современные телефоны на Android, планшетные компьютеры Galaxy, Microsoft Surface, и все остальные устройства, на которых работают WebKit, Firefox, Opera и Windows 8, обладают наиболее современными, совместимыми со стандартами браузерами. С прицелом на браузеры смартфонов (исключая некоторые устаревшие телефоны на базе Windows 7, но ситуация с ними стремительно меняется) мы можем двигаться дальше и программировать, используя самые передовые наработки CSS3 и HTML5. Держаться за старое не имеет смысла.

В данной главе мы направимся в путешествие, которое сделает нас разработчиками, использующими самые прогрессивные наработки CSS3. В предыдущей главе были рассмотрены селекторы CSS3 — самые передовые способы нацеливания на элементы с помощью CSS. Здесь приступим к рассмотрению самых новых значений CSS3. А в главе 9 будут рассмотрены вопросы использования некоторых самых новых свойств CSS3.

В спецификацию CSS3 введен ряд новых значений, а также ряд новых типов значений. В этой главе будут рассмотрены как старые, так и новые значения цветовых настроек, длины и углов. Будет показано, что эти значения можно использовать на всех браузерах, что несмотря на то, что они являются новыми значениями, появившимися в CSS3, они уже поддерживаются большинством браузеров, но при этом есть ряд значений ключевых слов, являющихся уникальными для тех или иных браузеров.

Цветовые значения CSS

До появления CSS3 было три типа цветовых форматов: шестнадцатеричный (и краткий шестнадцатеричный), формат `rgb()` и поименованные цвета. В CSS3 добавляется поддержка HSL, HSLA, RGBA и ряда других типов задания цвета, рассматриваемых в следующих разделах.

Форматы RGB, RGBA и шестнадцатеричный цветовой формат используют значения для красного, зеленого и синего цветов. Похожие на «фотошоповский» формат задания цвета HSB (hue, saturation, brightness — тон, насыщенность, яркость) форматы HSL и HSLA используют в качестве значений тон (hue), насыщенность (saturation) и яркость (light). Форматы RGBA и HSLA используются для объявления альфа-прозрачности выбранного цвета.

Переведем все это в примеры: в табл. 8.1 показаны форматы, включая новые, введенные в CSS3.

Таблица 8.1. Различные форматы объявления цвета в CSS[1]

Синтаксис задания цвета	Пример кода	Определение
#RRGGBB	#ff00ff	Шестнадцатеричный формат
#RGB	#f0f	Краткий шестнадцатеричный формат
rgb(r, g, b)	rgb(255, 0, 255) rgb(100%, 0, 100%)	Красный (red), зеленый (green), синий (blue)
hsl(h, s, l)	hsl(300, 100%, 50%)	Тон (hue), насыщенность (saturation), яркость (lightness)
cmyk[1](c, m, y, k)	cmyk(29%, 55%, 0, 0)	Голубой (cyan), пурпурный (magenta), желтый (yellow), черный (black)
hsla(h, s, l, a)	hsla(300, 100%, 50%, 1)	Тон (hue), насыщенность (saturation), яркость (lightness), альфа (alpha)
rgba(r, g, b, a)	rgba(255, 0, 255, 1) rgba(100%, 0, 100%, 1)	Красный (red), зеленый (green), синий (blue), альфа (alpha)
Поименованные цвета	fuchsia	Ограниченный перечень имен цветовых значений
Прозрачность	transparent	Прозрачность
Текущий цвет	currentColor	Цвет текста (текущий цвет)

Шестнадцатеричные значения

Шестнадцатеричные значения красного, зеленого и синего цветов можно задать в диапазоне от 0 до 255 в формате от `00` до `FF`, нечувствительном к регистру символов. Эти три значения нужно ставить вместе в следующем порядке: красный, зеленый, синий, а перед ними поставить знак решетки (#), в результате чего получится значение цвета.

Например, `#FFFFFF` означает полную насыщенность красного, синего и зеленого цветов, создающую белый цвет. Противоположностью этому служит полное отсутствие цветов. Следовательно, отсутствие красного, зеленого и синего цветов, записанное как `#000000`, создает черный цвет. Смесь сочетаний нечувствительных к регистру символов шестнадцатеричных значений от `00` до `FF` для красного, зеленого и синего цветов, объединяемая в следующем порядке: красный, зеленый, синий,

[1] CMYK-цвета поддерживаются не всеми браузерами и определены в модуле страничных сред — paged media, а не в цветовом модуле, как остальные перечисленные синтаксисы задания цвета.

может создать миллионы цветовых значений. Насыщенный красный цвет в отсутствие зеленого и синего цветов будет показан как ярко-красный, он записывается как #FF0000. Чем меньше насыщенность красного цвета, тем меньше будет его яркость, но он по-прежнему будет оставаться красным и может быть записан как #CC0000.

Я уже упоминала о нечувствительности символов к регистру? Не важно, как именно записывать цвет, #FFCC00 или #ffcc00, — синтаксис значений для цвета, как и для всех остальных значений свойств ключевых понятий, нечувствителен к регистру символов.

ПРИМЕЧАНИЕ

Тип ввода цвета, рассмотренный в главе 3, представляет цветовые значения в шестнадцатеричном формате с символами в нижнем регистре, с исходным значением #000000 в поддержке браузеров.

У шестнадцатеричной RGB-записи есть также краткая форма, #*RGB*, где каждый из компонентов, *R*, *G* и *B*, представлен одним нечувствительным к регистру символом в диапазоне A–F, или a–f, или 0–9. Три символа собираются вместе, и перед ними ставится символ решетки. Чтобы получить соответствие длинному формату, браузеры расширяют RGB-значение, например значение #369 расширяется до #336699. А значение #FF9900 может быть укорочено до #F90, но значение #F312AB не может быть записано в кратком виде[1].

Я считаю, что краткие записи читаются труднее, поэтому не использую их, но с точки зрения веб-стандартов в их применении нет ничего зазорного. К тому же, склоняясь к программированию с использованием символов только в нижнем регистре, я пришла к выводу, что шестнадцатеричные коды задания цвета легче читаются при применении прописных букв. Это мои личные предпочтения. Но какой бы вариант синтаксиса вы ни выбрали, нужно его придерживаться.

Следует заметить, что при использовании типа вводимых данных <input type="color"> там, где он поддерживается, возвращаемым значением по умолчанию будет #000000 и значения отправляются с символами в нижнем регистре.

Все браузеры поддерживают все шестнадцатеричные значения как в кратком, так и в развернутом варианте.

Если вы занимаетесь разработкой с 1990-х годов, то можете вспомнить дискуссию вокруг так называемых безопасных для веб-программирования цветовых настроек (web-safe colors). С существенным улучшением цветовой поддержки на всех устройствах, а не только на жидкокристаллических экранах «веб-безопасные цвета» утратили свою актуальность даже для портативных устройств. Теперь можно без всяких опасений использовать любые цветовые комбинации. Хотя некоторые цветовые сочетания могут иметь не слишком привлекательный вид или оказаться неразборчивыми, они все равно будут отображаться на экране.

[1] Восьмиразрядные шестнадцатеричные значения, в которых последние два разряда определяют альфа-прозрачность (где число FF означает полную непрозрачность, а 00 — полную прозрачность), являются частью спецификации CSS-цветов четвертого уровня (CSS Colors Level 4).

Синтаксис rgb()

Вместо использования для задания цвета рассмотренных в предыдущем разделе шестнадцатеричных значений для смеси красного, зеленого и синего цветов можно использовать значения в виде десятичных чисел или процентного отношения.

Вместо предшествующего цветовым установкам символа решетки в этом синтаксисе используется ключевое слово или функциональная запись `rgb`, за которой следуют разделенные запятыми значения в круглых скобках. Пробелы можно не ставить, но я считаю, что добавление пробелов облегчает чтение цветов:

```
#FFFFFF = #FFF = rgb(255, 255, 255) = rgb(100%, 100%, 100%).
```

Как правило, все цветовые комбинации RGB поддерживаются всеми браузерами. Некоторые браузеры позволяют смешивать `rgb()`-числа с процентами, но в спецификации четко определено, что это нестандартное поведение и его поддерживают не все браузеры, поэтому смешивания типов значений нужно избегать.

Правильно:

```
rgb(255, 255, 255)
rgb(100%, 100%, 100%)
```

Неправильно:

```
rgb(255, 100%, 255);
```

Добавление прозрачности с помощью формата RGBA

Новым в CSS3 является формат RGBA. Он похож на формат RGB, но с добавлением A для альфа-прозрачности.

Спецификация `rgb()` в CSS3 была расширена за счет включения спецификации `rgba()`, чтобы добавить термин alpha, позволяющий задавать степень прозрачности цвета. Первые три значения по-прежнему используются для задания красного (red), зеленого (green) и синего (blue) цветов. Четвертое значение задает степень прозрачности. Значение `1` означает полную непрозрачность, а значение `0` — полную прозрачность, значение `0.5` означает 50%-ную прозрачность. В качестве значения используется любое число с плавающей точкой между 0 и 1, включая и сами эти числа.

Расширяя пример с заданием белого цвета, напомню, что `1` означает полную непрозрачность, поэтому все следующие выражения будут эквивалентными:

```
rgb(255, 255, 255)
rgb(100%, 100%, 100%)
rgba(255, 255, 255, 1)
rgba(100%, 100%, 100%, 1)
```

Все они задают белый цвет, поскольку `1` означает полную непрозрачность. Но не запутайтесь: выражение `rgba(0, 0, 0, 0)` задает полную прозрачность, а не черный цвет, поскольку степень непрозрачности задана нулевой.

ПРИМЕЧАНИЕ

Учтите, что ключевое понятие прозрачности относится к прозрачности черного цвета, или rgba(0, 0, 0, 0), а не rgba(255, 255, 255, 0), что может иметь огромное значение при переходах от одних цветовых установок к другим.

На рис. 8.1 показано, что выражение `rgba(0, 0, 0, 1)` задает полностью непрозрачный черный цвет. Чем ближе к нулю значение альфа-прозрачности, тем более прозрачным будет цвет. На рис. 8.1 можно заметить, что через более прозрачные цвета фон проявляется более четко.

ПРИМЕЧАНИЕ

Формат RGBA особенно пригодится для создания теней элементов. Тени, отбрасываемые текстом или прямоугольными блоками, получаются за счет объявления цветов полностью прозрачными через несколько пикселов. Начинать нужно с частично прозрачного, а не со сплошного цвета, чтобы можно было что-то видеть за тенью.

Вместо text-shadow: 5px 4px 6px #666666; нужно использовать text-shadow: 5px 4px 6px rgba(0, 0, 0, 0.4);.

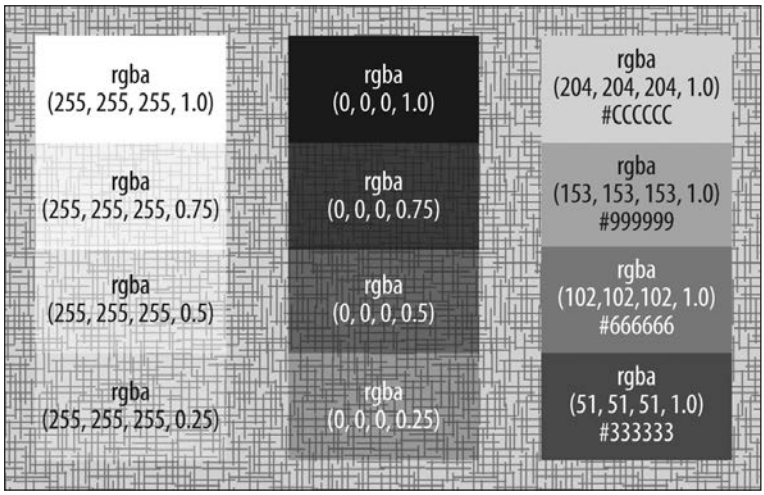


Рис. 8.1. Значение альфа-прозрачности позволяет объявлять цвет, который варьируется от полной непрозрачности до полной прозрачности

В отличие от RGB, для RGBA не существует шестнадцатеричной формы записи. Велись разговоры о включении восьмисимвольного шестнадцатеричного значения для RGBA, но до сих пор он так и не добавлен к черновой спецификации и его поддержка какими-либо мобильными браузерами не обеспечена.

Тон, насыщенность, яркость: HSL()

HSL является новым типом задания цвета, добавленным в CSS3. HSL означает: тон (hue), насыщенность (saturation) и яркость (lightness). HSL-формат упрощает создание цветовой палитры, поскольку в качестве основы можно выбрать тон, а затем

манипулировать настройками более светлых или темных и более или менее насыщенных оттенков выбранного тона.

Мониторы, в отличие от восприятия человеческим глазом, отображают цвета в оттенках красного, зеленого и синего цветов. Формат HSL имитирует восприятие, свойственное глазу человека. Мы воспринимаем цвета в тонах различных насыщенности и яркости. Формат HSL, как правило, более интуитивно понятен дизайнерам.

Синтаксис `hsl()` выглядит так же, как и синтаксис `rgb()`, но вместо применения значений для красного, зеленого и синего цветов значения, устанавливающие цвет, варьируются от 0 до 359, насыщенность и яркость задаются в процентах, при этом нормальной яркостью считается 50%.

В число значений для тона включаются: 0 = красный, 60 = желтый, 120 = зеленый, 180 = голубой, 240 = синий, 300 = пурпурный и все оттенки, находящиеся между ними. Поскольку книга отпечатана черно-белой (с оттенками серого цвета), возможность продемонстрировать цвета отсутствует. Но в онлайн-ресурсах к данной главе есть ссылка на средство выбора цвета в формате HSL.

Яркость означает количество света: 100% означают белый цвет (очень-очень яркий), 50% — фактический тон, а 0% — черный цвет при полном отсутствии света. Насыщенность 100% будет чистым тоном (полностью насыщенным цветом), а нулевая насыщенность даст оттенки серого от белого цвета до `#808080` и до черного цвета в зависимости от яркости.

Так же как у `rgb()` и `rgba()`, у формата `hsl()` есть версия с заданием альфа-прозрачности — `hsla()`. Синтаксис является функциональной записью `hsla()`, где тон (hue) задается степенью, насыщенность и яркость — процентным отношением, а альфа — значением от 0 до 1 и все эти значения заключаются в скобки в указанном порядке. Например, выражение `hsla(300, 100%, 50%, 0.5)` задает пурпурный, полностью насыщенный цвет со средней яркостью при 50%-ной непрозрачности.

При использовании формата HSL или HSLA значениями служат тон, насыщенность, яркость и альфа-прозрачность. Для создания белого и черного цветов тон может иметь любое значение, не обязательно 0, с полной яркостью (100%) для белого цвета и полным отсутствием таковой (0%) для черного.

CMYK

Приходилось ли вам менять чернильные картриджи в цветном принтере? Обратили внимание на цвета чернил в заменяемых картриджах? Скорее всего, вам приходилось ставить картриджи голубого цвета (похожего на бирюзовый), пурпурного (или ярко-розового) цвета, желтого и черного цветов. CMYK означает голубой (cyan), пурпурный (magenta), желтый (yellow) и черный (black), что соответствует системе, используемой в принтерах: дизайнеры печатной продукции (и ваш цветной принтер) используют CMYK, а не RGB.

Компьютерные мониторы показывают изображение в RGB. Люди, не страдающие дальтонизмом, обычно видят цвета как HSL. А вот в принтерах цвета определяются на основе CMYK. Формат CMYK поддерживают далеко не все браузеры

и, наверное, никогда не будут поддерживать, поскольку он разработан для принтеров. Да, формат CMYK входит в разряд непригодных для мобильных разработок, но я включила его сюда для полноты картины. CMYK упоминается в CSS3 в модуле страничных сред — paged media, а не в цветовом модуле (color module).

Поименованные цвета

Нами еще не рассмотрены ключевые слова, относящиеся к заданию цвета, такие как `aqua`, `fuchsia` и `lime`, и это связано с четырьмя основными причинами.

- Они сужают круг выбора значений и поэтому носят менее конкретный характер, чем миллионы значений, предоставляемых форматами RGB и HSL.
- При их задании нетрудно допустить опечатку, которая может вызвать непредсказуемые последствия (например, IE поддерживает только серые цвета — gray colors с буквой «а» в названии цвета, например lightgray, а не lightgrey).
- Из-за тонкостей реализации не все значения, выраженные ключевыми словами, абсолютно одинаково поддерживаются во всех браузерах или на всех дисплеях.
- И я полагаю, что от этого способа задания цвета нужно отказаться в силу первых трех причин.

Даже ключевое слово `transparent`, являющееся намного более быстрым способом написания эквивалента выражения `rgba(0, 0, 0, 0)`, не обошлось без проблем. Как уже упоминалось, слову `transparent` соответствует прозрачный черный цвет, что при переходе к цвету может привести к появлению неприглядных серых оттенков[1].

Если нужно перейти от `'transparent'` к красной тени `#FF0000`, используйте вместо этого выражение `rgba(255, 0, 0, 0)`, которое задает красный цвет с прозрачностью. В онлайн-ресурсах к главе есть список поименованных цветов, а также шестнадцатеричных, RGB- и HSL-значений.

CurrentColor

В CSS3 было добавлено еще одно ключевое слово, `currentColor` (текущий цвет), которое получает значение свойства `'color:'` или значение цвета текста или элемента, к которому оно применяется.

`currentColor` поддерживается всеми мобильными браузерами и наряду с текстовыми тенями может оказаться весьма полезным для создания более жирного текста для шрифтов, подобных Helvetica Neue Light:

```
.bolderText {text-shadow: 0 0 1px currentColor;}
```

[1] Использование ключевого слова transparent приведет при переходе к цветам, отличным от белого, к появлению неприглядных серых оттенков. Некоторые браузеры начали поддерживать transparent как прозрачный черный или прозрачный белый, но при переходе к любому тону во многих браузерах по-прежнему могут наблюдаться серые оттенки.

На устройствах с высоким разрешением экрана некоторые семейства шрифтов действительно выглядят слишком тонкими для чтения. Чтобы сделать текст немного разборчивее, можно непосредственно за ним добавить легкую тень того же цвета, что и текст. Это можно сделать, даже не зная цвета текста, объявив `currentColor` цветом тени.

Значения цветов браузера

В CSS 2.1[1] имеется множество определений системных цветов, подобных определенным в черновых спецификациях W3C CSS3 ключевым словам `currentColor` и `transparent`.

Эти поименованные цвета бывают разными при работе в разных браузерах и на разных операционных системах. Несмотря на то что они используются редко, я включила сюда их описание, поскольку они могут оказаться полезными при создании веб-приложений естественного внешнего вида:

- `activeBorder` — исходная граница активного окна;
- `activeCaption` — исходный заголовок активного окна;
- `appWorkspace` — исходный фоновый цвет рабочей области приложения или интерфейса;
- `background` — исходный цвет фона операционной системы;
- `buttonFace` — исходный внешний вид кнопки;
- `buttonHighlight` — исходный внешний вид выделенной кнопки;
- `buttonShadow` — исходная тень кнопки;
- `buttonText` — исходный текст кнопки;
- `captionText` — исходный текст заголовка;
- `grayText` — исходный серый текст;
- `highlight` — исходное выделение;
- `highlightText` — исходный выделенный текст;
- `inactiveBorder` — исходная неактивная граница;
- `inactiveCaption` — исходный неактивный заголовок;
- `inactiveCaptionText` — исходный неактивный текст заголовка;
- `infoBackground` — исходный фон информации;
- `infoText` — исходная информация;
- `match` — цветовое соответствие;
- `menu` — исходное меню;
- `menuText` — исходный текст меню;
- `scrollbar` — исходная полоса прокрутки;
- `threeDDarkShadow` — исходная темная 3D-тень;

[1] В спецификациях системных цветов.

- threeDFace — исходный внешний вид 3D-элементов;
- threeDHighlight — исходный внешний вид выделенного 3D-элемента;
- threeDLightShadow — исходная легкая 3D-тень;
- threeDShadow — исходная 3D-тень;
- windowFrame — исходная рамка окна;
- windowText — исходный текст окна.

Есть цвета, характерные для того или иного браузера, в том числе:

- -webkit-activelink — гиперссылка, на которой был сделан щелчок;
- -webkit-focus-ring-color — цвет, окружающий элемент пользовательского интерфейса, получивший фокус;
- -webkit-link — цвет посещенной гиперссылки;
- -webkit-text — цвет текста в окне, исходное окно;
- -moz-buttonDefault;
- -moz-buttonHoverFace;
- -moz-buttonHoverText;
- -moz-cellHighlightText;
- -moz-comboBox;
- -moz-ComboboxText;
- -moz-Dialog;
- -moz-DialogText;
- -moz-dragtargetzone;
- -moz-EvenTreeRow;
- -moz-Field;
- -moz-FieldText;
- -moz-html-CellHighlight;
- -moz-html-CellHighlightText;
- -moz-mac-accentdarkestshadow;
- -moz-mac-accentdarkshadow;
- -moz-mac-accentface;
- -moz-mac-accentlightesthighlight;
- -moz-mac-accentlightshadow;
- -moz-mac-accentregularhighlight;
- -moz-mac-accentregularshadow;
- -moz-mac-chrome-active;
- -moz-mac-chrome-inactive;
- -moz-mac-focusring;

- `-moz-mac-menuselect`;
- `-moz-mac-menushadow`;
- `-moz-mac-menutextselect`;
- `-moz-MenuHover`;
- `-moz-MenuHoverText`;
- `-moz-MenuBarText`;
- `-moz-MenuBarHoverText`;
- `-moz-nativehyperlinktext`;
- `-moz-OddTreeRow`;
- `-moz-win-communicationstext`;
- `-moz-win-mediatext`.

Цветовые установки с префиксами есть также у браузера Firefox.

С использованием цветов браузеров можно написать нечто подобное:

```
#myElement {
  color: -webkit-focus-ring-color;
  background-color: -webkit-link;
  text-shadow: 3px 3px 3px -webkit-text;
  -webkit-box-shadow: 4px 4px 4px -webkit-activelink;
}
```

Примеры кода со всеми этими ключевыми словами есть в онлайн-ресурсах к данной главе. Чтобы увидеть небольшие различия, откройте страницу в браузерах, использующих различные движки. Заметьте, что для большей удобочитаемости в элементах списка использованы символы нижнего и верхнего регистров, но на самом деле значения к регистру символов нечувствительны.

Запомните: браузеры игнорируют те строки CSS, которые он не понимают. Можно объявить цвет и тут же указать то же самое свойство и ключевое слово цвета из представленного ранее списка. Если браузер не понимает цветового значения, он проигнорирует эту строку CSS и применит ранее объявленный цвет.

Так какой же синтаксис использовать для задания цвета? Существует множество способов записи значений цвета в CSS. Темно-красный цвет может быть записан как `#800000`, `maroon`, `rgba(128, 0, 0)`, `rgba(128, 0, 0, 1.0)`, `hsl(0, 100%, 13%)` или `hsla(0, 100%, 13%, 1.0)`.

Если вы работаете в составе большой команды или все еще поддерживаете старые браузеры для настольных компьютеров, используйте шестнадцатеричный синтаксис, состоящий из шести цифр, поскольку он, наверное, наиболее понятен людям, не связанным с дизайном. Или воспользуйтесь препроцессором CSS, например Sass, и создайте переменные для использования разработчиками вашей команды. Переменные должны появиться в CSS, но пока их нет.

Если используются настройки прозрачности и градиентов, следует выбрать синтаксис `hsla()` или `rgba()`. А если нет, можно воспользоваться тем синтаксисом, который для вас наиболее удобен.

Мобильные и все остальные современные браузеры поддерживают все эти синтаксисы. И у каждого из нас есть предпочтения. Определитесь со своими предпочтениями — неважно какими — и строго их придерживайтесь.

Единицы измерения, используемые в CSS

Многие значения свойства являются ключевыми словами, уникальными для этого свойства. Ключевые слова, являющиеся уникальными (или не совсем уникальными) для свойства, будут описаны в рамках рассмотрения их свойств в следующих главах. Другие значения, такие как только что рассмотренные цвета, могут служить значениями для множества различных свойств.

Мы изучили практически все, что нужно знать о цветовых значениях, но цвета не являются единственным типом значений, общих для множества свойств. Есть также длина, параметры времени, частоты и углов.

Значения длины

К понятиям длины относятся как относительная, так и абсолютная длина. Краткая справка по всем типам значений длины дана в табл. 8.2.

Таблица 8.2. Единицы длины, используемые в CSS[1]

Единица измерения	Значение
em	Размер, задаваемый относительно размера шрифта родительского элемента
ex	Размер, задаваемый относительно высоты символа *x* в нижнем регистре
ch	Размер, задаваемый относительно размера символа 0 (нуль)
rem	Размер, задаваемый относительно размера шрифта корневого элемента
vw	Размер, задаваемый относительно ширины окна просмотра: ширина окна просмотра составляет 100 vw
vh	Размер, задаваемый относительно высоты окна просмотра: высота окна просмотра составляет 100 vh
vmin	Равно наименьшему из двух значений vh или vm
vmax	Равно наибольшему из двух значений vh или vm
px	Размер, задаваемый относительно разрешения экрана, а не размера окна просмотра, обычно равен одной точке, или 1/72 дюйма
in	Дюйм
cm	Сантиметр
mm	Миллиметр
pt	Пункт, или 1/72 дюйма
pc	Пика, или 1/12 пункта
%	Размер, задаваемый относительно родительского элемента, обычно определяется с помощью self или другого элемента, определенного свойством

[1] Браузерами поддерживаются все значения, за исключением vmax и ch, которые не поддерживаются в Android и Opera, и vmax, которое не поддерживается в Safari.

Наиболее часто в CSS используются такие типы значений, как пикселы и проценты. Очень эффективны в использовании и новые единицы измерения длины, такие как rem, vh, vw, vmin и vmax, особенно при разработке, выполняемой для множества устройств неизвестных размеров.

Одним из понравившихся мне новых свойств является единица измерения rem. Эта единица была представлена в CSS3 и означает root em. В то время как единица измерения em задает размер относительно размера шрифта родительского элемента, что может привести к неразберихе, единица измерения rem задает размер относительно размера шрифта корневого элемента — в нашем случае это `<html>`. Определяя размер единственного шрифта для корневого элемента, можно определить все единицы измерения rem в виде процентного отношения или относительно размера этого шрифта. Единица измерения rem поддерживается всеми мобильными браузерами, начиная с IE9, iOS 4 и Opera 12 (и всегда поддерживалась на Android).

ПРИМЕЧАНИЕ

Учтите, что значение нулевой длины (zero-length), или значение null, для любого определения длины может быть записано как 0. Все остальные типы значений требуют указания для нуля единицы измерения. Например, для нулевого значения угла в градусах нужно указать 0deg.

Пикселы могут считаться как относительной, так и абсолютной единицей измерения. Они являются относительной единицей измерения, поскольку измерения выполняются на основе разрешения монитора или экрана, а не на основе параметров окна просмотра. Но пикселы могут считаться и абсолютной единицей измерений, поскольку значения длины, заданные в пикселах, являются неизменными — они могут увеличиваться только посредством zoom-функций.

У изображений, таких как фотографии в JPEG- или GIF-формате, имеются абсолютная ширина и высота, определенная в пикселах. Увеличение или уменьшение размера изображений этих типов с помощью CSS или посредством атрибутов `width` и `height` тега `image` будут искажать изображение.

Некоторые свойства, ожидающие значения длины, могут также воспринимать значения в виде ключевых слов `auto` и `inherit`:

```
p {
  height: auto;
  font-size: inherit;
}
```

dpi, dpc, dppx

Изначально размер окна просмотра iPhone составлял 480 × 320 пикселов. i-устройства с высоким разрешением, такие как iPhone 4 и iPod touch 4G, физически имеющие такой же размер 320 × 480, обладают экраном 960 × 640. Окно просмотра iPad существенно больше — 1024 × 768 пикселов, а при смене ориентации — 768 × 1024 пиксела. Экран iPad третьего поколения имеет еще большее количество точек на дюйм (DPI) и разрешение 2048 × 1536. Safari и Chrome для настольного компьютера, браузеры для планшетов Nokia, смартфонов на основе OpenMoko, Android и другие WebKit-браузеры работают с разными размерами. И даже если будет возможность измерить

размер устройства, то, когда речь идет об устройствах с различными показателями DPI, уже нельзя говорить о том, что пиксел как единица измерения равен реальному пикселу устройства. Например, iPhone 4 может иметь высокий показатель DPI, следовательно, разрешение позволяет создать более привлекательную картинку, но изображения, отправляемые на устройство в качестве картинки переднего плана или фона, по-прежнему определяются в ожидаемых вами пикселах, как будто показатель DPI такой же, как на устройствах с менее высоким разрешением. В табл. 8.3 показаны различные единицы измерения разрешений и их значения.

Таблица 8.3. Единицы измерения разрешений экрана

Единица измерения	Значение
dpi	Разрешение в точках на дюйм
dpc	Разрешение в точках на сантиметр
dppx	Разрешение в точках на пиксел

С появлением устройств с различным разрешением экранов CSS3 обеспечил нас единицами измерения разрешений для включения значений в этих единицах в медиазапросы. Мы можем предназначать различные изображения для различных разрешений экрана на основе единиц измерения dpi, dpc или dppx.

CubeeDoo

Если вернуться к нашему приложению, то в нем мы использовали px для изображений и фоновых изображений, rem — для шрифтов и vh и vm — для задания ширины полей ввода. Нам удалось создать целое приложение без импортирования единого немасштабируемого изображения. Если бы использовались изображения, то в медиазапросах, предназначенных для обслуживания изображений с высоким разрешением, пришлось бы для устройств с более высоким разрешением использовать единицу измерения ddpx:

```
@media
only screen and (-webkit-min-device-pixel-ratio: 2),
only screen and (   min--moz-device-pixel-ratio: 2),
only screen and (   -o-min-device-pixel-ratio: 2/1),
only screen and (        min-device-pixel-ratio: 2),
only screen and (           min-resolution: 192dpi),
only screen and (            min-resolution: 2dppx) {

  /* задание стилей для устройств с высоким разрешением */

}
```

Значения углов, времени и частоты

Для некоторых новых свойств CSS3, например для преобразований и анимации, одних только единиц измерения длины не хватает. Нужно изучить и освоить также способы задания углов, времени и частоты. Эти измерения уже использовались

в звуковых таблицах стилей, но теперь, когда браузер поддерживает перемещения, преобразования и анимацию, углы и время приобрели значимость и для экрана. Единицы измерения углов, времени и частоты показаны в табл. 8.4 и более подробно рассмотрены в следующих подразделах.

Таблица 8.4. Единицы измерения углов, времени и частоты

Единица измерения	Значение
deg	Градус
grad	Градиан (град)
rad	Радиан
turn	Оборот
ms	Миллисекунда
s	Секунда
Hz	Герц
kHz	Килогерц

Исходным значением для длины, угла, времени и частоты является 0, и все значения интерпретируются как числа с плавающей точкой. При записи любой из этих единиц измерения нужно обязательно включать соответствующее сокращение из табл. 8.4. В отличие от единиц измерения длины, пропуск единицы измерения углов, времени и частоты в CSS недопустим, и такое объявление будет проигнорировано.

Изначально все эти единицы измерения частоты и времени, за исключением новой единицы измерения turn, были введены в качестве звуковых значений. Единицами измерений, использовавшимися для звуковых таблиц стилей, являлись углы, указанные в rad (радианах), deg (градусах) или grad (градианах). Частота указывалась в Hz (герцах) или kHz (килогерцах). Время указывалось в ms (миллисекундах) или s (секундах).

ВНИМАНИЕ

При записи любой из этих единиц измерения нужно использовать соответствующее сокращение. В отличие от единиц измерения длины, пропуск единицы измерения углов, времени и частоты приведет к ошибке, и объявление будет проигнорировано.

Единицы измерения углов

Единицами измерения углов являются градусы, градианы, радианы и обороты. Мы представим здесь эти единицы, но в примерах будем применять только градусы, поскольку они более понятны далеким от математики «ботаникам» вроде меня!

Углы

Диапазон углов составляет от 0 до 360°, и оба этих значения равны друг другу. Положительные углы отсчитываются по часовой стрелке, а отрицательные — против нее. Например, значение `-90 deg` равно четверти окружности против хода часовой стрелки и задает поворот влево. А значение `90 deg` задаст поворот на 90° по часовой стрелке.

Часть кода CSS для чтения рис. 8.2:

```
.image1, image5 {
  -webkit-transform: rotate(-5deg);
  -ms-transform: rotate(-5deg); /* для IE9 */
  transform: rotate(-5deg);
  }
.image2, image4 {
  -webkit-transform: rotate(7deg);
  -ms-transform: rotate(7deg);
  transform: rotate(7deg);
  }
```

Рис. 8.2. Поворот элементов на небольшие углы может создать интересные эффекты без потери удобочитаемости

Грады

Град, или градиан, эквивалентен 1/400 полной окружности. Как и в случае с градусами, положительное значение града будет отсчитываться по часовой стрелке, а отрицательное — против нее. 100 градов будут представлены прямым углом 90° (рис. 8.3).

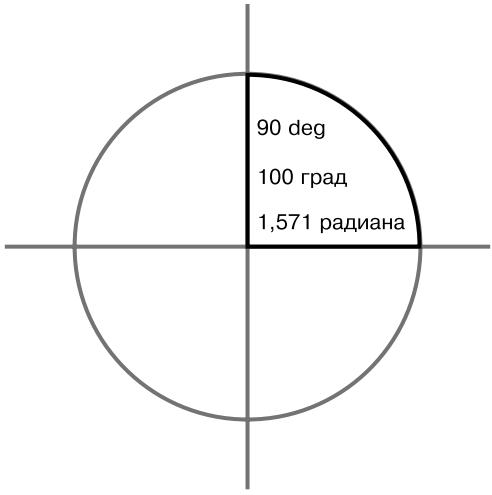


Рис. 8.3. 90° — то же самое, что 100 градов и 1,571 радиана

Рады

Рад, или радиан, равен 180°/π, или примерно 57,3°. Окружность составляет 2π радиана. Угол в 1 радиан на окружности соответствует дуге, длина которой равна радиусу окружности. Значение `1.570796326794897 rad` равно 100 град или 90 deg.

Обороты

Оборот равен 360 deg. Например, 2 turn = 720 deg. Учтите, что слово turn пишется здесь в единственном числе и между числом и названием единицы измерения нет пробела. Следующие строки являются равнозначными:

```
transform: rotate(900deg);
transform: rotate(2.5turn);
```

Время

Единицы измерения времени объяснить значительно проще, чем радианы! Существуют две единицы измерения: секунды (s) и миллисекунды (ms). 1000 мс составляют 1 с. Форматом значения времени является число, за которым следует `s` для секунд, или `ms` для миллисекунд. Как и все прочие единицы измерения, не относящиеся к длине, они предполагают обязательное включение `s` или `ms`, даже при нулевом значении. Следующие строки являются равнозначными:

```
animation-duration: 0.5s;
animation-duration: 500ms;
```

Частота

Значения частоты используются в звуковых (или речевых) каскадных таблицах стилей. Для этих значений существуют две единицы измерения: Hz (герцы) и kHz (килогерцы). `1000Hz = 1kHz` (регистр символов роли не играет). Частота может использоваться для изменения высоты голоса при чтении текста. Низкая частота соответствует басу, а высокая — дисканту.

В следующем фрагменте кода CSS низким голосом, например низким мужским голосом, будут озвучиваться слова абзацев с аргументом `class="low"`, и звуковое оформление изменится на высокий голос, когда встретится цитата с классом `squeal`. Следующие две строки отличаются друг от друга:

```
p.low { pitch: 105Hz; }
q.squeal {pitch: 135Hz;}
```

CubeeDoo. В нашей игре углы CSS используются для переворачивания карт, и этот момент выбирается как с помощью CSS, так и с помощью JavaScript. Таймер игры оперирует секундами и реализован на JavaScript. Плавное превращение лицевой стороны карты в тыльную и наоборот производится поворотом карт на 180 deg за 200 ms. Как это делается, будет показано в следующей главе с помощью анимации и преобразования.

Как избежать путаницы со сторонами прямоугольного блока при кратком объявлении свойств и значений

Мы уже рассмотрели большинство значений, не привязанных к конкретному свойству. Но в CSS есть еще одна особенность, которую нужно рассмотреть до перехода к углубленному изучению свойств CSS3. Здесь речь пойдет о таком понятии, как краткая запись, и о порядке следования в краткой записи значений для сторон прямоугольного блока.

В CSS имеется несколько кратких записей свойств. Они бывают двух типов: один тип позволяет разработчикам определять верх, низ, правую и левую стороны одним значением, а другой позволяет определять несколько обычно связанных друг с другом свойств из модуля CSS в одном вызове.

Например, вместо записи:

```
.sameValues {
  padding-top: 3px;
  padding-right: 3px;
  padding-bottom: 3px;
  padding-left: 3px;
}
.twoValues {
  padding-top: 3px;
  padding-right: 6px;
  padding-bottom: 3px;
  padding-left: 6px;
}
.threeValues {
  padding-top: 3px;
  padding-right: 6px;
  padding-bottom: 12px;
  padding-left: 6px;
}
.fourValues {
  padding-top: 3px;
  padding-right: 6px;
  padding-bottom: 9px;
  padding-left: 12px;
}
```

можно написать:

```
.sameValues {
  padding: 3px;
}
.twoValues {
  padding: 3px 6px;
}
.threeValues {
```

```
  padding: 3px 6px 12px;
}
.fourValues {
  padding: 3px 6px 9px 12px;
}
```

Учтите, что при написании кратких форм порядок их следования может играть весьма важную роль. Иногда, особенно это касается свойств CSS3 (что будет показано в главе 9), порядок следования будет важен при определении связанных друг с другом свойств в кратком формате. А иногда этот порядок в краткой записи, определяющей несколько взаимосвязанных свойств, не будет играть никакой роли. В данном примере краткие объявления, определяющие четыре стороны прямоугольного блока, имеют весьма характерный и несколько запутанный порядок, затрудняющий понимание.

Если присутствует только одно значение, то оно будет назначено всем четырем сторонам. Если указаны два значения, то первое из них определяет верх и низ, а второе — левую и правую стороны. Если значения три, то первое из них задает верх, второе относится к левой и правой сторонам, а третье — к низу. Мнемоническим правилом для запоминания последовательности может послужить английское слово TRouBLe (беспокойство) с выделенными буквами, обозначающими верх (T — top), правую сторону (R — right), низ (B — bottom) и левую сторону (L — left).

Для тех, кому проще учиться на примерах, в табл. 8.5 подробно показано, что будет представлено при задании в краткой записи одного, двух, трех и четырех значений.

Таблица 8.5. Стороны, на которые оказывается влияние при задании одного, двух, трех и четырех значений, и четыре краткие записи значений свойств

Пример	Значение	Порядок
3px	Все четыре стороны имеют одинаковые значения	TRBL (T — верх; R — правая сторона; B — низ; L — левая сторона)
2px 4px	Верх и низ: 2px Правая и левая стороны: 4px	TB RL
3px 1px 5px	Верх: 3px Правая и левая стороны: 1px Низ: 5px	T RL B
1px 2px 3px 4px	Верх: 1px Правая сторона: 2px Низ: 3px Левая сторона: 4px	T R B L

Следует учесть, что в этом правиле есть исключение: когда для CSS-свойства `background-position` задаются два значения, используется порядок LR TB, а не TB LR.

Краткие формы записи для определений некоторых, обычно взаимосвязанных свойств, за некоторыми исключениями, не касаются порядка следования значений. Большинство кратких форм позволяет сэкономить много времени на наборе кода. Например:

```
.myClass {
  border: 1px solid #ff0000;
}
```

можно также записать следующим образом:

```
.myClass {
  border-width: 1px;
  border-style: solid;
  border-color: #ff0000;
}
```

что, в свою очередь, является краткой формой для следующего кода:

```
.myClass {
  border-top-width: 1px;
  border-top-style: solid;
  border-top-color: #ff0000;
  border-right-width: 1px;
  border-right-style: solid;
  border-right-color: #ff0000;
  border-bottom-width: 1px;
  border-bottom-style: solid;
  border-bottom-color: #ff0000;
  border-left-width: 1px;
  border-left-style: solid;
  border-left-color: #ff0000;
}
```

Без использования кратких форм описание границы прямоугольной области заняло бы 12 строк. Краткая форма записи свойства `border` позволяет задать границу прямоугольной области одной строкой, используя меньшее количество символов в одной строке, чем в большинстве развернутых объявлений типа «свойство — значение».

Когда порядок следования значений свойств в краткой форме записи будет иметь какое-то значение, я специально скажу об этом. Я упомянула здесь об этом, чтобы вы взяли это заявление себе на заметку и не думали, что я просто пытаюсь помочь вам избавиться от бессонницы.

Резюме

В этой короткой главе было рассмотрено более 60 значений свойств, но если подсчитать все возможные цветовые значения, которые могут быть созданы, их получится более миллиона. Мы изучили различные типы цветовых значений CSS, добавленные в CSS3 новые цветовые значения и единицы измерения углов и длины. Также был рассмотрен порядок следования значений, определяющих стороны прямоугольной области, что является весьма запутанной, но жизненно важной темой. Теперь можно смело переходить к изучению свойств, с которыми могут быть связаны эти и другие типы значений. Веселье начинается!

9 CSS3: модули, модели и изображения

В отличие от CSS2, спецификация CSS3 была поделена на модули. За счет этого у W3C появилась возможность работать над разными модулями с разной скоростью, в результате чего одни модули уже находятся на уровне рекомендации (Recommendation level), а другие движутся в направлении окончательных рекомендаций в менее высоком темпе.

В спецификации CCS3 более 20 модулей, и каждый заслуживает отдельной главы (если не двух-трех глав). К сожалению, мы не в состоянии рассмотреть все модули и в данной главе коснемся тех тем CSS, которые имеют отношение к внешнему виду CubeeDoo. Чтобы включить в круг наших интересов ряд дополнительных свойств, мы также рассмотрим возможность изменения внешнего вида исходного экрана настроек смартфона iPhone.

Внешний вид исходных приложений iPhone может быть задан с помощью простых таблиц CSS. При этом можно будет воспользоваться такими хорошо известными и поддерживаемыми спецификацией CSS 2.1, а также более ранними спецификациями свойствами, как `border-radius` (скругление границ), свойствами фона, градиентами, `text-shadow` (текстовые тени), `box-shadow` (тени прямоугольных блоков), `background-size` (размеры фона), `text-overflow` (настройки вариантов отображения текста при переполнении прямоугольного блока). Затем эти свойства будут применены к CubeeDoo.

Отказ от изображений означает, что исходный внешний вид приложения iPhone можно создать без дополнительных HTTP-запросов и необходимости использования графической программы. Поскольку мы используем CSS, то, как только руководитель проекта примет решение об изменении цветовой схемы вашего приложения, вы сможете сделать это, не открывая программу редактирования изображения.

CSS3 позволяет создавать сайты с непривычным для вас объемом кода и количеством изображений, а чем меньше изображений, тем меньше HTTP-запросов, что повышает производительность. CSS3 разрешает также иметь несколько фоновых изображений для одного элемента, что помогает сократить количество DOM-узлов, необходимых для формирования внешнего вида сайта. Сокращение количества DOM-узлов повышает производительность, особенно когда речь заходит о перекомпоновке страницы и потреблении памяти.

Одни свойства появились только в CSS3, другие поддерживались годами. Но все же следует заметить, что применение CSS3 вместо изображений только потому, что использование некоторых из свойств этой спецификации позволяет сэкономить на количестве HTTP-запросов, не всегда является лучшим решением. Нужно сопоставить экономию на обслуживании HTTP-запросов с тем временем, которое необходимо браузеру для вычисления и прорисовки CSS-графики, и учесть ограничения памяти, присущие мобильным устройствам. На создание некоторых CSS-эффектов, например очень больших радиальных градиентов, задействуется много памяти, что может замедлить работу браузера, если заставлять устройство постоянно вести прорисовку в памяти.

Чем меньше кода, тем выше производительность и проще обслуживание сайта. Но не все свойства CSS хорошо работают на устройствах с ограниченным объемом памяти. При рассмотрении рискованных тем CSS я буду объяснять влияние свойств на производительность и рассматривать приемы, позволяющие избегать замедления работы или сбоев пользовательского браузера.

Поскольку некоторые свойства CSS3 и HTML5 могут снижать производительность работы устройства, я не стану полностью отказываться от изображений. В главе 11 будут рассматриваться несколько фоновых изображений и изображений границ и две функции, использующие изображения для быстрого и простого создания кнопок и фонов.

А вот модули речевых синтезаторов, страничных сред или ruby-модули рассматриваться не будут, потому что они предназначены соответственно для систем голосового чтения текста, принтеров и сайтов на азиатских языках[1]. А преобразования, переходы и анимация будут рассмотрены в следующей главе.

Хотя блочная модель CSS предшествовала появлению CSS3, ей тоже будет уделено время. Блочная модель CSS является основой разметки веб-страниц с момента создания CSS, поэтому очень важно как следует в ней разобраться.

Свойства блочной модели CSS

Перед тем как перейти к изучению новых свойств и значений CSS3, важно понять, что такое блочная модель и свойства, которые ее поддерживают. Все современные мобильные браузеры поддерживают и всегда будут поддерживать свойства границ (border), полей (margin) и отступов (padding):

- `border-bottom`;
- `border-bottom-color`;
- `border-bottom-style`;
- `border-bottom-width`;
- `border-color`;

[1] Если захочется заняться CSS-модулями speech, paged media и ruby, то с W3C-спецификациями можно ознакомиться по адресам: http://www.w3.org/TR/css3-speech/, http://www.w3.org/TR/css3-page/ и http://www.w3.org/TR/css3-ruby/

- border-left;
- border-left-color;
- border-left-style;
- border-left-width;
- border-right;
- border-right-color;
- border-right-style;
- border-right-width;
- border-style;
- border-top;
- border-top-color;
- border-top-style;
- border-top-width;
- border-width;
- margin;
- margin-bottom;
- margin-left;
- margin-right;
- margin-top;
- padding;
- padding-bottom;
- padding-left;
- padding-right;
- padding-top.

Границы

Для установки границ любых выводимых на экран элементов могут использоваться свойства границ и краткая форма их задания border. Значения этой краткой формы могут использоваться для задания стиля верха, низа, правой и левой стороны элемента. Развернутые формы записи свойств могут использоваться для задания стиля отдельного свойства для отдельно взятой стороны (верха, низа правой или левой стороны). Например, выражение border-style: dotted; установит границу для всех четырех сторон элемента в виде пунктирных линий, а выражение border-style-right: dashed; установит границу в виде пунктирной линии только для правой стороны.

О границах нужно знать следующее.

- Для отображения границы нужно использовать свойство border-style.
- Для работы свойства border-image требуется свойство border-style (см. главу 11).

- В соответствии с моделью блочного элемента, принятой W3C, при объявлении ширины и/или высоты элемента ширина левой и правой границ будет добавлена к ширине элемента, а высота верхней и нижней границ — к его высоте. Это неудобство может быть устранено с помощью свойства box-sizing, рассмотренного в одном из следующих разделов.

ПРИМЕЧАНИЕ

При использовании краткой формы записи без задания стиля границы не обойтись, а вот объявлять ширину и цвет не обязательно, и если их опустить, будут использованы значения по умолчанию.

border-style

Свойство border-style задает стиль четырех границ элемента.

В CSS3 нет новых значений свойства border-style, зато есть различные способы задания стиля границ с помощью свойств border-radius и border-image, которые будут рассмотрены в следующих разделах.

Значения для свойства border-style содержат ключевые слова dashed, dotted, double, groove, hidden, inset, none, outset, ridge и solid.

При задании значения стиля hidden, как и none, граница не выводится и не занимает пространства в блочной модели. Если нужна прозрачная граница, занимающая пространство в блочной модели, выберите значение стиля границы, отличное от none или hidden, и установите для цвета значение transparent. Значение hidden реально применяется только при определении стиля границ элемента таблицы.

Устаревшие WebKit-браузеры игнорируют задание свойства цвета границы border-color, когда ее вывод выполняется при заданных стилях inset, outset, groove и ridge. В новых версиях WebKit эта проблема устранена.

Чтобы граница была видна на экране, свойству border-style нужно присвоить значение, задающее ее видимость, поскольку по умолчанию свойству будет присвоено значение отсутствия — none. Если нужно отменить границу элемента, присвойте свойству border-style исходное значение none.

Чтобы заработало свойство border-image, нужно придать границе видимость, о чем пойдет речь в главе 11.

border-color

Свойство border-color позволяет определять цвет границы того элемента, для которого она устанавливается. Можно использовать любые типы значений цвета CSS, рассмотренные в разделе «Цветовые значения CSS» главы 8. Значением по умолчанию является текущий цвет — currentColor: если объявлено значение для свойства border-style, но не определено свойство border-color, у границы будет цвет текста, или currentColor.

Создание треугольников с помощью CSS. Изящный трюк по созданию треугольников состоит в задании границы. Чтобы блок текста выглядел как речевая

выноска со сгенерированным содержимым, создайте блок с нулевыми высотой и шириной, установите для трех цветов границы прозрачность, а граница четвертой стороны создаст треугольник:

```
blockquote {
  background-color: green;
  position: relative;
  color: white;
  padding: 15px 25px;
  margin: 10px 10px 0;
}
blockquote:after {
  border: 15px solid;
  border-color: green transparent transparent;
  top: 100%; left: 10px;
  width: 0; height: 0;
  position: absolute;
  content: '';
}
```

В предыдущем фрагменте кода я создала псевдоэлемент, не имеющий ни ширины, ни высоты и показывающий только одну из четырех сторон, имеющих границу, что и приводит к появлению треугольника. Ссылка на этот пример имеется в онлайн-ресурсах к данной главе. Результат выполнения кода показан на рис. 9.1.

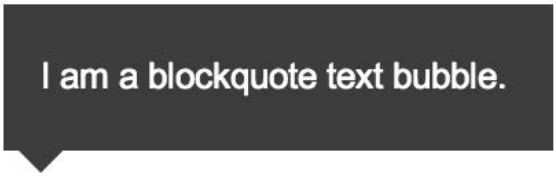


Рис. 9.1. Речевая выноска с хвостом, созданным с помощью верхней границы сгенерированного содержимого

А как бы вы сделали, чтобы хвост речевой выноски появлялся справа или слева от содержимого, а не снизу?

border-width

Свойство `border-width` устанавливает ширину четырех границ элемента. Значениями свойства `border-width` служат ключевые слова `thin`, `medium`, `thick` и `inherit`, а по умолчанию используется значение `medium`. Можно также воспользоваться любыми значениями, задающими длину (px, em и т. д.), рассмотренными в главе 8.

Чтобы получить границу для треугольника, показанного на рис. 9.1, мы создали границу толщиной 15 пикселов для сгенерированного содержимого, не имеющего ни высоты, ни ширины. Видимой была сделана только верхняя граница путем задания ей зеленого цвета и прозрачности — остальным сторонам.

Свойство ширины границы `border-width` играет для свойства `border-image` важную роль: если в краткой форме записи свойства `border-image` не объявить ту часть, которая относится к заданию свойства `border-width`, то свойство `border-image` унаследует значения `border-width`.

Ширина границы важна также для понимания внешних теней и теней, рисуемых внутри блока. Используя толстые границы и тени, можно получить ряд весьма интересных эффектов, и мы еще вернемся к этому в данной главе.

Блочная модель CSS

Все элементы HTML создаются на экране в виде прямоугольного блока. Блочная модель CSS определяет прямоугольник (или блок), составленный из полей, границ, отступов и содержимого, являющегося сутью каждого элемента. Блочная модель позволяет размещать элементы на странице, определяя ширину содержимого, пустое пространство между содержимым элемента и его границей и пустое пространство между несколькими элементами.

Как показано на рис. 9.2, компоненты блочной модели включают содержимое (content), отступы (padding), границы (border) и поля (margin).

- `Content` — содержимое блока, где появляются текст и изображения.
- `Padding` — отступ, пустое пространство между содержимым и границей. Если у элемента есть фоновое изображение или цвет, область отступа будет по умолчанию иметь в качестве фона этот цвет или это изображение[1].
- `Border` — граница, окружает отступ и находится внутри поля. Граница начинается там, где заканчивается отступ. Они не накладываются друг на друга. Граница занимает место в блочной модели. Если у элемента есть фоновое изображение или цвет, а граница выведена пунктиром или же полностью или частично прозрачна, то по умолчанию фон будет показан сквозь границу.
- `Margin` — поле, пустое пространство за пределами границы. Поле является прозрачным. Если у элемента есть фоновый цвет или изображение, то оно содержится в пределах границы и не будет показано сквозь область поля.

Чтобы задать ширину и высоту элемента, нужно иметь полное представление о работе блочной модели. В традиционной блочной модели объявленная ширина и высота, заданная элементу, применяются только для области содержимого. А ширина и высота, объявляемые вами, включают отступы и границы.

ПРИМЕЧАНИЕ

Ширина = левая граница + левый отступ + ширина + правый отступ + правая граница.
Высота = верхняя граница + верхний отступ + высота + нижний отступ + нижняя граница.

Обратите внимание на то, что на рис. 9.2 область поля не имеет цвета. Причина в том, что ширина этой области не входит в ширину и высоту, вычисляемые для свойств ширины и высоты. Поле, если его значение больше 0, занимает свое собственное место.

[1] Чтобы изменить это поведение по умолчанию, можно воспользоваться свойствами background-clip и background-origin.

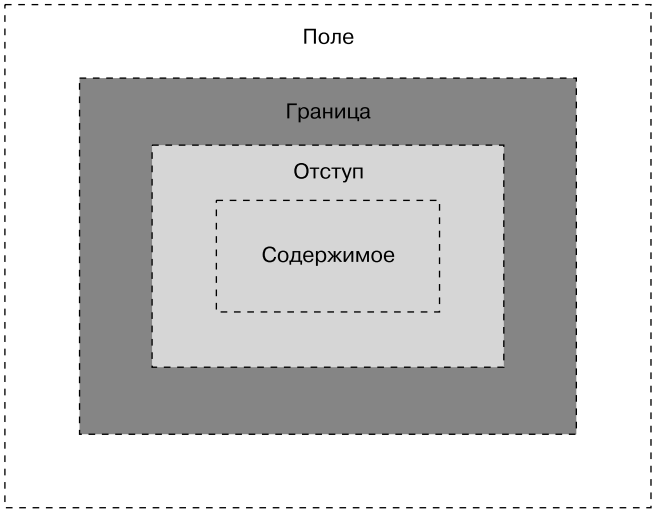


Рис. 9.2. Блочная модель W3C

В вашей разметке размер элемента образован размерами содержимого, отступа, границы и поля[1].

ПРИМЕЧАНИЕ

Общая ширина = левое поле + левая граница + левый отступ + ширина + правый отступ + правая граница + правое поле.

Общая высота = верхнее поле + верхняя граница + верхний отступ + высота + нижний отступ + нижняя граница + нижнее поле.

box-sizing

`box-sizing` — одно из лучших, если не лучшее свойство, которое дано нам спецификацией CSS3, особенно если вы полагаете, что Microsoft правильно понимает блочную модель, а W3C ошибается. Перед объявлением 100%-ной ширины элемента мы избегали объявления границ или отступов для этого элемента, поскольку он в итоге стал бы шире своего контейнера. Свойство `box-sizing` решает эту проблему — нужно просто установить для свойства `box-sizing` значение `border-box`:

```
.box {
  width: 100%;
  padding: 10px;
  border: 1px solid currentColor;
  -webkit-box-sizing: border-box; /* для устаревших Android (3.0) */
  -moz-box-sizing: border-box; /* для Firefox */
  box-sizing: border-box;
}
```

[1] Если два элемента следуют друг за другом, их примыкающие поля накладываются друг на друга.

ПРИМЕЧАНИЕ

Если нужно, чтобы 100 % были 100 %, несмотря на отступ и границу, можно сымитировать блочную модель IE6/IE7, установив для свойства box-sizing значение border-box.

Свойство `box-sizing` поддерживается, начиная с IE8, но во всех версиях Firefox и в мобильных WebKit-браузерах вплоть до iOS 4.3, Android 3.0 и BlackBerry 7, должно быть снабжено префиксом.

До появления полной поддержки `calc()` панацеей от этих бед будет `box-sizing: border-box`!

Поля. Еще одной частью блочной модели, которая кому-то может показаться запутанной, является эффект создания полей двух смежных элементов. Положительные значения полей не складываются друг с другом: если два примыкающих друг к другу элемента имеют поля с положительными значениями, расстоянием между ними будет большее из двух полей, а не сумма полей. Если же одна из этих границ будет иметь отрицательное значение, тогда расстоянием между элементами будет сумма положительного и отрицательного полей.

Изучение CSS3

Закончив с блочной моделью, можем перейти к некоторым весьма полезным свойствам CSS3. Начнем с создания внешнего вида приложения в стиле iPhone, используя для придания стиля только код CSS. Для примеров будет использоваться следующий код HTML:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <title>iPhone Look and feel</title>
  <meta name="viewport" content="width=device-width; initial-scale=1.0;"/>
</head>
<body>
<header>
  <nav>
    <ul>
      <li class="button cancel">Cancel</li>
      <li class="button done">Done</li>
    </ul>
    <h1>Languages</h1>
  </nav>
</header>
<article>
  <ul>
    <li lang="en-us">English</li>
    <li lang="fr-fr">Français</li>
    <li lang="es-es">Español</li>
    ...
  </ul>
```

```
</article>
</body>
</html>
```

Используя этот небольшой пример кода HTML и несколько правил CSS, мы создадим веб-страницу, похожую на страницу выбора языковых предпочтений в iPhone (рис. 9.3).

Рис. 9.3. Используя только CSS, без каких-либо изображений, мы собираемся создать внешний вид в стиле, присущем настроечному приложению iPhone

border-radius

Новыми в CSS3, но не новыми в мобильных браузерах являются скругленные углы, созданные с помощью свойства `border-radius`. Задание радиусов границ — быстрый и легкий способ усовершенствования пользовательского интерфейса без увеличения времени загрузки.

До появления возможности создания естественно выглядевших скругленных углов с помощью CSS-свойства `border-radius` веб-разработчики создавали скругленные углы, добавляя дополнительную разметку. Метод четырех углов требовал четырех добавленных элементов, у каждого из которых было фоновое изображение, абсолютно позиционируемое в каждом из четырех углов элемента. Метод раздвижных дверей — наиболее популярный — также требовал применения изображений и дополнительной разметки.

Хотя каждый из этих методов позволял создавать естественно выглядевшие скругленные углы, они требовали добавления к странице не пригодных ни для чего другого элементов в качестве привязок и контейнеров для фоновых изображений, а для этого нужно было создавать и загружать эти изображения. Для повышения производительности желательно сводить к минимуму количество HTTP-запросов, уменьшать время загрузки, нагрузку на канал передачи данных и количество DOM-узлов. Также можно не тратить попусту время на создание изображений, а затем на подгонку их под длину радиуса границы, цветовую схему или другие вносимые в дизайн изменения.

Для разработчиков самым большим недостатком решений, связанных с использованием изображений, является то, что любые изменения, вносимые в цвет границы, фоновый цвет, фон самого элемента, степень закругленности или размеры, потребуют создания новых изображений. Недостатком для пользователей (подводящим под всем этим общую черту) является то, что методы с использованием изображений снижают скорость загрузки и влияют на производительность портативных устройств с ограниченным объемом памяти. Свойство `border-radius` позволяет разработчикам добавлять скругленные углы вполне естественным образом, без каких-либо изображений углов! Еще одним преимуществом является отсутствие пикселизации при укрупнении изображения на экране!

Свойство `border-radius` является краткой формой записи следующих свойств:

```
border-top-right-radius:
border-bottom-right-radius:
border-bottom-left-radius:
border-top-left-radius:
```

и т. д.

Синтаксис для свойства `border-radius` имеет следующий вид:

```
border-radius: length{1,4} / length{1,4} /* краткая запись */
border-(top|bottom)-(left|right)-radius: length length /* развернутая запись */
```

В краткой форме записи значения перед слешем, если последний в нее включен, задают горизонтальные радиусы, а значения после слеша — вертикальные радиусы (рис. 9.4). В развернутой форме записи каждое из свойств может получить два значения: первое — для горизонтального, а второе (если будет указано) — для вертикального радиуса. При развернутом синтаксисе слеш не ставится. Если имеется только одно значение, вертикальный радиус будет таким же, как горизонтальный, благодаря чему будет создаваться симметрично скругленный угол со вторым значением, скопированным с первого. Если обе длины равны нулю, угол получается прямым без скруглений.

При задании четырех значений используется следующий порядок углов: верхний левый, верхний правый, нижний правый, нижний левый. Как при использовании системы TRBL для границ, если объявлены только два значения, то первое из них применяется к верхнему левому и нижнему правому, а второе — к верхнему правому и нижнему левому углам.

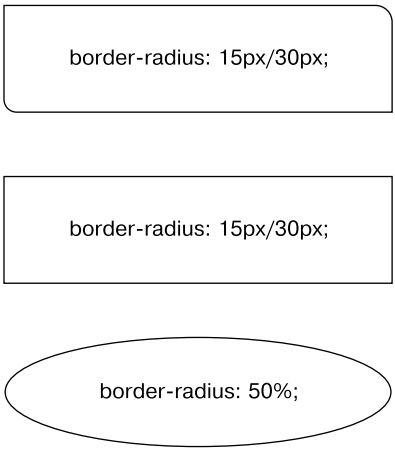


Рис. 9.4. Примеры работы свойства border-radius, включающие различные размеры радиусов и эллиптические формы

При нацеливании значений радиусов с помощью DOM используются следующие значения:

```
myObj.style.borderTopLeftRadius
myObj.style.borderTopRightRadius
myObj.style.borderBottomRightRadius
myObj.style.borderBottomLeftRadius
```

Учтите, что дефис (-) имеет в JavaScript особый смысл. Мы не хотим вычитать `radius` из `left`, `top` или `border`, что происходило бы при форме записи `border-top-left-radius`. В JavaScript CSS-свойства при добавлении в качестве свойств стилевого оформления DOM-узла записываются со смысловым разделением с помощью символов в разных регистрах. Тем самым, как говорится, соблюдается разделение представления и поведения и исключается использование JavaScript для изменения значений свойств CSS. Вместо этого нужно определять свои стили в стилевых блоках таблиц стилей и использовать JavaScript для изменения класса или состояния.

Использование border-radius для создания кнопок естественного вида для iPhone и в CubeeDoo

Применим знания на практике! Свойственные iPhone приложения (см. рис. 9.3) имеют ряд элементов со скругленными углами — кнопки Cancel (Отмена) и Done (Готово) и основное содержимое страницы. Полагаясь на возможности CSS3, можно элементы со скругленными углами создать любых размеров, цветов и радиусов кромок без использования изображений. Сфокусируемся на CSS для области содержимого:

```
1   article ul {
2     border: 1px solid #A8ABAE;
```

```
3     border-radius: 10px;
4     background-color: #FFFFFF;
5     width: 300px;
6     margin: 10px auto;
7   }
```

В строке 2 задается граница серого цвета толщиной 1 пиксел. В строку 3 включено свойство `border-radius`, чтобы добавить к границе радиус со значением 10 пикселов. Если вы работаете с Android 2.1 или iOS 3.2, можно непосредственно перед беспрефиксной формой записи свойства включить свойство с префиксом `-webkit-`. Но, может быть, вам и не нужно будет этого делать: устаревшие версии Android обрабатывают радиусы углов далеко не идеально, а отказ от задания скругленных границ с помощью объявления с префиксом позволит сэкономить немного памяти в не слишком производительных браузерах и избежать дефекта временного исчезновения скругленных углов при прокрутке страницы. В устаревших версиях Android ваше приложение без скругленных углов будет выглядеть иначе. Пользователи приложения могут и не заметить отсутствия скругленных углов, а вот сбой приложения они обязательно заметят. Поэтому я уже давно прекратила добавлять префиксы для объявления скругленных углов.

В строке 6 используем краткую форму записи свойства `margin` для перемещения неупорядоченного списка на 10 пикселов ниже панели навигации и центрируем область содержимого, устанавливая для левого и правого полей значения `auto`. Поскольку для свойства `margin` были объявлены два значения, первое будет предназначено для верха и низа (Top и Bottom), а второе — для левой и правой сторон (Left и Right). (Не понимаете? Перечитайте раздел «Как избежать путаницы со сторонами прямоугольного блока при кратком объявлении свойств и значений» главы 8.)

Так же задается стиль для элементов списка. Для них используется следующий код CSS:

```
8   article ul {
9     list-style-type: none;
10   }
11   article li {
12     line-height: 44px;
13     border-bottom: 1px solid #A8ABAE;
14     padding: 0 10px;
15   }
16   article li:last-of-type {
17     border-bottom: none;
18   }
```

Основная часть этого кода CSS должна быть вам хорошо знакома. Обратите внимание на псевдокласс `:last-of-type` в строке 16. Структурные селекторы были рассмотрены в главе 7. Используя блок `article li:last-of-type {border-bottom: none;}`, можно заставить браузер не включать нижнюю границу последнего элемента списка каждого элемента `<ul>`, находящегося в контейнере `<article>`.

Я не использую блок `article li:not(:last-of-type)` для настройки границ всех элементов, за исключением последнего, хотя могла бы. У нас есть нижняя граница самого элемента `<ul>`, и если не отказаться от нижней границы последнего элемента `<li>`, в нижней части каждого неупорядоченного списка появится двойная граница.

А где еще имеются скругленные углы? Их вполне можно добавить к кнопкам Done (Готово) и Cancel (Отмена) приложения iPhone, а также к кнопкам и картам CubeeDoo. Код для создания скругленных углов находится в файлах для главы.

Градиенты CSS

С помощью свойства радиуса границ мы смогли создать скругленные углы и добиться естественного внешнего вида области содержимого приложений, использовав всего несколько строк CSS и не добавляя ни HTML-элементов, ни изображений.

Кроме скругленных углов у кнопок Cancel (Отмена) и Done (Готово), есть градиентный фон. Можно добавить отдельное фоновое изображение для кнопки Cancel (Отмена) и фоновое изображение другого цвета для кнопки Done (Готово). Используя для границы радиус небольшой величины, мы можем создать кнопки, очень похожие на натуральные кнопки iPhone. Затем можно будет отправить другое фоновое изображение для панели навигации и еще одно — для фона всей страницы. Фактически у iPhone есть PNG-спрайт с кнопками, которые используются приложениями iOS в качестве изображений границ. И нам можно сделать то же самое. Но при использовании CSS3 все эти изображения, даже в виде единого спрайта, и связанные с их отправкой HTTP-запросы ни к чему. Естественная для приложений iOS разметка может быть полностью выполнена с помощью CSS, без импорта отдельного файла с расширениями `.svg`, `.gif`, `.webp`, `.png` или `.jpeg`.

Градиенты относятся не к свойствам, а к значениям и могут применяться везде, где используются изображения, включая свойства `background-images`, `list-style-images` и `border-images`. Они поддерживаются всеми современными браузерами, а в устаревших браузерах требуют указания различных префиксов.

Градиент является CSS-изображением, созданным браузером на основе определенных разработчиком цветов путем постепенной смены одного цвета на другой. Браузеры поддерживают линейные, радиальные и повторяющиеся градиенты. В модуль CSS-изображений четвертого уровня (CSS Images Level 4) были добавлены и конусообразные градиенты, но они пока не готовы для практического применения и здесь не рассматриваются.

Зато будет рассмотрен синтаксис, использующий префиксы производителей, поддерживаемый всеми браузерами, но необходимый только браузерам семейства WebKit. Так как версия CSS-градиентов без использования префиксов уже получила поддержку, ее нужно включить в свой код CSS в качестве исходного значения. Но нам по-прежнему приходится рассматривать синтаксис с использованием префиксов и весьма старый синтаксис WebKit, чтобы можно было поддерживать устаревшие мобильные устройства.

Типы градиентов: линейный или радиальный

Существуют два основных типа градиентов: линейный и радиальный. Чаще всего в код включаются линейные градиенты. Следовательно, значение свойства должно начинаться со следующих фрагментов:

```
background-image: -webkit-gradient(linear, /* Для очень старых версий WebKit[1] */
background-image: -webkit-linear-gradient( /* для Android, iOS вплоть до версии 6.1,
                                             BB10 */
background-image: linear-gradient( /* для iOS7, Chr26+, IE10+, FF 16+, O12.1+ */
```

А для радиальных градиентов потребуется включить следующие фрагменты:

```
background-image: -webkit-gradient(radial, /* Для очень старых версий WebKit */
background-image: -webkit-radial-gradient( /* для Android, iOS вплоть до версии 6.1,
                                             BB10 */
background-image: radial-gradient( /* для IE10, FF16, O12.1, Chr26, iOS7 */
```

В следующих примерах для второго варианта синтаксиса с префиксом будет применяться только префикс производителя WebKit, а в качестве третьего и последнего варианта синтаксиса будут использованы версии без префикса. Надеюсь, так будет легче разобраться с разметкой. В браузерах Mozilla или Opera[2] префиксы производителей больше не нужны, а в версиях Internet Explorer они никогда не использовались.

Радиальные градиенты

В данной книге радиальные градиенты рассматриваться не будут. При желании получить о них дополнительную информацию, ссылку на соответствующее руководство можно найти в онлайн-ресурсах к главе.

Эта книга посвящена разработке программ для мобильных устройств, а радиальные градиенты и мобильные устройства не всегда уживаются друг с другом. При создании радиального градиента он весь помещается в память. Если радиальный градиент небольшой, одноцветный, без прозрачности или ступенчатых переходов, объем затрачиваемой памяти будет небольшим. Но такие градиенты используются довольно редко. Линейные градиенты представляют собой небольшие изображения, которые выкладываются в памяти браузера, как плитки, а радиальные градиенты являются одним более крупным изображением, которое может занять много памяти и даже вызвать сбой мобильных браузеров на устройствах с ограниченным объемом оперативной памяти.

Слишком большие изображения выкладываются в памяти как плитки. Все устройства разные, но я с уверенностью могу предположить, что с изображениями,

---

[1] Исходный синтаксис градиентов WebKit использовался вплоть до появления Chrome 9, iOS Safari 4.3, Android 3.2 и BlackBerry 7.

[2] Примеры, имеющие префиксы -moz- и -o-, здесь не рассматриваются, поскольку современным браузерам они не нужны. Но вы можете включить их в свой код в том случае, если до сих пор поддерживаете Firefox 4–15 и/или Opera 11.1–12.0.

превышающими по размеру 1024 пиксела, все именно так и происходит. При использовании радиальных градиентов можно очень быстро дойти и до таких больших изображений.

Использованию радиальных градиентов на мобильных устройствах ничто не мешает. Нужно только учитывать возможное снижение производительности.

Линейные градиенты

Для задания линейного градиента, требующего указания префикса, используется следующий синтаксис:

```
-prefix-linear-gradient(<angle|keyterm>, <colorstop>, [<colorstop>,] <colorstop>)
```

Здесь `angle` — это угол пути распространения градиента в градусах. Вместо него может использоваться комбинация ключевых слов `top` или `bottom`, и/или `left` или `right`, определяющая начало линии градиента и как минимум один объявленный цвет с необязательными дополнительными цветами и позициями упомянутых цветов на пути распространения градиента.

Окончательный вариант синтаксиса линейного градиента имеет следующий вид:

```
linear-gradient([<angle>| to<keyword>], <colorstop>, [<colorstop>,] <colorstop>)
```

Основные отличия заключаются, во-первых, в использовании ключевого слова `to`, во-вторых, в характере влияния на градиент ключевых слов, задающих угол, и в-третьих, в значении, используемом для углов.

В синтаксисе с префиксом нужно показать, откуда исходит градиент, а в синтаксисе без префикса используется ключевое слово `to`, указывающее, куда направляется, а не откуда начинается градиент.

В синтаксисе с префиксом линия градиента начинается с точки, которая была бы концом гипотенузы прямоугольного треугольника, двумя другими вершинами которого были бы середина прямоугольного блока и ближайший угол.

Упрощенный беспрефиксный синтаксис отличает то, что в него добавлено ключевое слово `to`: теперь линия градиента идет от одной стороны или одного угла к противоположным стороне или углу, проходя через центр прямоугольного блока. Запутались? Не волнуйтесь. Я все объясню.

Углы и направления градиентов

Направление постепенного изменения линейного градиента управляется в CSS заданием угла (`<angle>`) или ключевого слова. Можно задать ключевое слово или угол в градусах, радианах, градах или оборотах.

Когда задается угол, путь градиента проходит через центральную точку фонового изображения под указанным углом. Но синтаксис с префиксом отличается от синтаксиса без префикса. В синтаксисе без префикса углы следуют по часовой стрелке и вверху находится значение `0`, а справа — значение `90`. В синтаксисе с префиксом углы следуют против часовой стрелки и значение `0` находится справа (рис. 9.5).

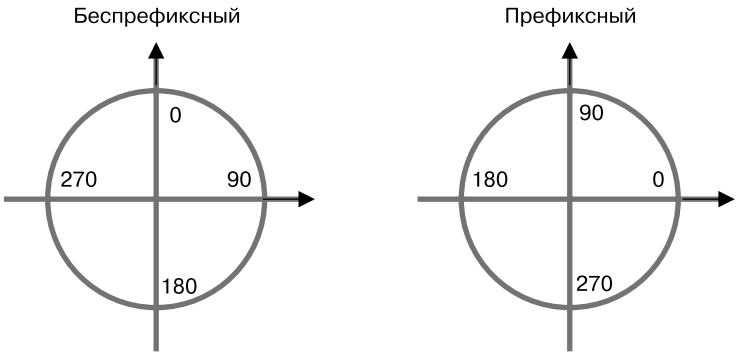


Рис. 9.5. Направление угла градиента при использовании экспериментального префиксного синтаксиса и окончательного беспрефиксного синтаксиса

При объявлении линейных градиентов сначала следует включать префиксные версии для устаревших браузеров. При использовании префиксной версии 0° находится справа и углы следуют против часовой стрелки, что может противоречить интуитивному представлению. Последней CSS-парой «свойство — значение» должно быть исходное значение без префикса. При отсутствии префикса точка 0° находится на севере, а положительные углы увеличиваются по часовой стрелке, поэтому 90° указывают на правую сторону. Линия градиента проходит через центр элемента под указанным углом.

Вместо использования углов можно объявить путь градиента с помощью ключевых слов. Ключевые слова являются названиями сторон: `top` (верх), `bottom` (низ), `left` (левая сторона) или `right` (правая сторона) — или углов, например: `top left` (верхний левый) или `bottom right` (нижний правый).

В синтаксисе с префиксом ключевые слова задают начальную точку пути градиента. Градиент будет брать начало в общей области, описанной ключевым словом, проходить через центр фонового изображения (`background-image`) к области противоположных стороны или угла. Например, при указании ключевого слова `top` линия градиента будет идти от центра верхней стороны к центру нижней стороны, проходя через центральную точку прямоугольного блока элемента, а при указании ключевого слова `top right` линия градиента будет идти от верхней правой области к нижней левой, проходя через центральную точку, то есть не обязательно будет проходить через сам угол.

Исходное CSS-объявление, не использующее префикс, должно включать в себя значение, предваряемое словом `to` (при использовании ключевых слов), указывающим, где линия градиента должна заканчиваться. Линия градиента будет начинаться с одной стороны или в одном углу, проходить через среднюю точку и заканчиваться на той стороне или на том углу, который был указан с помощью ключевого слова. Например, при указании `to top` линия градиента будет начинаться с центра нижней стороны и направляться к центру верхней стороны, проходя через центральную точку прямоугольного блока элемента, а при указании `to top right` линия градиента будет начинаться с нижнего левого угла, проходить через центральную точку и заканчиваться в верхнем правом углу.

Значением по умолчанию является top или 270deg для префиксной версии, и to bottom или 180deg в окончательной, беспрефиксной версии, которая будет выглядеть абсолютно так же, как показано на рис. 9.6. Чтобы градиент постепенно изменялся от черного к белому цвету по мере прохождения от верхней стороны к нижней, нужно написать любой из следующих вариантов кода:

```
/* префиксный */
-webkit-linear-gradient(#000000, #FFFFFF);
-webkit-linear-gradient(top, #000000, #FFFFFF);
-webkit-linear-gradient(270deg, #000000, #FFFFFF);

/* беспрефиксный */
linear-gradient(#000000, #FFFFFF);
linear-gradient(to bottom, #000000, #FFFFFF);
linear-gradient(180deg, #000000, #FFFFFF);
```

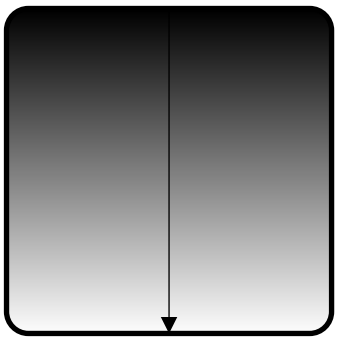


Рис. 9.6. Простой градиент и различные способы его объявления

Чтобы создать градиент, проходящий из верхнего левого в нижний правый угол (рис. 9.7), можно написать следующие две строки кода, аналогичные по назначению, но отличающиеся друг от друга по написанию:

```
/* от верхнего левого к нижнему правому углу */
-webkit-linear-gradient( top left, #000000, #FFFFFF);
linear-gradient(to bottom right, #000000, #FFFFFF);
```

Следует учесть, что при задании в префиксной версии угла 315deg или в беспрефиксной версии угла 135deg будут получены одинаковые градиенты, но они не будут создавать точно такие же градиенты, как при задании top left или to bottom right, которые не будут идентичны друг другу.

Префиксная версия с углом 315deg и беспрефиксная с углом 135deg создадут градиент, идущий от top left (верхнего левого угла) к to bottom right (к нижнему правому углу), если элемент имеет форму квадрата, но если фоновое изображение имеет форму вытянутого прямоугольника, от угла к углу будет направлен только тот градиент, который задавался условием to bottom right. Почему? При использовании угла путь градиента проходит через центр определенного фонового изображения под указанным углом, и при этом не важно, каково соотношение сторон у этого прямоугольника.

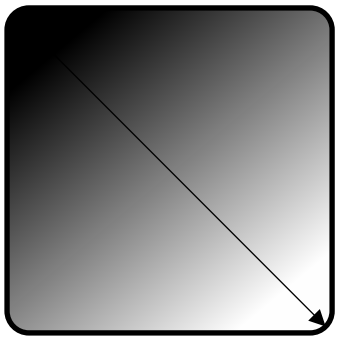


Рис. 9.7. Простой градиент, проходящий от верхнего левого к нижнему правому углу

ПРИМЕЧАНИЕ

По умолчанию путь градиента проходит через центр определенного элемента, поскольку по умолчанию фоновое изображение занимает 100% по ширине и длине этого элемента. При изменении позиции фона (background) или его размера (size) путь градиента уже не будет проходить через центр определенного элемента, а всегда будет проходить через центр определенного вами фонового изображения независимо от его размера или позиции.

В окончательном, или в беспрефиксном синтаксисе ключевые слова определяют, что путь градиента заканчивается в одном из углов после прохождения через центр фонового изображения. В префиксном, или устаревшем синтаксисе, если речь идет не о квадрате, путь градиента выходит за пределы краев прямоугольника, позволяя отображаться началу и концу градиента. Следовательно, когда используется префиксный вариант, градиент идет от угла к углу, только если прямоугольный блок в действительности является квадратом (рис. 9.8):

```
background-image: -webkit-linear-gradient(top left, #000000, #FFFFFF);
background-image: linear-gradient(bottom right, #000000, #FFFFFF);
```

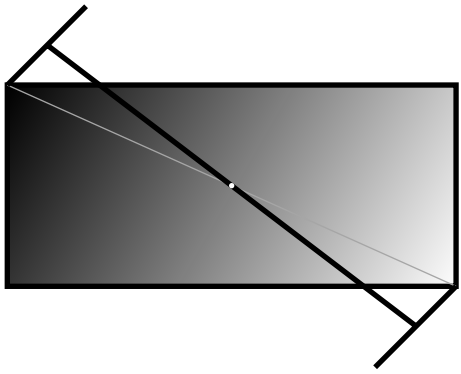


Рис. 9.8. Если заданы углы и фоновое изображение не является квадратом, то при использовании ключевых слов линии градиента у префиксной (показана черным) и беспрефиксной (показана серым) версий будут различаться

На рис. 9.8 черная линия показывает путь градиента, который начинается из верхнего левого угла и проходит через центр фонового изображения, показанный небольшим светлым квадратом в середине фигуры. Серым цветом показан путь градиента, который он проходит при поддержке беспрефиксного задания градиентов и указании угла `to bottom right` вместо `top left`. В префиксной версии конечные точки пути градиента выходят за пределы изображения и определяются как наиболее отдаленные от центра фонового изображения углы блока, где линия, проведенная перпендикулярно к линии градиента, будет пересекать угол прямоугольного блока в этом направлении.

Цвета градиента

Изучив порядок объявления угла градиента, можно перейти к объявлению цветов градиента. Мы создали линейный градиент, в котором цвет постепенно переходит от черного к белому. А что, если, продолжая весьма впечатляющий пример перехода от белого к черному, нам захочется сделать переход от сплошного цвета к сплошному цвету, показывая эти цвета с обоих концов примерно на 10% протяженности элемента? Мы легко справимся с этой задачей, если определим цветовые опорные точки.

Чтобы объявить цветовую опорную точку, нужно объявить цвет и позицию, в которой он заканчивается. Если первая опорная цветовая точка не имеет значения 0%, то первый цвет будет сплошным от 0% или 0 пикселов до первой цветовой опорной точки, где он начнет постепенно переходить к следующему цвету. То же самое произойдет и в том случае, если ваша последняя цветовая опорная точка установлена не на 100%.

Если позиция цветовой опорной точки не объявлена (а объявлен только цвет), такие точки будут равномерно распределены по всей ширине (или высоте) фонового изображения, причем первая точка будет располагаться на 0%, а последняя — на 100%.

Позиция может быть объявлена в виде процента от ширины фонового изображения или в виде абсолютной длины. Если используется фоновое изображение высотой 200 пикселов, то следующие четыре объявления будут вызывать одинаковый эффект (рис. 9.9):

```
-webkit-linear-gradient(top, #000000 10%, #FFFFFF 90%);
-webkit-linear-gradient(top, #000000 20px, #FFFFFF 180px);
linear-gradient(to bottom, #000000 10%, #FFFFFF 90%);
linear-gradient(to bottom, #000000 20px, #FFFFFF 180px);
```

Обратите внимание на то, что в примере, показанном на рис. 9.9, сверху и снизу на 10% высоты элемента перед постепенным переходом от черного к белому цвету цвета остаются сплошными. От верха до 10%-ной отметки изображение залито сплошным черным цветом. А от 90%-ной отметки до нижней кромки изображение имеет сплошной белый цвет. Постепенный переход от черного цвета к белому происходит на 80% высоты.

Цветовые опорные точки, в которых заканчивается сплошной цвет и начинается постепенный переход к другому цвету, показаны двумя стрелками. Но мы не ограничены применением только двух цветов. Для получения искомого эффекта

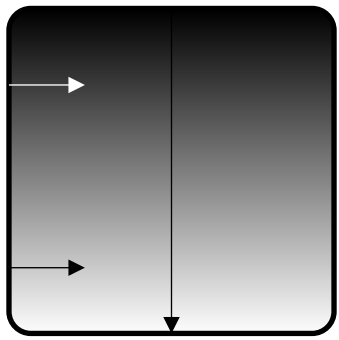


Рис. 9.9. Сплошными цветами закрашены промежутки от 0 до первой цветовой опорной точки и от второй цветовой опорной точки до края прямоугольного блока

можно добавлять столько цветовых опорных точек, сколько требуется (но при этом следует помнить, что можно — не значит нужно).

Иногда используется более двух цветов. Например, если хочется вдохнуть новую жизнь в сайт, созданный в 1996 году, придав ему впечатляющие свойства линейных градиентов 2013 года, можно создать для него радужный фон:

```
background-image: linear-gradient(
  red,
  orange,
  yellow,
  green,
  blue,
  purple);
```

Это можно записать и в следующем виде:

```
background-image: linear-gradient(
  red 0%,
  orange 20%,
  yellow 40%,
  green 60%,
  blue 80%,
  purple 100%);
```

Эти два одинаковых градиента создадут радугу, у которой верхний красный цвет постепенно переходит в оранжевый, желтый, зеленый, синий и фиолетовый и каждый переход равномерно распределяется по высоте элемента, что можно увидеть, обратившись к онлайн-ресурсам к главе.

Если хотите, чтобы фон был не уродливой градиентной, а уродливой полосатой радугой (если на большее фантазии просто не хватает), полосатость можно создать за счет жестко заданных цветовых опорных точек, по две из которых находятся в одном и том же месте, обеспечивая вместо градиента резкую смену цвета:

```
background-image: linear-gradient(
  red 16.7%,
  orange 16.7%,
```

```
4     orange 33.3%,
5     yellow 33.3%,
6     yellow 50%,
7     green 50%,
8     green 66.7%,
9     blue 66.7%,
10    blue 83.3%,
11    purple 83.3%);
```

Первые 16,7 % этот градиент будет красным, поскольку окончание первого цвета задано на этой отметке. Следует запомнить: если первая опорная точка установлена не на 0 %, первый цвет будет сплошным вплоть до первой цветовой опорной точки. На 16,7 % первый цвет должен постепенно перейти во второй. А следующая цветовая опорная точка также поставлена на 16,7 %. И переход к оранжевому цвету проходит в течение 0 % — именно этим и обусловливается эффект создания полос. Постепенного перехода от одного цвета к другому нет, а есть жестко заданные цветовые опорные точки. Такие точки хороши для создания полос и таких популярных эффектов, как кнопки-леденцы.

Если нужен слегка непрозрачный (или, для сторонников спора о полупустом или наполовину полном стакане, слегка прозрачный) градиент, следует воспользоваться цветами в формате HSLA или RGBA:

```
linear-gradient(180deg,
  rgba(0,0,0,0.1) 10%,
  rgba(0,0,0,0.9) 90%);
```

Линейные градиенты iPhone и CubeeDoo

Выяснив, что такое градиенты, можно приступить к созданию панели навигации для нашего искусственного iOS-приложения, представляющей собой довольно простой линейный градиент. Первой идет панель заголовка iOS. И сначала мы определим обыкновенную синюю панель — все, за исключением градиента:

```
1   header {
2     /* общий вид */
3     padding: 7px 10px;
4     background-color: #6D84A2;
5     display: block;
6     height: 45px;
7     -moz-box-sizing: border-box;
8     box-sizing: border-box;
9     line-height: 30px;
10    border-bottom: 1px solid #2C3542;
11    border-top: 1px solid #CDD5DF;
```

В строках 3–11 определяется общий вид. Обратите внимание на строку 4, в которой объявлен цвет фона. При использовании градиента или фонового изображения любого типа всегда нужно объявлять фоновый цвет на случай сбоя фонового изображения, например, в устаревших браузерах, не поддерживающих градиенты. А еще фоновый цвет объявляют для того, чтобы вместо создания градиентов раз-

личных цветов для каждого шаблона создаваемой цветовой схемы можно было создать несколько полупрозрачных белых градиентов и использовать их поверх сплошного фонового цвета, получая единый многократно используемый градиент для всех своих цветовых схем.

Поскольку мы включили в код высоту и отступ, то для получения блочной модели IE было сделано объявление `box-sizing: border-box`. Также мы добавили префикс `-moz-`, который все еще нужен, поскольку свойство, несмотря на повсеместную поддержку, до сих пор считается экспериментальным.

Начнем с объявления сплошных цветов. Верхняя половина панели заголовка окрашена в сплошной цвет, а с середины до нижней границы панель становится темнее. Для верхней части мы объявили сплошной цвет и задали начало постепенного перехода на отметке 50 %:

```
15   background-image: -webkit-linear-gradient(top,
16     rgb(176, 188, 205) 50%,
17     rgb(129, 149, 175) 100%);
18   background-image: linear-gradient(to bottom,
19     rgb(176, 188, 205) 50%,
20     rgb(129, 149, 175) 100%);
```

Первая цветовая опорная точка градиента находится на отметке 50 %, поэтому до нее фоновый цвет будет сплошным синим (без постепенного перехода). В нижней половине фон будет постепенно приобретать другой оттенок синего цвета.

Если нужно, чтобы этот градиент работал в BlackBerry 7, Android 3.0 и других устаревших браузерах, следует включить в код исходный префиксный синтаксис:

```
12   background-image: -webkit-gradient(linear, 0 0, 0 100%,
13     color-stop(0.5, rgb(176, 188, 205)),
14     color-stop(1.0, rgb(129, 149, 175)));
```

Для еще более старых WebKit-устройств, предшественников iOS 4.3, Android 3.0 и BB7, используется следующий синтаксис:

```
-webkit-gradient(linear, <начальная точка>, <конечная точка>,
  color-stop(<float>, <цвет>)[, color-stop(<float>, <цвет>),
  color-stop(<float>, <цвет>)])
```

Здесь начальная и конечная точки определяют путь градиента, а каждая цветовая опорная точка включает ключевое слово `color-stop`, за которым следует число с плавающей точкой в диапазоне от 0 до 1 вместо процента от размера изображения. Цветовая опорная точка включает отметку в виде числа с плавающей точкой и цвет, разделенные запятой и заключенные в скобки.

Начальная и конечная точки представляют собой пару разделенных запятой значений для отметок в виде чисел, процентных отношений или ключевых слов `top`, `bottom`, `left` и `right`. Для каждой добавляемой точки нужно использовать два значения.

Цветовая опорная точка (или точки) является необязательным объявлением вида `color-stop`, указывающим цвет и позицию этой точки.

```
background-image:
  -webkit-gradient(linear, left top, left bottom,
    color-stop(0.1, red),
    color-stop(0.3, orange),
    color-stop(0.5, yellow),
    color-stop(0.7, green),
    color-stop(0.9, blue)
  );
```

Данный пример создает радугу, простирающуюся сверху вниз (из левого верхнего в левый нижний угол), которая начинается со сплошного красного цвета, затем постепенно, на 10% пути переходит в оранжевый, затем в желтый, зеленый и синий, становясь сплошным синим цветом с отметки 90% до отметки 100%. Ссылка на разъяснения имеется в онлайн-ресурсах к главе.

Мы получили вполне рабочую версию, но давайте обеспечим для градиента возможность более широкого применения. Вместо создания нового градиента для каждого цвета можно создать полупрозрачный белый градиент и накладывать его сверху на кнопки и панели навигации любого цвета: создайте один градиент и применяйте его повсеместно. Многократно применяемый градиент можно создать использованием формата RGBA (или HSLA), этот прозрачный или слегка непрозрачный градиент позволит при желании быстро менять цветовые схемы:

```
21  background-color: rgb(129, 149, 175);
22  background-image: -webkit-gradient(linear, 00, 0100%,
23          color-stop(0.5, rgb(255, 255, 255, 0.4)),
24          color-stop(1.0, rgb(255, 255, 255, 0)));
25  background-image: -webkit-linear-gradient(
26      rgba(255, 255, 255, 0.4) 50%,
27      rgba(255, 255, 255, 0));
28  background-image: linear-gradient(
29      rgba(255, 255, 255, 0.4) 50%,
30      rgba(255, 255, 255, 0));
31
```

В этом втором из двух вариантов синтаксиса были убраны объявления `top to bottom`, поскольку они используются по умолчанию, и `100%`, так как по умолчанию, если позиция опущена, последний цвет останавливается на отметке 100%.

Как показано в онлайн-ресурсах к главе, за счет использования альфа-прозрачности вместо сплошного цвета можно создать приложение, похожее по внешнему виду на настоящее iPhone-приложение. Прозрачность позволяет быстро и легко менять цветовую схему. Один и тот же градиент для заголовка может использоваться во всех цветовых схемах. Если есть зеленая цветовая схема, можно просто заменить фоновый цвет на темно-зеленый.

А как насчет градиентов для кнопок? У них градиент также покрывает половину высоты кнопки. Можно взять использовавшийся ранее синтаксис градиента, инвертировав его (у панелей сплошной цвет сверху, а у кнопок — снизу):

```
.nav li {
  background-image: -webkit-linear-gradient(bottom,
      rgba(255, 255, 255, 0.4) 50%,
      rgba(255, 255, 255, 0));
  background-image: linear-gradient(to top,
      rgba(255, 255, 255, 0.4) 50%,
      rgba(255, 255, 255, 0));
}
.cancel {background-color: #4A6C9B;}
.done {background-color: #2463DE;}
```

Но это не совсем то, что нам нужно. У кнопок-леденцов при встрече градиента со сплошным цветом вместо плавных используются резкие изменения. Чтобы получить правильный внешний вид, следует задать подъем градиента только до отметки 50 %. Добиться этого можно тремя способами.

- Задать позицию градиента на отметке 50 %, используя свойство `background-position`.
- С помощью свойства `background-size` создать фоновое изображение, занимающее только половину высоты кнопок.
- Создать градиент с жестко заданной цветовой опорной точкой.

Как задать позицию фонового изображения с помощью свойства CSS 1 `background-position`, вы уже знаете (`background-position: 0 15px`), поэтому для большей гибкости давайте сделаем это с помощью свойства CSS3 `background-size` (а заодно изучим это новое свойство).

background-size

Свойство `background-size` указывает размер фоновых изображений. Значение должно быть длиной, заданной в виде абсолютной или относительной величины, или ключевым словом `contain`, `cover` или `auto`.

Значение `contain` при необходимости подгоняет масштаб изображения, сохраняя соотношение его сторон, из-за чего может остаться незанятое пространство. Прямоугольное фоновое изображение в элементе квадратной формы, установленное с применением свойства `background-size: contain`, не заполнит всю область, а будет подгоняться по масштабу до тех пор, пока полностью не поместится внутри своего контейнера. Какая из областей останется незанятой, зависит от значений свойств `background-position` и `background-repeat`.

Значение `cover` масштабирует изображение так, чтобы оно закрывало всю область элемента, но при этом, возможно, часть изображения будет не видна, если блок элемента и изображение не имеют одинакового соотношения сторон. Прямоугольное фоновое изображение в элементе квадратной формы, установленное

с применением свойства background-size: cover, будет обрезано. Значение auto сохранит исходный размер изображения. Эффект, получаемый от применения этих трех ключевых слов, показан на рис. 9.10.

Рис. 9.10. Свойство background-size, имеющее значения auto, cover, contain и числовые значения длины (при установке background-repeat: no-repeat)

Размер фонового изображения можно задать также в виде абсолютных или относительных величин, указав при этом одно или два значения. Если указано одно значение, оно будет принято за ширину, а высота будет масштабироваться вверх или вниз, следовательно, как показано в последнем примере на рис. 9.10, в него не вносятся искажения. Если отдавать предпочтение масштабированию по высоте, укажите для ширины значение auto.

Два значения нужно указывать в следующей последовательности: ширина и высота — и фоновое изображение будет растянуто (или сжато), чтобы соответствовать указанным значениям. На рис. 9.10 видно, что при задании 150 px на 150 px (в блоке 250 × 250) изображение искажается, чтобы поместиться в определенном размере.

Вероятность появления искажений возникает только в случае указания как ширины, так и высоты. При использовании ключевых слов contain, cover, auto, а также объявлении только одного значения изображение всегда будет сохранять исходное соотношение сторон. Если нужно объявить ширину или высоту, то, чтобы избежать искажений, можно объявить только один из параметров, установив для другого значение auto.

Свойство background-size можно использовать в отношении градиентов для кнопок, поскольку нам нужно, чтобы градиент находился в верхней части, а нижняя часть была окрашена в сплошной цвет. Разница при использовании кнопок-леденцов состоит в том, что нам не нужно, чтобы градиент постепенно переходил в сплошной цвет, а нужна жестко заданная цветовая опорная точка. Как отмечалось ранее, существует несколько способов получения такого эффекта. В примере 9.1 используется свойство background-size.

Пример 9.1. Кнопки iOS

```
.button {
  background-image:
    -webkit-gradient(linear, 0 100%, 0 0%,
      from(rgba(255,255,255, 0.1)),
      to(rgba(255,255,255,0.4)));
```

```
6     background-image:
7       -webkit-linear-gradient(bottom,
8         rgba(255, 255, 255, 0.1),
9         rgba(255, 255, 255, 0.4));
10    background-image:
11      linear-gradient(to top,
12        rgba(255, 255, 255, 0.1),
13        rgba(255, 255, 255, 0.4));
14    background-repeat: no-repeat;
15    background-size: 100% 50%;
16  }
17  .cancel {
18    background-color: #4A6C9B;
19      float: left;
20  }
21  .done {
22    background-color: #2463de;
23      float: right;
24  }
```

Мы определили градиент, подобный градиенту заголовка (`header`), но распространяющийся не сверху вниз, а снизу вверх. Затем, воспользовавшись свойством `background-size`, заставили его занять только верхнюю часть кнопки: предписали ему занять только 50 % высоты при 100%-ной ширине.

Нам не нужно, чтобы фоновый градиент был выложен как плитка или повторялся в вертикальном направлении, поэтому мы включили объявление `background-repeat: no-repeat`. Могли бы также сделать следующее объявление:

```
background-size: 2px 50%;
background-repeat: repeat-x;
```

С его помощью мы бы сделали фон шириной лишь 2 пиксела и повторили его по горизонтали. Использование ширины 2 пиксела (или большего значения), а не 1 пиксел обусловлено ошибкой, имеющейся в некоторых версиях Chrome. Все свойства фона объявлены отдельно, без использования краткой формы записи, из-за чего мы не сможем случайно отменить какое-либо из свойств фона, непреднамеренно переключив его на использование исходных значений.

По умолчанию фоновые изображения градиента заполняют весь фон элемента. Но, поскольку мы определяем размер градиента не 100 % на 100 %, как принято по умолчанию, а используем другое значение, следует рассмотреть необходимость повторения фонового изображения. В данном случае мы этого не делаем.

ПРИМЕЧАНИЕ

При создании кода сброса CSS я обычно использую для большинства элементов объявление общего характера background-repeat: no-repeat. Хотя значение, отменяющее повторение, no-repeat, не используется в браузере по умолчанию, чаще всего разработчики останавливают свой выбор именно на нем. Не используя краткую форму задания фона, я не отменяю действие того кода, который применяла при сбросе исходных установок браузера.

DPI и свойство background-size. Свойство `background-size` позволяет изображениям с высоким разрешением работать на устройствах с высоким показателем DPI.

Изображения с высоким разрешением очень хорошо выглядят на устройствах с высоким DPI, но все изображения идут по проводам с одинаковым разрешением. Если нужно показать изображение в высоком разрешении, вы показываете изображение, которое в большинстве случаев вдвое выше и вдвое шире дисплея. Именно здесь в CSS для мобильных устройств и проявляется жизненно важная роль свойства `background-size`.

Предположим, что логотип вашей корпорации имеет высоту 100 пикселов при ширине 300 пикселов. Изображение PNG-формата 100 × 300 пикселов подойдет для многих устройств. Но на устройствах с высоким DPI изображение PNG-формата 200 × 600 пикселов будет выглядеть еще лучше. Разрешение изображений абсолютно одинаковое, просто одно больше другого в четыре раза. Чтобы поместить более крупное изображение в исходное пространство 100 × 300 CSS-пикселов и обеспечить ему отличный внешний вид, присущий изображению с более высоким разрешением, используют свойство `background-size`.

По умолчанию для свойства `background-size` используется значение `auto`, которое заставит изображение появиться в размере 200 × 600 пикселов. Если ваш `div`-контейнер `#logo` имеет размер 100 × 300 пикселов, можно воспользоваться следующим объявлением:

```
#logo {
width: 300px; height: 100px;
background-image: url(logo.png);
background-size: contain; /* ИЛИ */
background-size: 300px 100px;
}
@media
screen and (-webkit-min-device-pixel-ratio: 2),
screen and (min--moz-device-pixel-ratio: 2),
screen and (-min-moz-device-pixel-ratio: 2),
screen and (-o-min-device-pixel-ratio: 2/1),
screen and (min-device-pixel-ratio: 2),
screen and (min-resolution: 192dpi),
screen and (min-resolution: 2dppx) {
  #logo {
    background-image: url(hidpi/logo.png);
  }
}
```

Будут работать оба объявления, `background-size: contain;` и `background-size: 300px 100px;`, поскольку ширина и высота `div`-контейнера соответствует тому размеру, в который нужно поместить изображение.

Но у свойства `background-size` есть один недостаток: изображения с объявленным `background-size` выводятся в два раза медленнее, поскольку нужно время на декодирование изображения и изменение его размеров. При анимации больших изо-

бражений или использовании большого количества перекомпоновок и перерисовок несколько дополнительных миллисекунд могут создать проблему.

Полосатые градиенты

При имитации приложений iOS было охвачено большинство фоновых изображений, но все же их список не является исчерпывающим. Как насчет полосатого фона для iOS-приложений? Неужели для него придется воспользоваться GIF-форматом? Не спешите с выводами. Как отмечалось ранее, полосы можно получить и с помощью CSS-градиентов. Фон iOS-приложения представляет собой градиент, а именно полосатый градиент.

Если в одной и той же позиции объявить две цветовые опорные точки, постепенного перехода одного цвета в другой не произойдет, то есть градиента не будет, а появится четкая линия. Этот прием играет весьма важную роль в создании различных графических форм. Например, чтобы получить полосы, можно составить следующее объявление:

```
background-image:
  -webkit-gradient(linear, 0 0, 100% 0,
    color-stop(0.4, #ffffff),
    color-stop(0.4, #000000),
    color-stop(0.6, #000000),
    color-stop(0.6, #ffffff));
background-image:
  -webkit-linear-gradient(right,
    #ffffff 40%,
    #000000 40%,
    #000000 60%,
    #ffffff 60%);
background-image:
  linear-gradient(to left,
    #ffffff 40%,
    #000000 40%,
    #000000 60%,
    #ffffff 60%);
```

Этот градиент изменяется справа налево, начинается и заканчивается белым цветом и имеет по центру вертикальную черную полосу, начало которой на 40%, а окончание — на 60% (рис. 9.11).

У градиента фонового изображения основной части документа имеются жестко заданные цветовые опорные точки. Для получения нужного эффекта мы создали эти точки так, что один цвет заканчивается, а другой начинается без постепенного перехода. А вот как выглядит код получения повторяющегося фонового изображения шириной 7 пикселов:

```
1   background-color: #C5CCD4;
2   background-image:
3     -webkit-gradient(linear, 0 0, 100% 0,
```

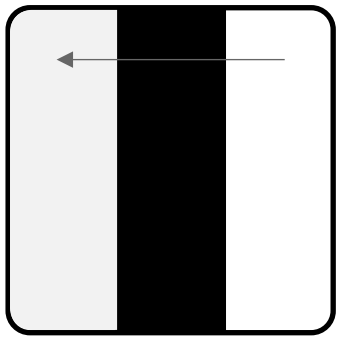


Рис. 9.11. Полоса, созданная с помощью линейного градиента с жестко заданными цветовыми опорными точками

```
4       color-stop(0.7142, #C5CCD4),
5       color-stop(0.7142, #CBD2D8));
6   background-image:
7     -webkit-linear-gradient(left,
8       #C5CCD4 0.7142,
9       #CBD2D8 0.7142%);
10   background-image:
11     linear-gradient(to right,
12       #C5CCD4 0.7142%,
13       #CBD2D8 0.7142%);
14   background-size: 7px 2px; [1]
15   background-repeat: repeat;
```

Поскольку цветовые опорные точки находятся в одном и том же месте, один цвет будет резко переходить в другой, создавая четкую линию. До первой цветовой опорной точки цветом фона будет цвет, указанный для этой точки. После последней цветовой опорной точки цветом фона будет цвет, указанный для данной точки.

Относительно градиента, полученного в строках кода 2–13, следует заметить, что им создается одно фоновое изображение, которое станет по умолчанию покрывать 100 % элемента, к которому будет применено. Что же касается фонового изображения основной части нашего веб-приложения, нужно, чтобы это изображение выкладывалось как плитка, столбцами или тонкими полосками. Для получения такого эффекта нужно задать изображению соответствующий размер и выложить его в виде плитки как по горизонтали, так и по вертикали или как минимум по горизонтали. Мы это сделаем с помощью свойств `background-repeat` и `background-size`.

Поскольку мы объявили ширину фонового изображения 7 пикселов, оно будет состоять из полосы шириной 5 пикселов, окрашенной в цвет #C5CCD4, и полосы

[1] Был установлен размер фона 2 пиксела, а не 1 пиксел, хотя подошла бы любая другая ненулевая высота. Это было сделано из-за ошибки Chrome, которая не позволяет успешно повторять градиенты меньше 2 пикселов.

шириной 2 пиксела, окрашенной в цвет #CBD2D8. Затем это узкое изображение будет повторено как по вертикали, так и по горизонтали. Мы могли бы также создать этот градиент, используя не проценты, а пикселы:

```
background-image:
  -webkit-gradient(linear, 0 0, 100% 0,
    color-stop(5px, #C5CCD4),
    color-stop(5px, #CBD2D8));
background-image:
  -webkit-linear-gradient(left,
    #C5CCD4 5px,
    #CBD2D8 5px);
background-image:
  linear-gradient(to right,
    #C5CCD4 5px,
    #CBD2D8 5px);
```

Или могли бы создать повторяющийся градиент с помощью значения `repeating-linear-gradient()` вместо использования свойств `background-size` и `background-repeat`.

Повторяющиеся линейные градиенты

CSS3 предоставляет свойства `background-size` и `background-repeat`, позволяющие создавать интересные эффекты, включая полосатый фон. Также в CSS3 есть свойство, позволяющее создавать повторяющиеся линейные градиенты, — `repeating-linear-gradient`:

```
background-image:
  -webkit-repeating-linear-gradient(left,
    #C5CCD4 0,
    #C5CCD4 5px,
    #CBD2D8 5px,
    #CBD2D8 7px);
background-image:
  repeating-linear-gradient(to right,
    #C5CCD4 0,
    #C5CCD4 5px,
    #CBD2D8 5px,
    #CBD2D8 7px);
background-size: 7px 7px;
background-repeat: repeat;
```

Относительно повторяющихся градиентов следует отметить несколько особенностей. Шириной градиента будет значение последней цветовой опорной точки — в нашем случае это 7 пикселов. Здесь нужно объявлять нулевое значение и последнее значение. В отличие от обычных линейных градиентов, эти значения по умолчанию для вас не объявляются.

К тому же не все браузеры поддерживают исходное свойство задания размеров повторяющихся градиентов[1]. Пока все недостатки не будут устранены, при использовании повторяющихся линейных градиентов проверяйте, не нужно ли включить в код свойство `background-size` и повторять градиент с помощью свойства `background-repeat`, как это сделано в строках 14 и 15.

Повторяющиеся линейные градиенты получают такую же поддержку со стороны браузеров, как и беспрефиксные обычные линейные градиенты. В настоящий момент они поддерживаются всеми основными браузерами, начиная с BlackBerry 10, iOS 5, Safari 5.1 и Chrome 10, хотя префиксы производителей по-прежнему актуальны для некоторых WebKit-браузеров. Как уже отмечалось, они некорректно выводятся в Chrome или в Chrome для Android. Хотя мне довольно часто нравится эффект, который получается в результате неверной обработки браузером Chrome повторяющихся линейных градиентов.

Градиенты в CubeeDoo

Хотя Chrome для Android не в состоянии правильно вывести повторяющиеся линейные градиенты, я их все же включила в создание фона приложения CubeeDoo. Мне действительно нравится эффект, который получается из-за странностей вывода браузером Chrome этих градиентов, а на небольшое отличие фона в других браузерах, которые правильно поддерживают эти градиенты, просто не стоит обращать внимания.

На небольших по размеру устройствах игровое поле CubeeDoo занимает все окно просмотра. В более крупных устройствах игра занимает только часть экрана, поэтому в качестве фона страницы на них используется такой же повторяющийся линейный градиент, как и на игровом поле:

```
body {
  background-color: #eee;
  background-image:
    -webkit-repeating-linear-gradient(-135deg,
      transparent 0,
      transparent 4px,
      white 4px,
      white 8px),
    -webkit-repeating-linear-gradient(135deg,
      transparent 0,
      transparent 4px,
      white 4px,
      white 8px);
  background-image:
    repeating-linear-gradient(-135deg,
      transparent 0,
```

[1] Например, в Chrome имеется весьма странная ошибка. Похоже, этот браузер считает, что размер фона должен быть 110 пикселов, и повторяет все, что меньше по размеру, каким-то странным образом, временами давая сбой, когда ширина повторяющегося линейного градиента — более 100 пикселов.

```
      transparent 4px,
      white 4px,
      white 8px),
    repeating-linear-gradient(135deg,
      transparent 0,
      transparent 4px,
      white 4px,
      white 8px);
}
```

Обратите внимание на то, что фактически в качестве двух фоновых изображений мы включили два градиента, создав на игровом поле эффект ромба.

Использование нескольких фоновых изображений

А как включить два градиента? CSS3 позволяет использовать для отдельного DOM-узла несколько фоновых изображений. Для этого нужно просто разделить разные фоновые изображения запятыми, при этом тип этих изображений роли не играет. В предыдущем примере можно заметить, что в одном объявлении `background-image` имеются два повторяющихся линейных градиента, отделенные друг от друга запятой.

Два градиента включены также в фоновое изображение игрового поля, они создают эффект шахматной доски. Градиент фона игрового поля настолько мал, что на устройствах с низким показателем DPI он полностью не выводится, но все же создает текстуру для всех устройств:

```
#board {
  color: #fff;
  height: 400px;
  width: 100%;
  float: left;
  text-align: center;
  background-color: #eee;
  background-image:
    -webkit-gradient(linear, 0 0, 100% 100%,
      color-stop(0.5, rgba(255,255,255,0)),
      color-stop(0.5, rgba(255,255,255,0.5))),
    -webkit-gradient(linear, 0 100%, 100% 0,
      color-stop(0.5, rgba(255,255,255,0)),
      color-stop(0.5, rgba(255,255,255,0.5)));
  background-image:
    -webkit-linear-gradient(-135deg,
      rgba(255,255,255,0) 50%,
      rgba(255,255,255,0.5) 50%),
    -webkit-linear-gradient(135deg,
      rgba(255,255,255,0) 50%,
      rgba(255,255,255,0.5) 50%);
  background-image:
    linear-gradient(-135deg,
      rgba(255,255,255,0) 50%,
```

```
      rgba(255,255,255,0.5) 50%),
    linear-gradient(135deg,
      rgba(255,255,255,0) 50%,
      rgba(255,255,255,0.5) 50%);
  background-size: 2px;
}
```

Как вы заметили, мы опять объявили в одном узле несколько фоновых изображений и только одно значение background-size. Когда для размера фона задается только одно значение, оно используется как для высоты, так и для ширины. А когда только одно значение используется для нескольких фоновых изображений, все они будут одинакового размера. Если бы фоновые изображения имели разные размеры, то несколько размеров фона нужно было бы указать через запятые и объявления background-size перечислять в том же порядке, в котором следовали объявления background-image.

Я не рекомендую использовать краткую форму записи свойства background, поскольку при этом для всех свойств фона устанавливаются их значения по умолчанию, даже если вы этого не хотели. Если станете использовать краткую форму записи с несколькими фоновыми изображениями, то в последнем объявлении фона нужно объявить только background-color.

Кнопки-леденцы и жестко заданные цветовые опорные точки

Научившись создавать в градиентах жестко заданные цветовые опорные точки, вернемся к кнопкам-леденцам. Для создания в градиенте постепенного перехода от четко обозначенного края было использовано свойство background-size, но можно воспользоваться и градиентом с жестко заданной цветовой опорной точкой:

```
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
      color-stop(.5, rgba(255, 255, 255, 0)),
      color-stop(.5, rgba(255, 255, 255, 0.1)),
      color-stop(1, rgba(255, 255, 255, 0.4)));
background-image: -webkit-linear-gradient(bottom,
      rgba(255, 255, 255, 0) 50%,
      rgba(255, 255, 255, 0.1) 50%,
      rgba(255, 255, 255, 0.4));
background-image: linear-gradient(to top,
      rgba(255, 255, 255, 0) 50%,
      rgba(255, 255, 255, 0.1) 50%,
      rgba(255, 255, 255, 0.4));
```

Как видите, добиться желаемого результата можно разными путями, но в данном случае удобнее всего воспользоваться градиентом.

Инструментарий создания градиентов

Конечно, разобраться в создании линейных градиентов не помешает (именно поэтому мы и рассматривали подробности их создания), но проще все же воспользоваться для этого каким-нибудь инструментом. В онлайн-ресурсах к главе имеются

ссылки на информацию, инструменты и библиотеки градиентов, которые могут помочь вам приступить к работе.

Вы, наверное, заметили, что мы включили в код фоновый цвет. Всегда задавайте фоновый цвет, потому что ваш сайт будут просматривать и в тех браузерах, которые не поддерживают градиенты, вы можете допустить опечатку в коде градиента или может произойти сбой при повторении заполняющего фон рисунка. Включая в код установку фонового цвета, можно в случае сбоя при выводе фонового изображения или градиента предотвратить потерю контраста между текстом и фоном.

ПРИМЕЧАНИЕ

В WebKit при использовании градиентов для маркеров элементов списка указать размер изображения невозможно. Браузеры на основе WebKit для изображения градиента будут выбирать размер по умолчанию относительно размера шрифта элемента списка.

Тени

Освоив работу с линейными градиентами, свойством `background-size` и скругленными углами, вернемся к нашим кнопкам и придадим им больше глубины. Мы уже ознакомились с большинством самых новых свойств CSS, за исключением теней и, возможно, менее известного свойства `text-overflow`.

Код кнопок имеет следующий вид:

```
.button {
  background-color: #4A6C9B;
  background-image:
    -webkit-gradient(linear, 0 100%, 0 0%,
      from(rgba(255,255,255,0.1)),
      to(rgba(255,255,255,0.4)));
  background-image:
    -webkit-linear-gradient(bottom,
      rgba(255, 255, 255, 0.1),
       rgba(255, 255, 255, 0.4));
  background-image:
    linear-gradient(to top,
      rgba(255, 255, 255, 0.1),
      rgba(255, 255, 255, 0.4));
  background-size: 100% 50%;
  background-repeat: no-repeat;
  color: #FFFFFF;
  border: 1px solid #2F353E;
  border-color: #2F353E #375073 #375073; /* T LR B */
  border-radius: 4px;
  text-decoration: none;
  font-family: Helvetica;
  font-size: 12px;
  font-weight: bold;
  height: 30px;
  padding: 0 10px;
```

```
27      text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.6);
28      overflow: hidden;
29      max-width: 80px;
30      white-space: nowrap;
31      text-overflow: ellipsis;
32      -webkit-box-shadow:
33        0 1px 0 rgba(255,255,255, 0.4);
34      -webkit-box-shadow:
35        0 1px 0 rgba(255,255,255, 0.4),
36        inset 0 1px 0 rgba(255,255,255,0.4);
37      box-shadow:
38        0 1px 0 rgba(255,255,255, 0.4),
39        inset 0 1px 0 rgba(255,255,255,0.4);
40    }
```

Солидный объем! Новые познания в области градиентов позволяют нам понять основную часть кода (до строки 26 включительно), но в нем могут быть и незнакомые свойства, например `text-shadow`, `box-shadow` и `text-overflow`. Но не стоит переживать! Рассмотрим и эти свойства.

Кнопка с переходом от синего к темно-синему цвету была создана с помощью фонового цвета, который виден сквозь полупрозрачный белый градиент. В строках 18–20 создается граница со скруглением угла радиусом 4 пиксела, состоящая из разных оттенков синего цвета. В строке 15 определяется размер градиента: верхняя часть кнопки является градиентом, а ее нижняя часть окрашена в сплошной цвет фона, просвечивающийся через градиент. Изображение градиента было определено для всей ширины, но только для половины высоты элемента.

Поскольку изображение градиента не занимает 100% площади элемента, то по умолчанию оно будет повторено по вертикали. Нам это не нужно, поэтому в строке 16 делается объявление `background-repeat: no repeat` — и повторение изображения отменяется. Если бы для изображения была определена ширина 2 пиксела (`background-size: 2px 50%;`), нам пришлось бы повторять это изображение по горизонтали, воспользовавшись объявлением `background-repeat: repeat-x;`.

Остальной код до строки 26 включительно задает цвет текста, гарнитуру, жирность и размер шрифта, а также удаляет подчеркивание ссылок. Хотя в коде HTML-примеры ссылки отсутствуют, они зачастую оформляются в виде кнопок и для них по умолчанию используется шрифт с подчеркиванием. Если бы вместо `<li>`-элементов я использовал ссылки, то по умолчанию они отображались бы как встроенные линейные элементы, и тогда высота не имела бы никакого значения. Но поскольку наши кнопки выполнены как плавающие элементы в `nav`-панели, то независимо от того, какой у элемента класс, `cancel` или `done`, он будет отображен в виде блока.

Для нас свойство `text-shadow` в строке 27 является новым, но в спецификации CSS оно известно довольно давно! Давайте его рассмотрим.

Текстовые тени

Текстовые тени были добавлены к спецификации CSS 2 и удалены из CSS 2.1. И вот они вернулись! Но уже в обратном порядке.

Компании Apple нравится подавать свой текст рельефно. На сайте Apple.com они создали рельефный текст, используя для него изображения, поскольку хотели обеспечить одинаковый внешний вид своего сайта во всех браузерах. Фактически для варианта навигации у них использовалось изображение спрайта, под которым было еще одно изображение для серой панели навигации. В целом при первоначальной загрузке страницы они всех нас заставляли загружать свыше 3,5 Мбайт изображений, потому что хотели гарантировать всем посетителям, даже тем, кто пользовался IE6 (хотя, если люди не желают раскошеливаться на обновление своего компьютера 10-летней давности, я не думаю, что они станут переходить на Mac), единый внешний вид сайта.

В мобильном пространстве нам не приходится волноваться о давно устаревших браузерах настольных систем наподобие старых IE, не поддерживающих текстовые тени. Не нужно включать спрайты в свои навигационные панели, и вместо изображений мы можем воспользоваться свойством `text-shadow`, сэкономив на количестве HTTP-запросов и упростив локализацию.

Свойство `text-shadow` определяет тень текста, добавляемую к текстовому содержимому элемента. Синтаксисом значения свойства `text-shadow` является список значений левого смещения (`leftOffset`), верхнего смещения (`topOffset`), степени размытости (`blurRadius`) и цвета тени (`color`), в котором в качестве разделителя используется запятая. Для тени требуется указать два смещения, а вот степень размытости и цвет указывать не обязательно:

```
text-shadow: <leftOffset> <topOffset> <blurRadius> <color>[,
  <leftOffset> <topOffset> <blurRadius> <color>][,
  <leftOffset> <topOffset> <blurRadius> <color>];
```

Для смещения используются две линейные величины: горизонтальное расстояние до правой стороны текста при положительном значении или до левой стороны текста — при отрицательном и вертикальное расстояние ниже текста при положительном значении и выше его — при отрицательном.

Третье значение определяет ширину размытости тени. Чем больше это значение, тем больше размытость, делающая тень текста больше и более прозрачной. Отрицательные значения запрещены. Если этот параметр не указан, то по умолчанию радиус размытости устанавливается в 0, создавая четкий край тени.

Смещения указывать обязательно, но им можно присваивать значение `0`, при этом тень будет выглядеть как равномерное свечение со всех сторон. Это свойство может применяться вместе со свойством `currentColor` (см. главу 8).

Некоторые шрифты, например Helvetica Neue Light, при просмотре текста на устройствах с высоким показателем DPI кажутся слишком тонкими и неудобными для чтения. Но с помощью объявления:

```
text-shadow: 0 0 1px currentColor;
```

текст можно сделать более четким. В одной строке CSS задано свечение в 1 пиксел сразу же за текстом, выполненное текущим цветом шрифта. Используя свойство `currentColor`, можно создать единое объявление свойства и значения для всего текста и не переживать за то, что у разных элементов цвет текста будет разным.

При использовании нескольких теней их эффекты применяются в порядке, обратном их объявлению: последняя тень рисуется первой и каждая предыдущая тень помещается поверх нее.

Текстовые тени могут создаваться весьма широкими. Но при этом текстовая тень не оказывает никакого влияния на блочную модель. Текстовая тень может выходить за пределы своего родительского элемента, не увеличивая размеры блока этого элемента. Поэтому, если задать очень широкое отрицательное горизонтальное смещение, можно непреднамеренно закрыть предыдущий текст (пример такого эффекта имеется в онлайн-ресурсах к главе).

Получить красивые текстовые тени позволяют использование значения степени размытости и задание для цвета тени слегка прозрачной версии: обычно для получения легкой серой тени используется объявление `rgba(0,0,0,0.4);`. Кажущийся серым прозрачный черный цвет позволяет видеть сквозь тень фоновый цвет или изображение.

Но я не дизайнер, и тем более не дизайнер компании Apple, которой нравится использовать темно-серую верхнюю тень в 1 пиксел. Чтобы сделать текст рельефнее, добавим такую тень к заголовку страницы (к содержимому элемента `<h1>`) и к кнопкам на странице с перечнем языков:

```
text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.6);
```

Этот код предназначен для создания слегка прозрачной текстовой тени в 1 пиксел, которую отбрасывают верхние части букв.

ПРИМЕЧАНИЕ

Обратите внимание на то, что при указании свойства text-shadow префикс не используется. Кроме того, следует заметить, что большие полупрозрачные тени в сочетании с другими свойствами CSS могут требовать много времени на прорисовку и задействовать большой объем памяти. Учитывайте это при работе.

Подгонка текста с помощью свойств width, overflow и text-overflow

В показанном ранее примере в строках кода с 28-й по 31-ю гарантируется, что кнопки не занимают слишком много места. На мобильных устройствах в нашем распоряжении весьма ограниченное пространство. В примере с перечнем языков мы постарались уместить в верхней области навигации три элемента: две кнопки и заголовок страницы. Поэтому нужно гарантировать не только то, что кнопки не займут слишком много места, но и то, что содержимое элемента `<h1>` не будет слишком широким. Слово Languages в нем помещается, а что делать, если заголовок будет содержать слова Supported Languages?

Вообще-то объявление ширины для линейного элемента наподобие `<button>` не возымеет никакого эффекта. Но поскольку мы сделали наши кнопки плавающими, видимые плавающие элементы по умолчанию будут отображаться как блоки (а не как элементы `list-item`, `inline`, `inline-block` и т. д.), стало быть, свойство `width` работает!

Свойство text-overflow

Свойство `text-overflow` является частью спецификации базового пользовательского интерфейса — Basic UI. Это свойство позволяет обрезать текст, который слишком широк для своего контейнера. Значение `ellipsis` вызывает появление многоточия в том месте, где текст обрезан:

```
text-overflow: ellipsis | clip
```

Но для того, чтобы обрезать текст и включить в него многоточие, одного применения объявления `text-overflow: ellipsis;` недостаточно. Если текст переносится или ему разрешено распространяться за пределы границ содержащего его блока, слова не обрежутся и многоточия также не будет[1]. Многоточие появится, только если для свойства «переполнение» — `overflow`, заданного для содержащего текст блока, будет установлено значение, отличающееся от `visible` (позволяет тексту выходить за пределы блока), а для свойства `white-space` — значение `nowrap` или `pre`. Можно также использовать значение `clip`, которое приведет к обрезанию текста без применения многоточия.

При использовании ключевого слова `ellipses` текст будет обрезан сразу же после последней буквы, полностью помещающейся в родительский блок. В Firefox, начиная с версии 9, даже было добавлено экспериментальное значение `string`.

Хотя в действительности вам вряд ли придется обрезать текст, зачастую этого требует ограниченное пространство панели навигации мобильного приложения. Вообще-то с точки зрения пользовательского восприятия скрытие текста таким (или каким-нибудь другим) образом нельзя считать удачным дизайнерским ходом. Если вы собираетесь обрезать текст, то для улучшения пользовательского восприятия нужно использовать многоточие, информирующее пользователя о том, что текст был обрезан:

```
h1 {
  white-space: nowrap;
  width: 180px;
  overflow: hidden;
  -o-text-overflow: ellipsis; /* opera mini, mobile */
  text-overflow: ellipsis;
}
```

Чтобы свойство заработало, должны быть объявлены свойства `white-space`, `overflow` и, как правило, свойство `width`, следовательно, это свойство в отношении элементов, отображаемых во встроенном режиме, не работает.

Свойство white-space

Свойство `white-space` определяет порядок обработки пробельных символов внутри элемента. Значения для свойства `white-space` включают `normal`, `nowrap`, `pre`, `pre-line`,

[1] Чтобы добавить многоточие к многострочному тексту, существуют специфические для тех или иных производителей значения, включающие свойство -webkit-line-clamp, а также значение text-overflow: -o-ellipsis-lastline; для WebKit и Opera соответственно.

pre-wrap и inherit. Когда для свойства устанавливается значение normal, текст в конце строки будет переноситься, а пробельные символы будут сокращены до одного пробела. Установка значения nowrap заставит текст помещаться в родительском контейнере без разбиения строк. Значение pre-wrap зачастую делает текст более читаемым, чем при использовании значения nowrap: наряду с сохранением исходных пробельных символов текст при достижении конца содержащего его блока переносится на новую строку.

Чтобы заработало объявление text-overflow: elipses;, текст не может переноситься. Именно поэтому сюда и включено это свойство, принадлежащее спецификации CSS 1.

Блочная тень

Блочные тени чаще всего используются для создания отбрасываемых теней, придающих элементам эффект парения над экраном. В CubeeDoo свойство блочной тени box-shadow используется для придания картам эффекта небольшого парения над игровым полем. В наших действиях по обеспечению средствами CSS3 внешнего вида, присущего приложениям iPhone, свойство box-shadow используется для создания эффекта двух тонких теней для кнопок.

Свойство box-shadow дает возможность разработчику добавлять к элементу отбрасываемые или внутренние тени или оба вида теней одновременно. К блоку может быть применено одно или несколько свойств box-shadow, с помощью которых можно создать ряд привлекательных эффектов. Тени рисуются непосредственно за пределами границы или сразу же внутри границы, если используется значение inset.

В понятиях блочной модели тени похожи на свойства внешней границы outline и текстовой тени text-shadow тем, что они не влияют на разметку — могут накладываться на другие блочные элементы. Тени поддерживаются во всех браузерах, начиная с версии IE9. Для Android 3.0 и iOS 5 нужно включать исходный синтаксис без указания префикса производителя и версию с префиксом производителя -webkit-.

Синтаксис и построение изображения свойства box-shadow аналогичны используемым для свойства text-shadow, но с двумя дополнительными признаками распространения размытости и направления внутрь блока — inset:

```
box-shadow: inset <leftOffset> <topOffset> <blurRadius> <spread> <color>[,
  inset <leftOffset> <topOffset> <blurRadius> <spread> <color>][,
  inset <leftOffset> <topOffset> <blurRadius> <spread> <color>];
```

Для свойства box-shadow используются следующие значения: none для отказа от тени (значение по умолчанию) или список значений, разделенных запятыми, для смещения влево (leftOffset), смещения вверх (topOffset), радиуса размытости (blurRadius), радиуса распространения (spread) и цвета (color) с необязательным ключевым словом inset для внутренних теней (как противоположности отбрасываемым теням). Если в объявление включается более одной блочной тени, каждую тень нужно отделять от другой тени запятой.

Значения горизонтального и вертикального смещения являются обязательными. Так же как и для свойства text-shadow, первые три значения в указанном порядке определяют горизонтальную позицию, вертикальную позицию и радиус размытости. Четвертое значение — радиус распространения — делает тень больше при положительном и меньше — при отрицательном значении. Ключевое слово inset, если оно включено в объявление, превращает тень из отбрасываемой во внутреннюю. Ключевое слово inset, значения радиусов размытости и распространения, а также цвета являются необязательными.

Следует заметить, что inset служит вариантом настройки только свойства box-shadow и не является частью свойства text-shadow. Также следует учесть, что Android, вплоть до версии 3.0, и iOS 3.2 требуют использования префикса производителя -webkit- и не поддерживают ключевое слово inset. iOS 4.x и BlackBerry 7 также требуют использования префикса производителя -webkit-, но внутренние тени, задаваемые ключевым словом inset, поддерживают.

Например (этого кода нет в онлайн-ресурсах к главе)[1]:

```
1   -webkit-box-shadow:
2     3px 4px 5px 6px rgba(0,0,0, 0.4);
3   -webkit-box-shadow:
4     3px 4px 5px 6px rgba(0,0,0, 0.4),
5     inset 1px 1px 1px #FFFFFF;
6   box-shadow:
7     3px 4px 5px 6px rgba(0,0,0, 0.4),
8     inset 1px 1px 1px #FFFFFF;
```

В каждой группе имеются два объявления box-shadow, разделенные запятой. Значения, задающие каждую из двух блочных теней и представляющие собой список с пробелами в качестве разделителей, включают:

- левое, или горизонтальное, смещение тени, являющееся первым значением длины. Отрицательное значение приведет к помещению тени слева;
- верхнее, или вертикальное, смещение, являющееся вторым значением длины. Отрицательное значение приведет к помещению тени над блоком;
- необязательное значение радиуса размытости, являющееся третьим значением длины. Отрицательное значение не допускается. Если значение равно нулю, тень получится сплошной. Чем больше значение, тем более размытой будет тень;
- необязательное значение распространения размытости, которое (при использовании) станет четвертым значением длины. Положительное значение приводит к росту тени. По умолчанию тень имеет такой же размер, как и элемент;

---

[1] Строки 1 и 2 предназначены для Android вплоть до версии 3.0 и для iOS 3.2, которые не поддерживают ключевое слово inset. Строки 3–5 предназначены для устаревших браузеров на основе WebKit, в том числе для браузеров вплоть до Safari 5, iOS 4.3, Android 3.0 и BB 7. Строки 6–8 предназначены для Safari 5.1, Opera 10.5+, Opera Mobile 11+, Chrome 9+, IE9+ и Firefox 4+. В Opera Mini эти объявления не поддерживаются.

- необязательное значение цвета. Если радиус размытости больше нуля, рекомендуется использовать альфа-прозрачность. Если начать с немного прозрачной версии черного цвета, эффект будет лучше, чем при начале с непрозрачной версии серого цвета. Но полупрозрачные тени задействуют больше памяти, и на их прорисовку может уйти больше времени, что может вызвать проблемы на устройствах с ограниченным объемом оперативной памяти или при анимации и/или частой перерисовке элемента. Если цвет не определен, то по умолчанию браузеры будут использовать значение свойства `currentColor`;
- необязательное ключевое слово `inset`, которое вызовет создание не отбрасываемой, а внутренней тени.

В нашем примере первая тень шириной 5 пикселов будет окрашена в серый прозрачный цвет и расположится снизу и справа от элемента. Значение второй блочной тени в каждой группе включает необязательное ключевое слово `inset`. Эта тень будет полностью непрозрачным белым внешним контуром шириной 1 пиксел (или все же внутренним?) внутри блока. Тени, если таковые имеются, появляются с внешней и внутренней стороны границы соответственно.

Значения смещений по осям *X* и *Y*, а также необязательные значения радиуса размытости и распространения размытости должны быть записаны именно в таком порядке. Порядок следования в объявлении остальных двух значений (ключевого слова `inset` и цвета) неважен, но чтобы код было проще читать и обслуживать, лучше придерживаться единого порядка. В случае включения нескольких теней порядок следования их значений играет весьма важную роль. Так же как и в случае применения свойства `text-shadow`, тени, задаваемые свойством `box-shadow`, рисуются в порядке, обратном очередности их объявления. Отбрасываемые тени всегда рисуются за блоком, как будто у них более низкий показатель z-index, и не показываются сквозь альфа-прозрачные блоки (в отличие от альфа-прозрачного текста, сквозь который текстовая тень будет видна), что можно увидеть в онлайн-ресурсах к главе.

Тонкая граница с внутренней тенью может использоваться для получения интересного эффекта, но анимация большой полупрозрачной тени может привести к отказу браузера, поскольку все тени, изображения и градиенты вычисляются и рисуются на экране от нижних слоев к верхним, даже если они никогда не становятся видимыми пользователю.

ВНИМАНИЕ

На мобильных устройствах все объявленные свойства заставляют создавать на странице рисунки, даже если они в итоге не видны. Следует учесть, что при добавлении 20 теней и градиентов, даже если они перекрывают друг друга, выходят за пределы страницы, находятся под другим элементом или становятся невидимыми по какой-то другой причине, они все равно рисуются. Перерисовка 20 уровней и вычисление добавленного эффекта прозрачности на каждом пикселе может занять слишком много времени, чтобы получить анимацию с частотой 60 кадров/с.

Хотя поддержка теней появилась еще на первых iPhone, тени могут занимать очень много памяти и подвешивать браузер, особенно если будут полупрозрачными, а не сплошными, и все это усугубляется, если элемент с тенью анимируется.

Браузер при каждой перерисовке должен пересчитывать каждый пиксел, даже если тень становится невидимой.

ВНИМАНИЕ

Большие тени, особенно внутренние, используют большой объем памяти и могут подвесить мобильный браузер.

В строках 37–39 нашей CSS-таблицы для создания кнопок-леденцов включена двойная тень:

```
box-shadow:
  01px 0 rgba(255,255,255, 0.4),
  Inset 01px 0 rgba(255,255,255,0.4)
```

Это объявление приведет к добавлению двух полупрозрачных белых теней: тени толщиной 1 пиксел с внешней стороны нижней границы блока и тени шириной 1 пиксел внутри кнопки в верхней части блока. То же самое мы объявили также с префиксом `-webkit-` с двумя тенями, а также с удалением внутренней тени для очень старых браузеров Android.

А теперь все вместе: CubeeDoo

В CubeeDoo имеется несколько теней. Рассмотрим некоторые выдержки из CSS. Это не весь код, имеющийся в файлах CSS. Здесь представлены только те селекторы, свойства и значения, которые могут быть осмыслены вами в данный момент:

```
1   #board > div {
2     box-shadow: 1px 1px 1px rgba(0,0,0,0.25);
3   }
4   .back {
5     border: 5px solid white;
6   }
7   .back:after,
8   .face:after {
9     position: absolute;
10     content: "";
11     top: 0;
12     left: 0;
13     right: 0;
14     bottom: 0;
15     border-radius: 3px;
16     pointer-events: none;
17   }
18   .back:after {
19     background-repeat: no-repeat;
20     background-color: #fff;
21     background-size:
22       50% 50%,
23       00;
24     background-image:
```

```
25       url('data:image/svg+xml;utf8,<svg width="40" height="40"
        xmlns="http://www.w3.org/2000/svg"><g><text
        xml:space="preserve"
        text-anchor="middle" font-family="serif" font-size="40"
        id="svg_1"
        y="30" x="20" stroke-width="0" stroke="rgb(119, 160, 215)"
        fill="rgb(119, 160, 215)">❁</text></g></svg>'),
26       -webkit-linear-gradient(-15deg,
        rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.05));
27     background-image:
28       url('data:image/svg+xml;utf8,<svg width="40" height="40"
        xmlns="http://www.w3.org/2000/svg"><g><text
      xml:space="preserve"
        text-anchor="middle" font-family="serif" font-size="40"
      id="svg_1"
        y="30" x="20" stroke-width="0" stroke="rgb(119, 160, 215)"
        fill="rgb(119, 160, 215)">❁</text></g></svg>'),
29       linear-gradient(75deg,
        rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.05));
30     box-shadow:
31       inset 1px 1px 0 currentcolor,
32       inset -1px -1px 0 currentcolor,
33       1px 1px 1px rgba(0,0,0,0.1);
34       color: rgb(119, 160, 215);
35   }
36   #board > div.flipped {
37     box-shadow: -1px 1px 1px rgba(0,0,0,0.25);
38   }
39   .control {
40     border: 1px solid rgba(0, 0, 0, 0.25);
41     box-shadow: inset 1px 1px 0 rgba(255, 255, 255, 0.5);
42     background-image:
43       -webkit-linear-gradient(-15deg,
44         rgba(0, 0, 0, 0),
45         rgba(0, 0, 0, 0.025));
46     background-image:
47       linear-gradient(75deg,
48         rgba(0, 0, 0, 0),
49         rgba(0, 0, 0, 0.025));
50     text-shadow: 1px 1px 0px rgba(255, 255, 255, 0.5);
51     border-radius: 5px;
52   }
```

В этом фрагменте кода содержится много примечательных моментов. Давайте проведем построчный разбор.

1. В строке 1 мы нацелились на `<div>`-элементы, являющиеся непосредственными дочерними элементами `#board`, для чего воспользовались комбинатором дочерних элементов. Элементом, на который произведено нацеливание, является контейнер карты. К картам в исходном состоянии в строке 2 добавлена легкая отбрасываемая тень.

2. Путем создания псевдоэлементов в строках с 7-й по 17-ю мы определили внешний вид сгенерированного содержимого. В строке 10 включен требуемый атрибут `content`, который позиционирован на 0 пикселов от верха, левой стороны, низа и правой стороны, тем самым принудительно настраивая абсолютно позиционированное[1] сгенерированное содержимое на точно такие же размер и позицию, что и у родительского узла. Мы также включили объявление свойства `border-radius` со значением `3 px`, чтобы скруглить углы по образцу родительских узлов.

ПРИМЕЧАНИЕ

Объявление для сгенерированного содержимого абсолютной позиции position: absolute при смещении, со всех четырех сторон равном нулю, обеспечит для сгенерированного содержимого те же размеры, что и у родительского элемента.

3. В строках с 17-й по 35-ю задается стиль тыльной стороны карты. Сначала объявляются свойства фона, включая `background-repeat` и `background-color`. У нас имеются два фоновых изображения различных размеров, поэтому в качестве значения свойства `background-size` объявляются два значения, разделенные запятой.
4. У тыльной стороны карты имеются синее обрамление, небольшой цветок и градиент для придания привлекательности. Эффект обрамления создан с помощью свойства блочной тени `box-shadow`. Цветок и придающий привлекательность градиент выполнены с помощью фоновых изображений.

Два фоновых изображения были объявлены двумя разными способами. Первое фоновое изображение представлено в виде файла формата SVG, который рассматривался в главе 5. Второе изображение является градиентом. Они разделены запятой. Эта комбинация изображений объявлена дважды (или же должна быть объявлена трижды, если нужна поддержка старых WebKit-браузеров), потому что синтаксис градиентов изменился, а SVG в более старых версиях Android не поддерживается. Может создаться впечатление, что символ ❁ можно было бы просто поместить в генерируемое содержимое, но, поскольку в следующих главах мы будем переворачивать карту, а поддержка анимации сгенерированного содержимого еще не получила повсеместного распространения, создаем внешний вид сгенерированного содержимого с помощью фонового изображения в виде файла формата SVG.

5. Для создания синей кромки (рис. 9.12) используются две внутренние тени: по верхней и левой и по нижней и правой сторонам. Обратите внимание на то, что синее обрамление повторяет изгибы скругленных углов. Оно отстоит от края карты на 5 пикселов. Возможность создания эффекта обрамления возникла благодаря тому, что в строке 5 была добавлена граница толщиной 5 пикселов. А мы знаем, что тени и внутренние тени создаются с внешней и внутренней сторон границы соответственно.

[1] Родительский элемент настроен на относительное позиционирование, поэтому сгенерированное содержимое будет позиционироваться относительно этого элемента.

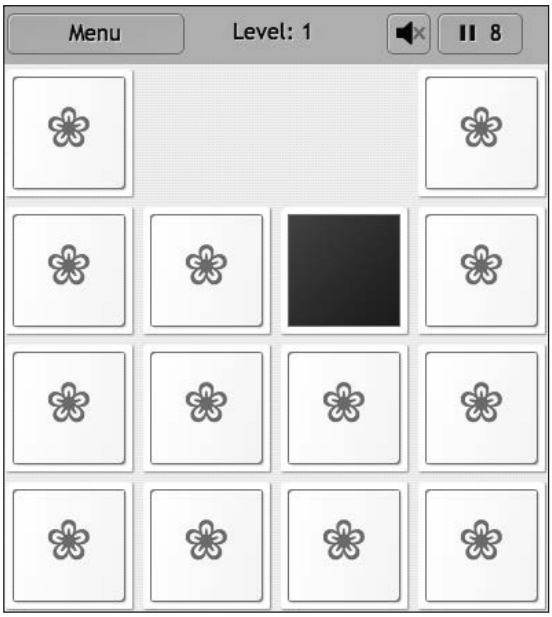


Рис. 9.12. CubeeDoo: синее обрамление создано с помощью толстой границы и двух внутренних теней

6. Последняя из трех теней, та, что объявлена в строке 33, создает эффект отбрасываемой картами тени.
7. В строке 34 объявляется цвет. Казалось бы, что это ни к чему, поскольку в элементе нет никакого текста. Даже «текст» в виде цветка (❀) выполнен с помощью изображения в виде файла формата SVG, а не с помощью сгенерированного содержимого. Но включая в этой строке объявление цвета, мы получаем возможность заставить соответствовать этому объявлению значение `currentColor` в свойстве `drop-shadow`.
8. Когда пользователь переворачивает карту, тень возле нее остается. Но вам, похоже, такой эффект не нужен. Источник света не изменился, следовательно, тень должна быть в том же месте относительно игрового поля, а не карты, когда она переворачивается. В строке 36 используется комбинатор дочернего элемента, но на самом деле он не нужен, поскольку нет никакого другого `div`-контейнера, который бы имел класс `flipped` (перевернутый). Когда `div`-контейнер находится в перевернутом состоянии, мы меняем левое смещение карты, так что визуально она всегда находится на одном и том же месте. Преобразование карт (переворачивание или вращение) будет рассмотрено в главе 11.
9. В строках с 39-й по 52-ю осуществляется настройка элементов управления, в качестве которых фигурируют кнопки CubeeDoo. Для создания эффекта двойной границы (черной и белой) для верхней и левой сторон мы в строках 40 и 41 включили полупрозрачную черную границу и добавили белую внутреннюю тень.

10. В качестве фона кнопок добавили полупрозрачный градиент, чтобы придать кнопкам объем. Обратите внимание на то, что для префиксного и беспрефиксного синтаксиса углы записаны по-разному, но задаваемый ими внешний вид будет одинаковым.
11. В строках 50 и 51 мы добавили кнопкам полупрозрачную белую текстовую тень, а также скругленные углы.

Мы рассмотрели все свойства, задействованные при создании внешнего вида, присущего приложению iPhone, без использования изображений, и все свойства исходного внешнего вида приложения CubeeDoo. Нами успешно созданы версии этих интерфейсов, в которых используются только свойства CSS3, исключившие необходимость открывать графические редакторы или создавать дополнительный HTTP-запрос.

С помощью CSS3 и файла формата SVG, рассмотренного в главе 5, действительно можно создавать практически любые формы, не прибегая к использованию готовых изображений.

Но иногда решение об использовании изображений может быть оправданным, поскольку изображения поддерживаются во всех браузерах и их применение не вызывает затруднений. Ряд решений, связанных с использованием изображений, включая изображения границ, а также нескольких фоновых изображений, будет рассмотрен в главе 11.

10 CSS3: преобразования, переходы и анимация

В основном та анимация, которую вам, вероятно, приходилось видеть на настольных компьютерах, не имела никакого отношения к CSS-анимации и реализовывалась посредством использования Flash, Canvas или JavaScript. На мобильных устройствах для оживления элементов, которое невозможно реализовать такими технологиями, важно использовать те преобразования, переходы и анимацию, которые реализуются средствами CSS3.

А почему бы не реализовать все это с помощью Flash, JavaScript или `<canvas>`? Flash не поддерживается и не будет поддерживаться на мобильных устройствах iOS. Создатель Flash, компания Adobe прекратила разработку мобильного средства Flash Player, последним выпуском которого была версия Flash Player 11.1 for Android, и BlackBerry PlayBook в далеком 2011 году.

Разработка Flash продолжается для настольных устройств, но без поддержки или установки Flash на новых устройствах вы потеряете значительную часть, если не весь рынок мобильных устройств. А те, кто будет использовать устаревшие устройства, браузеры которых поддерживают Flash, быстро посадят аккумуляторы.

Аналогично анимация, реализованная на JavaScript, при отсутствии аппаратного ускорения блокирует поток пользовательского интерфейса. При этом все приложение, если не сама анимация, будет работать рывками, не отвечать на действия пользователя, чрезмерно потреблять память и быстро разряжать аккумулятор.

На мобильных устройствах CSS-анимация является прекрасной альтернативой этим технологиям. Браузеры оптимизированы для работы с CSS, следовательно, на эту технологию тратится меньше памяти, ресурсов центрального процессора и энергии аккумулятора[1]. И анимация, преобразования и переходы поддерживаются всеми браузерами современных смартфонов, поэтому нет никаких причин отказываться от их использования.

---

[1] Анимацией определенных значений CSS-свойств также можно посадить аккумулятор, но в целом браузеры хорошо оптимизированы для работы с CSS.

Все это так, но по поводу преобразований, переходов и анимации следует сделать одно важное замечание: то, что их можно применять, еще не означает, что это нужно делать. Инструктивную анимацию нужно применять для демонстрации процедур или задач, которые трудно поддаются описанию с помощью статичных изображений. Сопровождайте инструкции текстом, объясняющим анимацию.

Анимация может применяться для привлечения внимания к элементу на странице, например к сообщению об ошибке отправки формы, об успешном выполнении на одностраничном приложении, или к обновлению важной страницы, которые пользователь может по-другому и не заметить.

Не используйте анимацию, пока не понадобится привлечь внимание, и не анимируйте элементы, предназначенные для чтения или взаимодействия с ними. Разумеется, игры являются исключением из этих правил.

Эти свойства CSS при разумном и умеренном использовании могут быть весьма увлекательными. Но ими не следует злоупотреблять, если только они не используются для игр, иначе ваш сайт станет напоминать сайт Geocities из 1990-х годов.

CSS-переходы

Переходы позволяют CSS-свойствам менять одно значение на другое за определенный период времени. Если вам для изменения цвета ссылки когда-либо приходилось пользоваться псевдоклассом `:hover`, значит, вы пользовались переходом, но с длительностью 0 мс. Используя CSS-переходы, можно добиться изменения цвета и целого набора других свойств за определенный период времени.

CSS-переходы применимы к любому изменению одного состояния на другое. Краткая форма записи свойства перехода складывается из четырех свойств: `transition-property`, которое определяет, какие свойства будут задействованы, `transition-duration`, устанавливающего продолжительность эффекта перехода, `transition-timing-function`, которое обрисовывает порядок ускорений, замедлений или других изменений во время осуществления перехода, и `transition-delay`, которое устанавливает время ожидания перед началом перехода после его инициирования.

Чтобы создать переход, следует установить стили элемента, которые нужно подвергнуть переходу. В свойства исходного состояния включаются название свойства или свойств, задействованных в переходе, наряду со временем перехода, его скоростью и задержкой его начала, если таковые нужны. В спецификации показан следующий синтаксис[1]:

---

[1] Необходимы следующие префиксы производителей: -webkit- до iOS 6 включительно, Android, BlackBerry 10 и Chrome до 25; -o- для Opera вплоть до 12 и -moz- для Firefox вплоть до 15. В IE поддержка начинается с IE10, используется беспрефиксный вариант. Я все же рекомендую включать -webkit-, но остальные префиксы включать не обязательно, поскольку браузеров, требующих эти префиксы, практически не осталось.

```
nav a {
  background-color: rgb(255,255,255);
  border: 5px solid #000000;
  transition-property:background-color;
  transition-timing-function:linear;
  transition-duration:0.8s;
  transition-delay:200ms;
}
```

Это начальный ключевой кадр. Ключевыми кадрами в анимации (или фильме) называются рисунки начальной или конечной точки любого плавного перехода. В переходах ключевые кадры в явном виде не применяются. Они используются для создания анимации, применяющей свойство `animation`, которое будет рассмотрено в данной главе. Но если сейчас понять, что такое ключевой кадр, то легче будет не запутаться в анимации потом.

Затем определяются значения свойств, перечисленных в значении свойства `transition`: в данном случае значения свойства `background-color`. Если бы мы хотели устроить переход от исходного к конечному состоянию для всех способных на это свойств, например при наведении на элемент указателя мыши, то можно было бы воспользоваться ключевым словом `all`:

```
nav a:hover, nav a.hover {
  background-color: rgb(0, 0, 0);
  border: 5px dashed #CCCCCC;
}
```

В предыдущем примере при проходе указателя мыши над ссылкой в разделе навигации документа фон ссылки будет меняться от белого к черному (рис. 10.1). Цвет границы и стиль изменятся сразу же при наведении указателя. Переход фонового цвета начнется по истечении 200 мс, затем пройдут 800 мс перехода, и переход завершится. На рис. 10.1 можно заметить, что граница изменилась сразу же, а цвет фона начал меняться после задержки продолжительностью 200 мс.

Рис. 10.1. Последовательные кадры перехода значения свойства background-color, показывающие, как граница изменилась сразу же после наведения указателя мыши, а цвет фона менялся в течение 800 мс после задержки в 200 мс

Возможно, сразу всего этого не понять, поэтому перейдем к различным значениям переходов. В этом разделе не будет никаких копий экрана, поскольку визуальные эффекты очень трудно перенести в печатное издание. Примеры можно найти в онлайн-ресурсах к главе.

Свойство transition-property

В свойстве `transition-property` объявляется список свойств, которые будут подвергаться переходу в процессе анимации. К свойствам, к которым могут применяться переходы, относятся:

- `background-color`;
- `background-position`;
- `background-size`;
- `border` (цвет и ширина, но не стиль);
- `border-color`;
- `border-radius`;
- `border-width`;
- `border-spacing`;
- `box-shadow`;
- `bottom`;
- `clip`;
- `color`;
- `columns`;
- `column-width`;
- `column-count`;
- `column-gap`;
- `column-rule` (цвет и ширина, но не стиль);
- `crop`;
- `flex`;
- `flex-basis`;
- `flex-grow` (кроме перехода от нуля или к нулю);
- `flex-shrink` (кроме перехода от нуля или к нулю);
- `font-size`;
- `font-size-adjust`;
- `font-stretch`;
- `font-weight`;
- `height`;
- `left`;
- `letter-spacing`;
- `line-height`;
- `margin`;

- max-height;
- max-width;
- min-height;
- min-width;
- opacity;
- order (часть от flex);
- outline-color;
- outline-offset;
- outline-width;
- padding;
- perspective;
- perspective-origin;
- right;
- text-decoration-color;
- text-indent;
- text-shadow;
- top;
- transform;
- transform-origin;
- vertical-align;
- visibility;
- width;
- word-spacing;
- z-index.

ПРИМЕЧАНИЕ

В общем, переходам может подвергаться любая пара значений свойств, в отношении которой можно определить некую среднюю точку. Следует учесть, что в зависимости от аргументов функции распределения скорости перехода по времени средняя точка между двумя значениями может не показываться именно в середине перехода. Такие функции распределения перехода по времени, как ease in, ease out и другие из разряда cubic-bezier, будут иметь разные срединные точки.

Например, можно определить, что средняя точка от состояния `top: 0;` до состояния `top: 100px;` имеет значение `50 px`, следовательно, свойство можно подвергнуть переходу, но от состояния `display: block;` до состояния `display: none;` (используемого в табл. 10.1 в качестве примера свойства) определить среднюю точку невозможно, следовательно, переход для этого свойства недоступен. Можно осуществить переход от `height: 600px;` к `height: 700px;`, но не от `height: auto;` к `height: 700px;`.

Таблица 10.1. Определение пригодности свойства к переходу

Свойство	Исходное значение	Конечное значение	Есть средняя точка?	Пригодность к переходу
height	100 px	200 px	150 px	Да
height	auto	200 px	?	Нет
opacity	0	1	0,5	Да
display	block	none	?	Нет

Устраивать переходы между фоновыми изображениями, включая градиенты, пока что нельзя, но можно получить ряд интересных эффектов, создав переходы для свойств `background-position` и `background-size`.

Исключением из правила средней точки является свойство `visibility`. Казалось бы, видимость является свойством, непригодным для переходов, и тем не менее его можно включать в переходы и анимацию.

Значение может переходить от видимого (visible) к скрытому (hidden) в конце эффекта перехода, если свойство входит в список подвергаемых переходу. Ведутся дискуссии по поводу того, чтобы сделать способными к переходам и анимации все свойства, но ситуация пока не изменилась.

Значением свойства `transition-property` служит список с запятой в качестве разделителя, в котором может быть названо любое количество свойств, или же в качестве значения может быть использовано ключевое слово `all`. Переходу ко времени его инициализации будут подвергаться лишь те свойства, которые перечислены в списке, если только не было объявлено ключевое слово `all`:

```
nav a {
  -webkit-transition-property: background-color; /* iOS6-, BB, Android, Ch25-*/
  -moz-transition-property: background-color; /* FF4 to 15 */
  -o-transition-property: background-color; /* O 10.5 to 12 */
  transition-property: background-color; /* IE10, FF16+, O12.5+, Ch26+, iOS7 */
```

Префиксы `-moz-` и `-o-` добавлены для демонстрации поддержки на устаревших браузерах. Префикс `-ms-` не нужен был никогда, а в используемых в настоящее время мобильных браузерах не нужны префиксы для Firefox и Opera. Единственным префиксом, который нужен свойству `transition` в мобильном пространстве или где бы то ни было, является `-webkit-`.

Если свойство, не подходящее для переходов, будет включено в значение свойства `transition`, это значение будет проигнорировано и его состояние будет изменено мгновенно, без растянутого во времени перехода. Значение, которое со временем не может быть подвергнуто переходу, затронуто не будет, но сам переход сбоя не даст.

Свойство transition-duration

Свойство `transition-duration` позволяет установить продолжительность перехода от первого до последнего ключевого кадра. С его помощью можно установить количество

секунд или миллисекунд, затрачиваемых на анимированный переход от исходного к конечному значению.

Единицы измерения времени рассматривались в главе 8. Здесь они будут применены нами впервые. Следующие две строки полностью эквивалентны друг другу (но предназначены для разных браузеров) и используют такие единицы измерения, как миллисекунды и секунды:

```
-webkit-transition-duration: 0.5s;
    transition-duration: 500ms; ...
```

Свойство transition-timing-function

Функция `transition-timing-function` позволяет управлять переходом, описывая порядок его прохождения во времени. Значением может быть одно из нескольких ключевых слов — `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, `step-start`, `step-end`, `steps(x, start)`, `steps(x, end)` — или значение кубической функции Безье.

Значением кубической кривой Безье являются четыре точки на плоскости: начинаясь в первой точке, она проходит через вторую и достигает последней точки, направляясь к ней от третьей. Если заниматься подобными расчетами года три, то смысл всего этого вполне можно постичь! К счастью, ряд кубических кривых Безье уже включены в CSS3 в виде предопределенных ключевых слов, а ссылки на инструментарий, помогающий разобраться с кубическими кривыми Безье, содержится в онлайн-ресурсах к главе.

Значения, являющиеся ключевыми словами и не относящиеся к функции `step` (`ease`, `linear`, `ease-in-out` и т. д.), представляют собой кубические кривые Безье с четырьмя значениями фиксированных точек:

- `ease`, или `cubic-bezier(0.25, 0.1, 0.25, 1.0)`. Значение по умолчанию, скорость перехода увеличивается к его середине и замедляется к концу;
- `linear`, или `cubic-bezier(0.0, 0.0, 1.0, 1.0)`. Переход осуществляется с постоянной скоростью;
- `ease-in`, или `cubic-bezier(0.42, 0, 1.0, 1.0)`. Переход начинается медленно, а затем его скорость увеличивается вплоть до завершения;
- `ease-out`, или `cubic-bezier(0, 0, 0.58, 1.0)`. Переход начинается быстро, а затем по мере осуществления замедляется;
- `ease-in-out`, или `cubic-bezier(0.42, 0, 0.58, 1.0)`. Переход начинается медленно, затем ускоряется и снова замедляется;
- `cubic-bezier(p1,p2,p3,p4)`. Здесь значения `p1` и `p3` от 0 до 1[1].

[1] Хотя объяснение кубической кривой Безье и не входит в круг вопросов, рассматриваемых в данной книге, есть весьма хорошее инструментальное средство, с помощью которого можно определить, какие значения могут еще понадобиться для функции распределения скорости перехода по времени. Еще один сайт, http://cubicbezier.com, позволяет сравнивать развитие перехода при использовании разных функций.

Step-функции — steps(x, end), steps(x, start), step-end и step-start — делят время перехода на равные интервалы. Каждый интервал является равнопротяженным по времени шагом на пути перехода от исходного к финальному состоянию. Функция также определяет, где находится шаг, в начале или в конце интервала.

Иными словами, если используются пять шагов, steps(5, start) будет устанавливать шаги на 0, 20, 40, 60 и 80 % по направлению к финальному состоянию. При steps(5, end) будут получены шаги, установленные на 20, 40, 60, 80 и 100 % по направлению перехода.

Значение step-start эквивалентно steps(1, start), а значение step-end эквивалентно steps(1, end). Значения steps() хорошо подходят для анимированных спрайтов фонового изображения при создании анимации. В онлайн-ресурсах к главе имеется ряд примеров использования этих значений. Продолжим работу с нашим примером кода:

```
  ...
-webkit-transition-timing-function: linear;
    transition-timing-function: linear;
  ...
```

Свойство transition-delay

Свойство transition-delay определяет количество миллисекунд или секунд ожидания между запуском изменения состояния, вызванного переходом, и началом эффекта перехода.

Значение по умолчанию 0s показывает, что анимация перехода должна начинаться немедленно. Положительное значение времени приведет к задержке начала эффекта перехода на указанное значение. Отрицательное значение вызывает немедленное начало перехода, но с эффекта, который должен быть в середине этого перехода.

Хотя свойство transition-delay может казаться бесполезным, оно способно существенно улучшить пользовательское восприятие. Зачастую не хочется делать эффекты прохода указателя или прикосновения слишком чувствительными. Если пользователь быстро проводит пальцем или указателем мыши по экрану из точки A в точку B, не хочется, чтобы все точки между ними, к которым он случайно прикоснется, реагировали на это прикосновение слишком быстро. Обычно нежелательный эффект предотвращается задержкой перехода в 50 мс. При этом при преднамеренных прикосновениях не теряется ощущение отклика, но страница не реагирует на непреднамеренную активизацию объекта, в отношении которого намечен переход.

Отрицательные задержки перехода также могут послужить намеченной цели. Пока абсолютное значение задержки будет меньше значения свойства transition-duration, переход, будучи инициированным, начнется с промежуточного состояния, с точки, пропорциональной разнице во времени. Например, если время перехода, заданное с помощью свойства transition-duration, составляет 10 с, а задержка имеет значение -4, переход начнется немедленно, но с 40 % пути перехода:

```
  ...
  -webkit-transition-delay: 250ms
    transition-delay: 0.25s;
}
```

Краткая форма записи свойства transition

Все это вместе похоже на какую-то путаницу:

```
nav a {
  -webkit-transition-property: background-color;
    transition-property: background-color;

  -webkit-transition-duration: 0.5s;
    transition-duration: 500ms;

  -webkit-transition-timing-function: linear;
    transition-timing-function: linear;

  -webkit-transition-delay: 250ms;
    transition-delay: 0.25s;
}
```

Вместо задания перехода с помощью 8, 12 или 16 строк кода[1] можно воспользоваться краткой формой записи свойства `transition`, собирающей воедино все только что рассмотренные четыре свойства. Следует заметить, что порядок следования значений продолжительности и задержки должен соблюдаться (то есть он должен сохраняться прежним):

- `transition-property`;
- `transition-duration`;
- `transition-timing-function`;
- `transition-delay`.

```
nav a {
  background-color: #FFFFFF;
  -webkit-transition: background-color 500ms linear 250ms;
    transition: background-color 500ms linear 250ms;
}
nav a:hover, nav a.hover {
  background-color: #FF0000;
}
```

Если привести переход в исходное состояние, то при проходе указателя над элементом или изменении его класса свойства, подвергаемые переходу, изменят свои значения на новые, а при уходе указателя мыши (завершении прикосновения) или удалении нового класса произойдет обратный переход. Свойства перехода можно перевести в исходное или какое-нибудь другое состояние. Все зависит от

[1] В зависимости от количества включаемых префиксов производителей.

того, когда и как вы собираетесь реализовать постепенный переход элемента из одного состояния в другое.

Множественные переходы

А как осуществить постепенные переходы сразу нескольких свойств? Возможно, понадобится применить одно свойство перехода для изменения свойств background-color, border-color и color? Свойства transition позволяют осуществлять сразу несколько переходов за один вызов.

Предположим, что вместо простого перехода свойства background-color нужно получить еще и переход свойства границы — border (для этого свойства доступен переход к другому цвету или к другой ширине, но не к другому стилю). Для этого нужно будет, во-первых, включить новое свойство border в объявление стиля, подвергаемого переходу, во-вторых, включить свойство border в список значений свойства transition-property либо через запятую наряду с другими названиями свойств, либо путем использования ключевого слова all. Следует учесть, что all нужно использовать только в том случае, если требуется осуществить одинаковый переход для всех свойств.

Например, в ранее показанных примерах в состояниях прохождения указателя мыши или прикосновения определялись оба свойства и границы, и фонового цвета, однако в качестве значения свойства transition-property было включено только свойство фонового цвета. В этих примерах переход значения background-color осуществлялся медленно (за 250 мс), а цвет границы менялся сразу же, как только проходил указатель или совершалось прикосновение, привычным образом, как и всегда без использования перехода или как будто продолжительность перехода была установлена в 0s с задержкой в 0s.

Если нужно получить изменение обоих свойств в одинаковом темпе и с одинаковой задержкой, можно в краткой записи свойства transition воспользоваться ключевым словом all, поскольку переходу подвергаются все свойства, перечисленные в состоянии hover:

```
nav a {
  background-color: #FFFFFF;
  border: 5px solid #CCCCCC;
  -webkit-transition: all 500ms linear 250ms;
    transition: all 500ms linear 250ms;
}
```

При использовании ключевого слова all все свойства подвергаются переходу в едином темпе, с одинаковыми скоростью и задержкой. Если нужно применить переход к части свойств так, чтобы он осуществлялся в едином темпе, с одинаковыми скоростью и задержкой, следует указать эти свойства в transition-property через запятую:

```
nav a {
  background-color: #FFFFFF;
  border: 5px solid #CCCCCC;
```

```
  color: red;
  -webkit-transition: border, color 500ms linear 250ms;
    transition: border, color 500ms linear 250ms;
}
```

Если не нужно, чтобы все свойства подвергались переходу в едином темпе, а необходимо, чтобы у одних задержка была больше, чем у других, или чтобы эффект перехода применялся только к некоторым свойствам, укажите различные свойства, подвергаемые переходу, через запятую, включая для каждого как минимум `transition-property` и `transition-duration`:

```
nav a {
  background-color: #FFFFFF;
  border: 5px solid #CCCCCC;
  color: red;
  -webkit-transition:
    background-color, color 500ms linear 750ms,
    border 500ms linear 250ms;
  transition:
    background-color, color 500ms linear 750ms,
    border 500ms linear 250ms;
}
```

В данном примере граница (`border`), у которой значение свойства `transition-delay` меньше, подвергнется переходу первой. После того как переход границы завершится, а это произойдет на отметке `750ms`, которая является значением свойства `transition-delay` для цвета фона и складывается из значения `500ms`, установленного для перехода границы (`border`), и значения `250ms`, установленного для задержки этого перехода, через полсекунды завершится переход фонового цвета и цвета первого плана (`background-color` и `color`):

```
transition-property:   background-color, color, border;
transition-duration:   500ms;
transition-timing-function: linear;
transition-delay:     750ms, 750ms, 250ms;
```

Можно также воспользоваться развернутой формой записи свойств, отделяя свойства друг от друга запятыми, но тогда в сценарии, как показано ранее, будет больше кода. Я считаю, что краткая запись проще, лаконичнее и понятнее и код с такой записью легче обслуживать.

В CubeeDoo переход, заключающийся в перевороте карты, осуществляется за 0,25 с. Переворачивать карту мы еще не научились (эта тема будет рассмотрена в следующем разделе), но с помощью приведенного далее кода можем приказать все перевернуть без задержки и за 0,25 с:

```
1   #board > div {
2     position: relative;
3     width: 23%;
4     height: 23%;
5     margin: 1%;
```

```
6     float: left;
7     -webkit-transition: 0.25s;
8     transition: 0.25s; ...
```

В исходном состоянии до переворота, в строках 7 и 8, картам сообщается, что, когда будет изменяться состояние, нужно немедленно (в соответствии с используемым по умолчанию значением свойства `transition-delay`) осуществить переход всех свойств (что является значением по умолчанию для свойства `transition-property`) с используемым по умолчанию для свойства `transition-timing-function` значением `ease`.

Я воспользовалась кодом `transition: 0.25s`, но могла бы написать и `transition: all 0.25s ease 0ms;`, что было бы длиннее, но проще в обслуживании, поскольку в последней записи авторские намерения выражены более четко.

CSS3-преобразования

Преобразования позволяют изменять размер элементов, вращать, наклонять, перемещать их и иными способами менять их позицию. Существует 2D-версия модуля преобразования, которая поддерживается всеми браузерами, начиная с IE9[1], и еще одна версия для 3D, которую начинают поддерживать все большее количество браузеров.

ПРИМЕЧАНИЕ

В отличие от переходов, преобразования в IE9 поддерживаются.

К элементу могут применяться CSS3-преобразования, включая множественные преобразования в отношении отдельно взятого элемента.

Для создания преобразований используются два CSS-свойства: `transform`, определяющее тип применяемых к элементу преобразований, и `transform-origin`, устанавливающее исходную точку, из которой начнется преобразование.

Свойство transform-origin

Сначала рассмотрим установку исходной точки преобразования элемента, которая задается значением свойства `transform-origin`.

По умолчанию для свойства `transform-origin` используется значение `50% 50% 0`, представляющее собой центр элемента. Первое значение указывает координату *x*, или перемещение влево или вправо, а второе значение указывает координату *y*, или перемещение вверх или вниз, эти значения вычисляются от верхнего левого угла элемента. Третье значение задает *z*-смещение, имеющее смысл при 3D-преобразованиях. Значения можно указывать в единицах измерения длины, в процентах или ключевыми словами `left`, `center`, `right`, `top` и `bottom`, а *z*-смещение задавать не обязательно, но

[1] Более ранние версии Internet Explorer поддерживают преобразования через фильтр: progid:DXImageTransform.Microsoft.Matrix().

здесь в единицах длины может указываться только то, что не может быть выражено в процентах.

В значении свойства `transform-origin` устанавливается та самая точка, вокруг которой будет осуществляться преобразование. Когда исходная точка установлена в центре элемента (это значение по умолчанию), его преобразование, в данном случае вращение, будет выполняться вокруг этой точки (рис. 10.2). Когда исходная точка в свойстве `transform-origin` устанавливается в другом месте, допустим в верхнем левом углу, как показано на рисунке, преобразование, например вращение, осуществляемое вокруг нее, создаст совершенно иной эффект. Элемент движется по орбите вокруг этой точки.

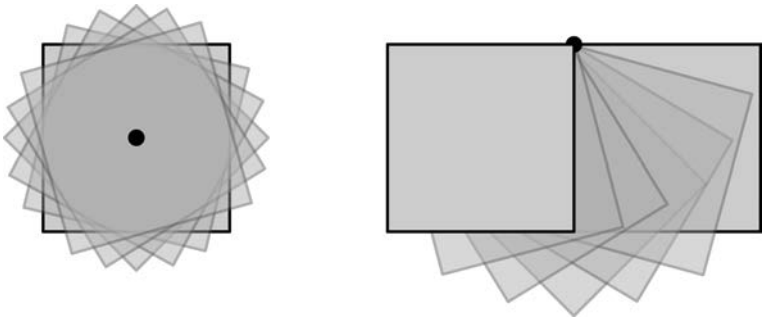


Рис. 10.2. Исходные точки, установленные по умолчанию в верхний левый угол, коренным образом меняют эффект преобразования, демонстрирующий вращение элемента на 90°

Для эффекта в левой части рис. 10.2 взято значение свойства `transform-origin` по умолчанию, поэтому само свойство может быть опущено. Синтаксис для эффекта, показанного в правой части этого рисунка, может иметь следующий вид:

```
-webkit-transform-origin: top left 0; /* для всех браузеров webkit и blink */
   -moz-transform-origin: top left; /* для FF 3.5 - 15 */
    -ms-transform-origin: 00; /* для IE9 */
     -o-transform-origin: 000; /* для O 11.0-12.0 */
        transform-origin: top 00; /* только для IE10+, FF16+, O12.1 only */
```

где `top left, 00, 000` и `top left 0` эквивалентны[1].

После установки исходной точки (или пропуска этой установки и использования значения по умолчанию `transform-origin: center center 0;`) применяется тип преобразования, устанавливаемый с помощью свойства `transform`, в качестве значения которого используется список, состоящий из одного или нескольких преобразований.

---

[1] Префикс производителя нужен для IE9, Firefox 3.5 до версии 15, Opera до версии 12 включительно и для всех версий браузеров WebKit. Он больше не нужен для Firefox 16, IE10, Opera 12.1 и Opera Mobile 11. Opera Mini преобразования не поддерживает. Все браузеры, поддерживающие transform-origin, поддерживают все функции 2D-преобразований. Браузер Opera не требовал префикса в версии 12.1, но ему стал нужен префикс -webkit-, когда разработчики этого браузера отказались от использования движка Presto.

Свойство transform

CSS-свойство `transform`, которое еще до появления iPhone поддерживалось в Firefox 3.5+, Opera 10.5, Internet Explorer 9 и WebKit, позволяет изменять пространство координат модели визуального форматирования CSS. При использовании этого свойства элементы могут подвергаться перемещению, вращению, масштабированию и наклону. CSS-преобразования изменяют пространство координат, позволяя менять позицию подвергаемого преобразованию содержимого без нарушения обычного технологического процесса. Преобразуемый элемент занимает то местоположение и пространство, которые использовались им перед применением преобразования.

С помощью функции преобразования можно манипулировать внешним видом элемента. Значением свойства `transform` является список разделенных запятыми функций преобразований, применяемых в указанном порядке. В число этих функций входят рассматриваемые далее.

translate()

Функция `translate(x, y)` (рис. 10.3) похожа на относительное позиционирование, перемещение или изменение местоположения элемента на значение *x* слева и на значение *y* сверху:

```
-webkit-transform: translate(15px, -15px);
    -ms-transform: translate(15px, -15px);
        transform: translate(15px, -15px);
```

translateX()

Функция `translateX(x)` похожа на функцию `translate()`, но для нее указывается только значение для перемещения влево или вправо:

```
-webkit-transform: translatex(15px);
    -ms-transform: translatex(15px);
        transform: translatex(15px);
```

translateY()

Функция `translateY(y)` похожа на функцию `translate()`, но для нее указывается только значение для перемещения вверх или вниз:

```
-webkit-transform: translatey(-15px);
    -ms-transform: translatey(-15px);
        transform: translatey(-15px);
```

scale()

Свойство `scale(w, h)` (рис. 10.4) масштабирует элемент на *w* единиц по ширине и на *h* единиц по высоте:

```
-webkit-transform: scale(1.5, 2);
    -ms-transform: scale(1.5, 2);
        transform: scale(1.5, 2);
```

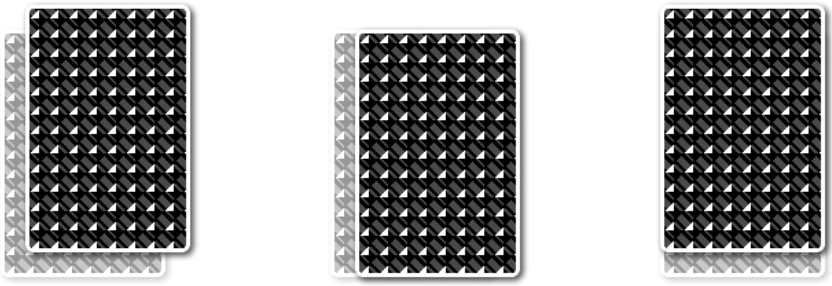


Рис. 10.3. Функции преобразования translate: *слева направо* — translate(15px, -15px), translateX(15px) и translateY(-15px)

Если указано только одно значение, будет проведено пропорциональное масштабирование. Поскольку искажать элемент, скорее всего, не захочется, у этой функции преобразования обычно можно увидеть только один параметр:

```
transform: scale(2);
```

Следует учесть, что при использовании преобразования с целью увеличения размера элемент может получиться нерезким, чего и следует ожидать при масштабировании изображений. Поэтому в качестве общей рекомендации я могу предложить вам при необходимости увеличения размера сделать исходным уменьшенный элемент, а затем увеличить его, воспользовавшись функцией `scale(1)`.

scaleX()

Функция `scaleX(w)` похожа на функцию `scale()`, но для нее указывается только ширина. Это аналогично объявлению `scale(w, 1)`:

```
-webkit-transform: scalex(0.5);
    -ms-transform: scalex(0.5);
        transform: scalex(0.5);
```

ПРИМЕЧАНИЕ

Префиксы -o- и -moz- не применяются для преобразований, так как Mozilla и Presto по умолчанию поддерживают трансформации.

scaleY()

Функция `scaleY(h)` похожа на функцию `scale()`, но для нее указывается только высота. Это аналогично объявлению `scale(1, h)`:

```
-webkit-transform: scaley(2);
    -ms-transform: scaley(2);
        transform: scaley(2);
```

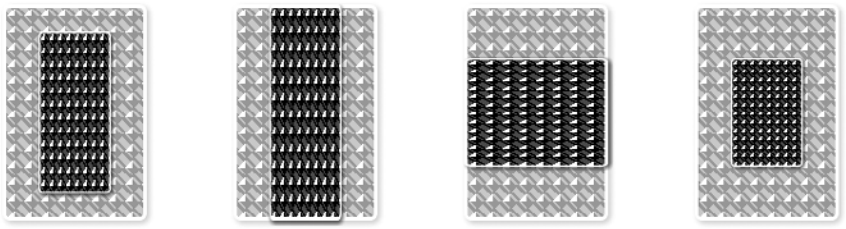


Рис. 10.4. Функции преобразования scale: *слева направо* — scale(0.5, 0.75), scaleX(0.5), scaleY(0.5) и scale(0.5

rotate()

Функция `rotate(`*`угол`*`)` (рис. 10.5) будет вращать элемент вокруг исходной точки (как было изображено на рис. 10.2) на указанную угловую величину:

```
-webkit-transform: rotate(15deg);
    -ms-transform: rotate(15deg);
        transform: rotate(15deg);
```

rotateX()

Функция `rotateX(`*`угол`*`)` будет вращать элемент вокруг его оси *X*:

```
-webkit-transform: rotatex(15deg);
    -ms-transform: rotatex(15deg);
        transform: rotatex(15deg);
```

Вращение элемента на 90° по оси *X* приведет к его исчезновению, а его вращение на 180° приведет к полному перевороту, и вы увидите его тыльную сторону, а лицевая сторона станет тыльной. Видимость содержимого, показываемого после переворота, можно установить с помощью рассмотренного далее свойства `backface-visibility`.

rotateY()

Подобно `rotateX()`, функция `rotateY(`*`угол`*`)` будет вращать элемент на указанный угол вокруг его оси *Y*:

```
-webkit-transform: rotatey(15deg);
    -ms-transform: rotatey(15deg);
        transform: rotatey(15deg);
```

В анимации CubeeDoo при щелчке пользователя на карте используется функция `rotateY(180deg)`, переворачивающая контейнер карты, при этом тыльная сторона карты исчезает и мы видим ее лицевую сторону.

Затем карта с помощью функции `rotate(0deg)` анимируется в обратном порядке или же выбирается функция `rotate(360deg)` для более длительного вращения, если пользователь потерпел неудачу в выборе соответствия.

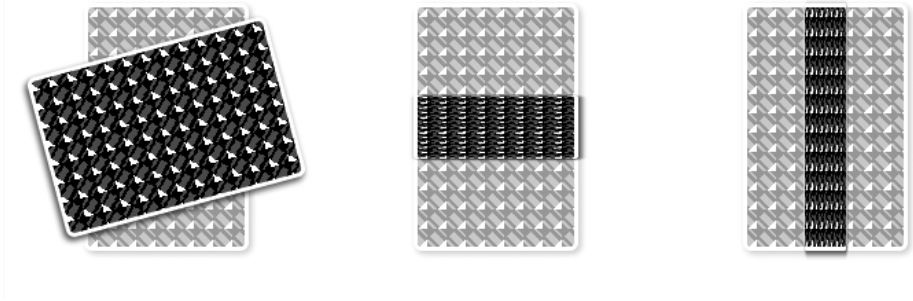


Рис. 10.5. Функции преобразования rotate: *слева направо* — rotate(75deg), rotateX(75deg) и rotateY(75deg) (rotateX осуществляет вращение вокруг оси *X*, и при указании угла 90° элемент исчезнет, а вращение текста на угол, превышающий 90°, приведет к появлению его перевернутого изображения)

skew()

Функция `skew(x, y)` (рис. 10.6) указывает степень наклона вдоль осей *X* и *Y*. Значение *x* определяет наклон по оси *X*, а значение *y* — по оси *Y*. Если указан только один параметр, то функция будет работать точно так же, как при использовании функций `skew(x, 0deg)` или `skewX(x)`. Значения указываются в углах, градусах, оборотах или градах:

```
-webkit-transform: skew(15deg, 4deg);
    -ms-transform: skew(15deg, 4deg);
        transform: skew(15deg, 4deg);
```

skewX()

Функция `skewX(x)` похожа на функцию `skew()`, но для нее указывается только значение наклона по оси *X*, и наклон будет осуществляться не по осям *X* и *Y*, а только по оси *X*. Верхняя и нижняя стороны блока останутся на прежнем уровне, а левая и правая будут наклонены:

```
-webkit-transform: skewx(15deg);
    -ms-transform: skewx(15deg);
        transform: skewx(15deg);
```

skewY()

Функция `skewY(y)` похожа на функцию `skew()`, но для нее указывается только значение наклона по оси *Y*. Ее использование эквивалентно объявлению `skew(0deg, y)`. Левая и правая стороны остаются вертикальными, а верхняя и нижняя будут наклонены:

```
-webkit-transform: skewy(-3deg);
    -ms-transform: skewy(-3deg);
        transform: skewy(-3deg);
```

Рис. 10.6. Функции преобразования skew: *слева направо* — skew(15deg, 15deg), skewX(-15deg) и skewY(-15deg)

Объявления `skewX(x)` или `skewY(y)` похожи, но не являются точными аналогами объявлений `skew(x, 0deg)` и `skew(0deg, y)` соответственно. Если сделать в отношении элемента одно из двух объявлений, то действительно не будет никакой разницы, но объявление `skew(x, 0deg)` и `skew(0deg, y)` приведет к тому, что последнее объявление перепишет более раннее, а `skewX(x)` или `skewY(y)` будут эквивалентны написанию `skew(x, y);`, потому что вместо отмены одного свойства другим свойства будут объединены. Работа функций наклона показана на рис. 10.6.

Множественные преобразования

В предыдущем разделе рассматривались одиночные преобразования, но элемент можно подвергнуть более чем одному преобразованию. Чтобы задать более одного преобразования, нужно просто разделить функции преобразования пробелами:

```
.enlargen:hover, .enlargen.hover {
  -webkit-transform: translate(-50%, -50%) scale(2) rotate(0deg);
      -ms-transform: translate(-50%, -50%) scale(2) rotate(0deg);
          transform: translate(-50%, -50%) scale(2) rotate(0deg);
}
```

Этот код сделает элемент в два раза выше и в два раза шире. За счет перемещения элемента на 50 % вверх и влево нижний правый угол останется на том же месте, что и прежде. Объявление `rotate(0deg)` здесь лишнее, поскольку любые преобразования, объявленные с использованием селектора с менее слабой конкретизацией и содержащие функцию `rotate`, будут переписаны независимо от того, задана эта функция или нет. Я включила ее в код в качестве примера того, как делать это с функцией преобразования `rotate()`, и чтобы напомнить вам, как включать единицы измерения, не имеющие отношения к длине.

ВНИМАНИЕ

Хотя значение для вращения установлено на 0°, это не отменяет для него установки единицы измерения в виде градусов. То же самое касается времени (s или ms), радианов — rad, градов — grad, оборотов — turn, герц — Hz и килогерц — kHzs.

Следует заметить, что значения свойства `transition-property` отделяются друг от друга запятыми, а значения свойства `transform` — пробелами.

Класс `enlargen` (увеличивающиеся) может пригодиться для добавления к галерее изображений с целью выделения изображения, над которым находится указатель, путем увеличения его в четыре раза (в два раза по ширине и высоте). Это позволит прекратить всякие споры насчет того, что могло бы быть интересным в виде миниатюр, но было бы некрасивым в полном размере.

matrix(). Функция преобразования `matrix()` является единственной функцией, определяющей перемещение, наклон, вращение и масштабирование элемента. Ей передаются шесть параметров. Если используется инструмент для создания преобразований, программное обеспечение обычно создает функцию `matrix`, а не четыре отдельные функции преобразования. Следующие две строки могут быть эквивалентны друг другу, а два последних значения функции `matrix` зависят от размера и местоположения того элемента, который подвергается преобразованию:

```
transform: translate(-50%, -50%) scale(2) rotate(0deg);
transform: matrix(2, 0, 0, 2, -100, -172.5)
```

Обычно если в разметке CSS встречается функция `matrix`, она является результатом компьютерной генерации. Думаю, у вас вряд ли когда-нибудь появится желание записывать значения `matrix`. Я рассмотрела эту функцию с целью объяснить, что это такое, если вам когда-либо придется с ней столкнуться, но не стала объяснять, что она означает на самом деле.

Преобразования в составе переходов

В разделе, посвященном свойству `transition-property`, были перечислены все свойства, которые могут подвергаться преобразованиям. Если при объявлении свойства `transition` были использованы ключевое слово `all` или краткая форма записи `transition`, то в качестве составной части установки для всех (`all`) будет включено и свойство `transform`.

Можно объявить свойства `transform` по отдельности, в виде части списка свойств `transition`, перечисленных через запятую. Если включать в список фактическое преобразование, то оно, если браузер того требует, должно включать префикс производителя браузера:

```
p {
  -webkit-transition: -webkit-transform 500ms linear 250ms;
          transition: transform 500ms linear 250ms;
   -webkit-transform: translate(0) rotate(0deg);
       -ms-transform: translate(0) rotate(0deg);
           transform: translate(0) rotate(0deg);
```

```
}
p:hover {
  -webkit-transform: translate(100px, -100px) rotate(90deg);
      -ms-transform: translate(100px, -100px) rotate(90deg);
          transform: translate(100px, -100px) rotate(90deg);
            padding: 3px;
             border: 5px solid #00ff00;
}
```

Префикс `-ms-` был включен для свойства `transform`, но не для свойства `transition`, поскольку переходы получили поддержку только в IE10, а преобразования предваряются префиксами в IE9, но не в IE10.

Функции 3D-преобразований

Браузеры не спешат поддерживать 3D-преобразования, но когда-нибудь все они к этому придут. С выходом iOS 3.2, Android 3, BlackBerry 10, Firefox 10, IE10, Safari 4, и Chrome 12 3D-преобразования начали поддерживаться, но только с префиксами производителей. 3D-преобразования намечены к поддержке и в Opera 15 при использовании средства визуализации Blink. 3D-преобразования начали поддерживаться только с появления iPhone 2 (не исходной модели) и только при наличии Mac OS X v10.6 или более новой версии.

В CSS 3D-преобразования позволяют позиционировать элементы на странице в трехмерном пространстве. Так же как и раньше, для создания 3D-перемещений 3D-преобразования можно комбинировать с переходами (и с анимацией, которая будет рассмотрена позже).

Подобно функциям 2D-преобразований, большинство браузеров, начиная с IE10, поддерживают свойства 3D-преобразований. На момент написания данной книги для браузеров на основе WebKit и Blink все еще требовалось указывать префикс `-webkit-`.

По поводу 3D-преобразований следует сделать пару замечаний: во-первых, для преобразования элементов в 3D-пространстве используется аппаратное ускорение, во-вторых, у таких элементов имеется собственное накопительное окружение.

translate3d()

Функция `translate3d(x, y, z)` перемещает элемент на значение *x* вправо, на значение *y* вверх и на значение *z* по направлению к пользователю (или от него, если значение отрицательное). В отличие от значений *x* и *y* значение *z* не может быть задано в процентах.

translateZ()

Функция `translateZ(z)` похожа на функцию `translate3d()`, но оказывает воздействие только на позиционирование *z*. При положительном значении *z* элемент перемещается в сторону пользователя, а при отрицательном — от пользователя. Параметр может указываться в любых единицах измерения длины, кроме процентов.

Благодаря преимуществам аппаратного ускорения функция `translateZ(0)` зачастую используется в качестве универсального средства решения всех проблем (подобно тому как объявление `zoom:1` было панацеей в IE6), чтобы визуализация возлагалась не на центральный процессор, а на графический, что улучшит ее воспроизведение.

Рисование лучше дается графическому, а не центральному процессору. При анимации с небольшими прорисовками получение элемента поднимается в графическом процессоре на его собственный уровень, который называется RenderLayer, и выполняется быстрее, делая анимацию более качественной. Находясь на собственном уровне, любое 2D-преобразование, 3D-преобразование или изменение прозрачности может производиться исключительно с помощью графического процессора с сохранением очень высокой скорости, обеспечивающей частоту смены кадров более 60 кадров/с.

Следует учесть, что, поскольку элементы в 3D-среде имеют собственное накопительное окружение, любые элементы, которые должны появиться над элементами, подвергаемыми преобразованию (как будто у них более высокий *z*-индекс), и не вложенные в преобразуемый элемент, также должны быть преобразованы в 3D-пространство. Для этого разработчики прикрепляют к элементам, для которых в иной ситуации не требуется преобразование, объявление `transform: translateZ(0)`.

scale3d()

Функция `scale3d(w, h, z)` масштабирует ширину (*w*), высоту (*h*) и *z*-шкалу (*z*) элемента. При этом *z*-шкала влияет на масштабирование преобразуемых элементов по оси *Z*.

scaleZ()

Функция `scaleZ(z)` похожа на функцию `scale3d()`, но ей предоставляется только значение для *z*-шкалы, и она оказывает влияние на масштабирование по оси *Z* элемента и его потомков, не имеющих абсолютного позиционирования.

rotate3d()

Функция `rotate3d(x, y, z, angle)` вращает элемент в 3D-пространстве. Первые два параметра просто задают вращение элемента вокруг горизонтальной и вертикальной осей. Угол (*angle*) может задаваться в углах (`deg`), радианах (`rad`) или градах (`grad`). Последний параметр позволяет вращать элемент вокруг произвольного вектора в 3D-пространстве; вектор, вокруг которого нужно осуществить вращение, должен быть задан параметрами *x*, *y* и *z*. А внешний вид должен быть нормализован браузером.

perspective()

Функция преобразования `perspective(p)` позволяет помещать в матрицу преобразования перспективу, задавая ее для отдельно взятого элемента:

```
transform: perspective(100px) rotatey(3deg);
```

Перспектива может быть применена также с помощью свойства perspective, действие которого распространяется на дочерние элементы данного родительского элемента.

Дополнительные свойства 3D-преобразований

Для того чтобы вы могли успешно осуществлять 3D-преобразования, в дополнение к ранее представленному ряду свойств, связанных с преобразованиями, мы рассмотрим некоторые новые свойства и усовершенствования некоторых свойств, предоставленных для 2D-преобразований.

Возвращение к свойству transform-origin

Ранее мы уже узнали, что свойство transform-origin устанавливает для элемента исходную точку перехода. В 3D-преобразованиях это свойство теперь допускает установку значения *z*-смещения. Свойство transform-origin получает три значения, позволяющие указать *z*-смещение, задающее точку начала преобразования:

```
transform-origin: 0 0 500px;
```

Наряду с тем, что этот эффект получил поддержку только с появлением Safari 4+ на Mac OS X v10.6 и более новых версий и iPhone 2.0 и более новых версий (но не на первом iPhone), более ранние версии, поддерживающие transform-origin, делают последнее в том случае, если объявляются два значения.

Свойство perspective

Свойство perspective не следует путать с функцией 3D-преобразования perspective(). Свойство perspective, записанное с префиксом производителя, используется для придания иллюзии глубины и определяет изменения размеров на основе своего *z*-смещения, когда за плоскость берется значение $z = 0$.

Появившийся на плоскости $z = 0$ объект имеет свой обычный размер. При *z*-смещении $p/2$ (на половину расстояния между наблюдателем и плоскостью $z = 0$) объект будет выглядеть вдвое больше, а при *z*-смещении p — в половину своего размера. Таким образом, чем больше значения, тем меньше чувствуется перспектива, а чем они меньше, тем перспектива чувствуется больше. Для основной части содержимого приемлемо выглядящий результат дают значения между 500 и 1000 пикселами.

По умолчанию исходной точкой для эффекта перспективы является центр блока границы элемента, но этой точкой можно управлять, задав соответствующее значение свойству perspective-origin.

Перспектива не воздействует на объект напрямую — она воздействует на внешний вид подвергаемых 3D-преобразованиям потомков этого элемента, позволяя этим потомкам совместно использовать одну и ту же перспективу при перемещении в окне просмотра.

Вполне закономерен вопрос: в чем же разница между объявлением perspective: 600px и объявлением transform: perspective(600px)? Первое объявление делается

в отношении родительского элемента, чтобы у всех его потомков была одна и та же точка схода. А второе объявление делается в отношении самого преобразуемого элемента для придания ему его собственной перспективы.

Свойство transform-style

Свойство `transform-style` определяет, как в 3D-пространстве выводятся на экран вложенные элементы. Все эффекты 3D-преобразований являются просто эффектами рисования. Преобразуемые дочерние элементы по-прежнему выводятся в плоскости своего родительского элемента, иными словами, они плоские. Начиная создание иерархии объектов с 3D-преобразованиями, родительские и дочерние элементы должны жить в общем трехмерном пространстве и использовать общую неплоскую перспективу, распространяющуюся из какого-нибудь контейнера. Здесь вступает в игру свойство `transform-style`.

Это свойство принимает одно из двух значений: `flat` и `preserves-3d`. Используемое по умолчанию значение `flat` делает преобразованные дочерние элементы плоскими, проецируя их на плоскость их родительского элемента. Значение `preserves-3d` предписывает дочерним элементам пребывать в общем 3D-пространстве их родительского элемента.

Свойство backface-visibility

Свойство `backface-visibility` определяет, будет элемент виден или нет, когда он подвергается преобразованию, при котором его тыльная сторона поворачивается к зрителю. Свойство принимает одно из двух значений: `visible` (используется по умолчанию) или `hidden`. Например, в CubeeDoo мы переворачиваем двухсторонние карты, чтобы показывать их лицевую часть, когда карта выбрана. Когда мы видим лицевую часть, нам не нужно видеть тыльную часть карты, и наоборот, поэтому для свойства `backface-visiblity` нужно установить значение `hidden`.

А теперь все вместе

При переворачивании карт в CubeeDoo мы используем множество только что перечисленных свойств:

```
#board > div {
  position: relative;
  width: 23%;
  height: 23%;
  margin: 1%;
  float: left;
  -webkit-transition: 0.25s;
  transition: 0.25s;
  -webkit-transform: rotatey(0deg);
  transform: rotatey(0deg);
  -webkit-transform-style: preserve-3d;
  transform-style: preserve-3d;
  box-shadow: 1px 1px 1px rgba(0,0,0,0.25);
```

```
  cursor: pointer; /* for desktop */
}
#board.level2 > div {
  height: 19%;
}
#board.level3 > div {
  height: 15%;
}
.back,
.face,
.back:after,
.face:after {
  position: absolute;
  content: "";
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  border-radius: 3px;
  pointer-events: none;
  -webkit-backface-visibility: hidden;
  backface-visibility: hidden;
}
.back {
  border: 5px solid white;
}

.back:after {
  font-size: 2.5rem;
  line-height: 100%;
  background:
    50% 50% no-repeat,
    0 0 no-repeat #fff;
  font-style: normal;
  box-shadow: inset 1px 1px 0 currentcolor,
    inset -1px -1px 0 currentcolor,
    1px 1px 1px rgba(0,0,0,0.1);
  color: rgb(119, 160, 215);
  background-image:
    url('data:image/svg+xml;utf8,<svg width="40" height="40"
    xmlns="http://www.w3.org/2000/svg"><g><text xml:space="preserve"
    text-anchor="middle" font-family="serif" font-size="40" id="svg_1"
    y="30" x="20" stroke-width="0" stroke="rgb(119, 160, 215)"
    fill="rgb(119, 160, 215)">□</text></g></svg>'),
    -webkit-linear-gradient(-15deg,
    rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.025));
  background-image:
    url('data:image/svg+xml;utf8,<svg width="40" height="40"
      xmlns="http://www.w3.org/2000/svg"><g><text xml:space="preserve"
      text-anchor="middle" font-family="serif" font-size="40"
      id="svg_1"
```

```
      y="30" x="20" stroke-width="0" stroke="rgb(119, 160, 215)"
      fill="rgb(119, 160, 215)">□</text></g></svg>'),
    linear-gradient(75deg,
      rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.025));
  -webkit-transform: rotatey(0deg);
  -webkit-transform: rotatey(0deg) translatez(0);
  transform: rotatey(0deg)
  transform: rotatey(0deg) translatez(0);
}
.face {
  -webkit-transform: rotatey(180deg);
  -ms-transform: rotatey(180deg);
  transform: rotatey(180deg);
}
#board > div.flipped {
  -webkit-transform: rotatey(180deg);
  -webkit-transform: rotatey(180deg) translatez(0);
  transform: rotatey(180deg);
  transform: rotatey(180deg) translatez(0);
  box-shadow: -1px 1px 1px rgba(0,0,0,0.25);
}
```

В CubeeDoo преобразование используется для переворачивания карты, а переход — для того чтобы переворачивание занимало 250 мс. Карты являются оболочкой для двух дочерних элементов: лицевой и тыльной стороны карты. Поскольку для стилизации лицевой стороны карты используются классы CSS, лицевые стороны всех карт можно создать с помощью сгенерированного содержимого. Единый путь кода обслуживать проще. Поэтому, несмотря на имеющуюся возможность поместить цветовую схему непосредственно на лицевую сторону с помощью свойства `background-color` и поместить в качестве значения свойства `background-image` SVG-образы непосредственно в `<div>`-контейнер лицевой стороны, цифра и вторая тема образа должны быть сгенерированным содержимым. Чтобы упростить код, мы поместили SVG и цвета в псевдоэлемент `::after`, который также генерирует содержимое.

Когда пользователь касается карты или щелкает на ней, карта переворачивается. Этот переворот выполняется с помощью объявления `transform: rotatey(180deg) translatez(0);`, примененного к контейнеру карты. Проблема в том, что в HTML исходный порядок таков, что контейнер тыльной стороны (с классом `back`) всегда следует после контейнера лицевой стороны (с классом `face`):

```
<div data-value="0" data-position="2">
  <div class="face"></div>
  <div class="back"></div>
</div>
```

Поэтому тыльная сторона будет всегда находиться над лицевой. Чтобы скрыть тыльную сторону, когда карта переворачивается, и показать вместо нее лицевую, мы добавляем объявление `backface-visibility: hidden;`. При этом способе, когда карта лежит лицевой стороной от нас, мы не видим элементы на этой стороне (мы видим сторону, которая обращена к нам, а не от нас).

Добавляя 3D-преобразование `translateZ(0)`, мы применяем аппаратное ускорение на тех устройствах, которые его поддерживают, обеспечивая выполнение анимации на графическом, а не на центральном процессоре. Поэтому мы включили четыре объявления:

```
-webkit-transform: rotatey(180deg);
-webkit-transform: rotatey(180deg) translatez(0);
        transform: rotatey(180deg);
        transform: rotatey(180deg) translatez(0);
```

поскольку не все браузеры поддерживают 3D-преобразования. Если браузер не понимает строку CSS, он пропускает все объявление «свойство — значение». Поэтому первое объявление не имеет `translateZ()` и предназначается для тех браузеров, которые не понимают этой инструкции, а за ним следует объявление с инструкцией `translateZ()` для тех браузеров, которые ее понимают. В этом фрагменте используются объявления как с префиксом, так и без него. В третьей строке — объявление без префикса, нацеленное на браузеры, не поддерживающие 3D, будет понятно браузерам. Они поддерживают преобразования, но не 3D-преобразования.

В код также включено объявление `transform-style: preserve-3d`, поскольку нужно обеспечить, чтобы лицевая и тыльная стороны карты, ее дочерние элементы, были с картой в одном и том же 3D-пространстве.

Если в игре перейти с первого уровня на второй, а потом на третий, высота карт уменьшится. Поскольку объявление о переходе предусматривает его осуществление в исходном состоянии за 0,25 с, когда карты уменьшаются в размере, они делают это за 0,25 с.

Вообще-то смена высоты нежелательна. Переходы, касающиеся блочной модели, заставляют браузер заниматься переформатированием каждого кадра, что приводит к нежелательным переформатированиям и перерисовкам. На мобильных устройствах эта проблема приобретает особую остроту и усугубляется наличием большого количества DOM-узлов. Можно было бы решить эту проблему с помощью перехода `transform: scaleY(0.8)`, поскольку это сохранит ширину карт и уменьшаться будет только их высота, но это исказит темы образов и цифр. Мы оставим все как есть. Изменение высоты карт будет происходить максимум два раза за всю игру. Мы проявили осторожность и не стали использовать слишком много DOM-узлов. Переход с рекалькуляцией всех DOM-узлов при всей своей неоптимальности в данном сценарии работает довольно хорошо.

Лучшим решением могли бы стать анимация масштабирования и выключение класса на основе завершения анимации. Далее будет показано, что анимация намного эффективнее переходов. Анимация дает возможность изменять класс или добавить к окончанию анимации прослушиватели событий.

CSS3-анимация

Явная анимация, дополняя переходы и преобразования, предоставляет способ объявления повторяющихся эффектов оживления элементов с помощью ключевых кадров.

В простых переходах, когда известны стартовое и финишное значения и требуется только один анимационный проход, ваши потребности в анимации вполне могут быть удовлетворены свойствами переходов. Если требуется более точное управление промежуточными значениями анимации или анимация должна повторяться, можно воспользоваться свойствами анимации и ключевыми кадрами CSS3 из модуля animation.

В число свойств анимации входят:

- `animation-name` — задает имя, которое дается определению вашей анимации с использованием ключевых кадров. По умолчанию используется значение `none`, что соответствует отсутствию анимации. Поэтому при желании анимировать элемент нужно воспользоваться свойством `animation-name`;
- `animation-duration` — задает продолжительность одного цикла анимации в секундах или миллисекундах. Значение по умолчанию — `0s`, то есть отсутствие какой-либо заметной анимации. Иными словами, нужно использовать свойство `animation-duration`, имеющее значение больше 0 с;
- `animation-timing-function` — задает порядок хода анимации в течение одного цикла. Это свойство получает значения, аналогичные тем, которые получает свойство `transition-timing-function`. Хотя значения весьма скромны, свойство `animation-timing-function` можно «оживить» в определениях ваших ключевых кадров. По умолчанию используется значение `ease`;
- `animation-iteration-count` — задает количество циклов анимации, выраженное целым числом или значением `infinite` (бесконечность). По умолчанию используется значение в один цикл;
- `animation-direction` — задает чередование циклов прямого и обратного проигрывания (`alternate`) или обычный режим (`normal`);
- `animation-play-state` — определяет состояния анимации: `running` — запущена, `paused` — приостановлена. При приостановке анимации отображается текущее значение анимации в статике. Когда выполнение приостановленной анимации возобновляется, она стартует с текущего значения. По умолчанию используется значение `running`;
- `animation-delay` — определяет момент запуска анимации. Интересно то, что при отрицательном значении анимация начнется где-то с промежуточного этапа. Например, если при продолжительности анимации, равной 10 с, задать для свойства `animation-delay` значение `-4s`, анимация начнется сразу же с отметки 40 % первого анимационного цикла;
- `animation-fill-mode` — определяет, какие значения будут применены анимацией до ее начала и после ее окончания. Возможны четыре значения:
  - `backwards`. Как только анимация будет применена, на все время задержки ее запуска используются значения, определенные для позиции 0 % ее ключевого кадра;
  - `forwards`. Как только анимация будет завершена, значения, определенные в последнем ключевом кадре, сохранятся до тех пор, пока стиль анимации не будет удален из любого селектора, нацеленного на данный узел;

- both. Реализуются оба значения, и forwards и backwards, как только анимация будет завершена и перед тем как она будет запущена соответственно;
- none. Значение по умолчанию, не приводит ни к каким действиям или же удаляет любые действия, связанные с применением значений forwards или backwards;

❍ animation — краткая форма записи свойств анимации, где через пробелы перечисляются значения свойств animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction и animation-fill-mode. Если объявляются сразу несколько анимаций, группы свойств, выраженных в краткой форме и относящихся к каждому отдельному имени анимации, отделяются друг от друга запятыми.

Ключевые кадры

Ключевые кадры задают состояние ваших элементов на различных стадиях анимации. Они определяются с помощью правила @keyframes. Это правило состоит из ключевого слова @keyframes, за которым следуют идентификатор, дающий имя анимации, и набор стилевых правил, заключенный в фигурные скобки ({}). Идентификатор[1] (имя) анимации задается вами. В кавычки его брать не нужно. Это имя используется в качестве значения свойства animation-name.

Селектор ключевого кадра для правила ключевого кадра состоит из списка значений, в котором в качестве разделителя применяется запятая. Можно использовать либо процентные значения, либо ключевые слова from и to. Например:

```
@keyframes crazyText {
  from {
    font-size: 1em;
  }
  to {
    font-size: 2em;
  }
}
```

В WebKit-браузерах это будет выглядеть следующим образом[2]:

```
@-webkit-keyframes crazyText {
  from {
```

[1] В CSS идентификаторы (IDENT) включают имена элементов, классы, идентификаторы элементов (ID) и имена ключевых кадров анимации и могут содержать только символы [a–zA–Z0–9] и символы ISO 10646 U+00A0 и выше плюс дефис (-) и символ подчеркивания (_). Они не могут начинаться с цифры, двух дефисов или дефиса, за которым следует цифра. Идентификаторы в кавычки не берут.

[2] Анимация поддерживается в IE10, BlackBerry, Android, Chrome for Mobile, iOS и во всех современных мобильных браузерах. Она ожидаемо не поддерживается в Opera Mini. Префикс все еще нужен, но только в WebKit-браузерах и в Boot2Gecko. Firefox избавился от префикса в Firefox 16, Opera — в Opera 12.1, а Opera Mobile никогда не поддерживал версию с префиксом. IE начал поддерживать анимацию с IE10 без использования префикса.

```
    font-size: 1em;
  }
  to {
    font-size: 2em;
  }
}
```

Ключевое слово `from` является эквивалентом значения 0%, а ключевое слово `to` — значения 100%. Учтите, что здесь должен использоваться спецификатор, выраженный в процентах. Поэтому 0 в качестве селектора ключевого кадра применять нельзя.

Если нужно определить более двух точек, то есть не только старт и финиш анимации, более детального управления анимацией можно добиться за счет определения промежуточных ключевых кадров, воспользовавшись процентным отношением. Например:

```
@-prefix-keyframes rainbow {
  0% {
    background-color: red;
  }
  20% {
    background-color: orange;
  }
  40% {
    background-color: yellow;
  }
  60% {
    background-color: green;
  }
  80% {
    background-color: blue;
  }
  100% {
    background-color: purple;
  }
}
```

Селекторы ключевых кадров используются для указания процентного отношения, представляемого ключевым кадром, к общей продолжительности анимации. Можно провести аналогию между 20, 40, 60% и т. д. и псевдоклассами продолжительности перехода (конечно, процентные показатели таковыми не являются, но их можно представлять в этом качестве). Ключевой кадр указывается с помощью стилевых правил (блока кода, состоящего из значений свойств), объявленных в селекторе ключевого кадра.

Процентный показатель или селектор ключевого кадра определяет место ключевого кадра в анимации. Чтобы с кадрами было проще разобраться, я рекомендую размещать селекторы ключевых кадров по возрастанию от 0 до 100%, хотя это делать и не обязательно.

Стилевые блоки для селекторов ключевых кадров состоят их свойств и значений. Свойства, поддающиеся анимации, перечислены в разделе «CSS3-преобразования». Свойства, которые невозможно анимировать, в этих правилах игнорируются.

Чтобы определить набор кадров, все значения в селекторах сортируются в возрастающем по времени порядке. Селекторы ключевых кадров не каскадируются, поэтому анимация никогда не извлекает ключевые кадры более чем из одного селектора ключевых кадров. При наличии каких-либо дубликатов для предоставления информации о ключевом кадре для этого времени используется последний определенный в правиле `@keyframes` селектор ключевого кадра.

Механизм анимации станет равномерно интерполировать стиль между селекторами ключевых кадров. В примерах, показанных в онлайн-ресурсах к главе, где анимация занимает 10 с при линейной функции развития, на отметке 5 с для анимации `crazyText` применяется стиль `font-size: 1.5em`, а элемент с анимацией rainbow имеет желтовато-зеленый фон.

Применение анимации

Возможно, мы определили анимацию, но не прикрепили ее ни к одному из элементов. После определения анимации с помощью объявления `@keyframes` мы применяем ее, используя для этого как минимум два обязательных свойства: `animation-name` и `animation-duration`. Все остальные имеющие к этому отношение свойства являются дополнительными:

```
div {
  animation-name: crazyText;
  animation-duration: 1s;
  animation-iteration-count: 20;
  animation-direction: alternate;
  animation-delay: 5s;
  animation-fill-mode: both;
}
```

Предыдущее правило прикрепляет анимацию crazyText, устанавливает ее продолжительность равной 1 с на каждый проход и общее количество ее выполнений равным 20, задает ее проигрывание в обратном порядке при каждом втором проходе и задержку, равную 5 с, перед началом первого прохода.

Установка для свойства `animation-fill-mode` значения `both` показывает, что при первом отображении контейнера `<div>` на странице свойству `font-size` будет задано значение `1em`, что соответствует состоянию ключевого кадра `0%` или `from` до 5-секундной задержки перед первым проходом.

Затем значение `font-size` будет удваиваться за одну секунду, потом вновь уменьшаться до `1em` за следующую секунду благодаря объявлению `animation-direction: alternate`. Если бы было сделано объявление `animation-direction: normal` или же свойство `animation-direction` вообще было опущено, значение свойства `font-size` удваивалось бы за одну секунду, а затем моментально возвращалось назад к `1em`, перед тем как удвоиться опять в течение секунды в ходе второго и всех последующих проходов.

Когда анимация завершит 20 проходов, через 25 с после применения анимации (20 односекундных проходов плюс пятисекундная задержка), значение свойства `font-size` останется таким же, каким было в последнем ключевом кадре, благодаря объявлению `animation-fill-mode: both`. При использовании объявления `animation-direction: alternate` значение свойства `font-size` будет равно `1em`. Если бы свойство `animation-direction` было опущено или же использовано объявление `animation-direction: normal`, у последнего ключевого кадра было бы значение `2 em`. Для контейнера `<div>` это значение размера шрифта останется «навсегда», пока объявление `font-size`, нацеленное на данный узел, его не заменит.

С указанием отступов и добавлением анимации `rainbow` это можно было бы записать в следующей краткой форме:

```
div {
  padding: 20px;
  animation:
    crazyText 1s 205s alternate both,
    rainbow 4s infinite alternate;
}
```

Анимация для получения эффекта отскакивающего мяча

Для анимации действует такое же правило, как и для переходов: анимировано может быть любое свойство, имеющее явно обнаруживаемое среднее состояние. Из этого правила существует два исключения: свойства `visibilility` и `animation-timing-function`. У них нет средних состояний, но оба они могут быть добавлены к стилевому блоку ключевого кадра.

Если в ключевой кадр входит свойство `animation-timing-function`, то с данной точки анимация переключится с установки по умолчанию или с текущей функции, задающей порядок хода анимации, на новое объявление этой функции. Хотя этот прием используется довольно редко, он все же может пригодиться. Например, если создается эффект отскакивающего мяча, гравитация задает ускорение (или `ease-in`) при его падении и замедление при отскоке (или `ease-out`):

```
@keyframes bouncing {
  0% {
    bottom: 200px;
    left: 0;
    animation-timing-function: ease-in;
  }
  40%, 70%, 90%{
    animation-timing-function: ease-out;
    bottom: 0;
  }
  55% {
    bottom: 50px;
    animation-timing-function: ease-in;
  }
  80% {
    bottom: 25px;
```

```
    animation-timing-function: ease-in;
  }
  95% {
    bottom: 10px;
    animation-timing-function: ease-in;
  }
  100% {
    left: 110px;
    bottom: 0;
    animation-timing-function: ease-out;
  }
}
```

В этом примере кода (который имеется в онлайн-ресурсах к главе) есть ряд примечательных моментов. У нас есть три ключевых кадра с абсолютно одинаковыми значениями — ключевые кадры 40, 70 и 90%, поэтому все они помещены в одной строке. Селекторы ключевых кадров отделены друг от друга запятыми, как это делается в обычных селекторах CSS.

Здесь анимируется не одно, а несколько свойств, но все значения в блоке каждого ключевого кадра не объявляются. Движение влево-вправо происходит плавно, поэтому нам нужно объявить соответствующее свойство только дважды: в блоках 0 и 100%. Перемещение мяча вверх и вниз производится множество раз с применением более детального управления, поэтому, в отличие от значения свойства `left`, значения для свойства `bottom` мы объявляем в каждом ключевом кадре.

Мы также *оживили* или изменили значение свойства `animation-timing-function`, чтобы придать отскоку более плавный и естественный вид. Без этого анимация отскакивающего мяча будет слишком дерганой. Свойство `animation-timing-function` является единственным свойством анимации, которое можно включать в объявления ключевых кадров[1].

Анимированные спрайты

Когда я думаю об HTML и анимации, то имею в виду анимацию отдельного узла при перемещении его на новое место или еще что-нибудь ужасно скучное. Одной из особенностей CSS-анимации является анимация символов, подобная анимации прыгающего лемминга или, как в данном случае, лемминга, прыгающего в воду с обрыва.

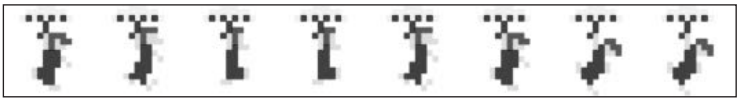


Рис. 10.7. Спрайт лемминга для символьной анимации

[1] Чтобы узнать о программировании анимации больше, можно в онлайн-ресурсах к главе воспользоваться ссылками на панель обучения анимации, на которой каждое свойство анимации демонстрируется в действии.

Для создания символьной анимации со спрайтом используется объявление значения `animation-timing-function: step(x, start)`. Значения `step` не обеспечивают плавный переход от одного ключевого кадра к другому. При его задании анимация разбивается на объявленное количество шагов и осуществляется моментальный переход от одного шага к другому. Чтобы анимировать спрайт, показанный на рис. 10.7, мы перемещаем фоновое изображение:

```
.lemming {
  height: 32px;
  width: 32px;
  background-image:url(lemming.gif);
  background-repeat: no-repeat;
  -webkit-animation: lemming 1s steps(8,end) alternate infinite;
  animation: lemming 1s steps(8,end) alternate infinite;
}
@-webkit-keyframes lemming {
  from {
    background-position: 0 0;
  }
  to {
    background-position: -256px 0;
  }
}
@keyframes lemming {
  from {
    background-position: 0 0;
  }
  to {
    background-position: -256px 0;
  }
}
```

Заметьте, что спрайт имеет ширину 256 пикселов, поэтому при объявлении `background-position: 256px 0;` фонового изображения видно не будет. Свойство `animation-timing-function` позволяет объявлять значение `steps(x, start)`, указывающее на то, что изменения в значении свойства начинаются со старта каждого шага, и значение `steps(x, end)`, указывающее на то, что изменения в свойстве осуществляются в конце каждого шага.

Давайте воспользуемся анимацией, состоящей из пяти шагов. Если объявить `steps(5, start)`, переход будет осуществляться в начале шага и у нас получится пять шагов на отметках 20, 40, 60, 80 и 100%, так что 0% показывать нет смысла, потому что изменения свойства от 0 до 20% происходят в начале шага и зрители видят состояние, соответствующее отметке 20%, все время от 0 до 20%. Если объявить `steps(5, end)`, переход к следующему шагу произойдет в конце интервала, то есть будут показаны состояния на отметках 0, 20, 40, 60 и 80%, потому что последний шаг находится за пределами спрайта и состояние на отметке 100% никогда не показывается.

Создавая спрайт с фазами движения, анимируя свойство `background-position` и перемещаясь по этому спрайту с использованием шагов, можно создавать ани-

мацию движения. Дополнительные примеры анимации с использованием спрайтов можно найти в онлайн-ресурсах к главе.

Анимация в CubeeDoo

В CubeeDoo используется очень простая анимация. Когда пара карт совпадает, анимируется исчезновение карт. А когда счет становится рекордным, он на больших экранах выделяется анимацией, подобной вспышке, в области высоких оценок:

```
#board > div.matched {
  -webkit-animation: fade 250ms both;
  animation: fade 250ms both;
}
#board > div.matched:nth-of-type(1) {
  -webkit-animation-delay: 250ms;
  animation-delay: 250ms;
}

@-webkit-keyframes fade {
  0% {
    -webkit-transform: scale(1.0) rotatey(180deg) rotate(0)
    translatez(0);
  }
  100% {
    -webkit-transform: scale(0) rotatey(180deg) rotate(720deg)
    translatez(0);
  }
}
@keyframes fade {
  0% {
    transform: scale(1.0) rotatey(180deg) rotate(0) translatez(0);
  }
  100% {
    transform: scale(0) rotatey(180deg) rotate(720deg) translatez(0);
  }
}
#highscores li.current {
  -webkit-animation:
    winner 500ms linear 8 alternate forwards;
a  nimation:
    winner 500ms linear 8 alternate forwards;
}
@-webkit-keyframes winner {
  0% {background-color: hsla(74, 64%, 59%,1);}
  100%{background-color: hsla(74, 64%, 59%,0)}
}
@keyframes winner {
  0% {background-color: hsla(74, 64%, 59%,1);}
  100%{background-color: hsla(74, 64%, 59%,0)}
}
```

Когда к карте добавляется класс `matched` (совпала), к ней прикрепляется анимация постепенного исчезновения — `fade`. Класс `matched` добавляется к двум картам, имеющим класс `flipped` (перевернута), если две перевернутые карты совпадают. В противном случае, если две перевернутые карты не совпадают, класс `flipped` удаляется и происходит переход, рассмотренный в предыдущих разделах, посвященных переходам. Для того чтобы анимация выполнялась графическим, а не центральным процессором, если это позволяют браузер и устройство, мы включили в код 3D-преобразование.

Название анимации `fade` (постепенное исчезновение) неправильно отражает суть происходящего. На самом деле за 250 мс элемент поворачивается и сжимается. В конце анимации кодом JavaScript удаляются оба класса, и `flipped` и `matched`, и путем установки пары «атрибут — значение» `data-value="0"` карта сбрасывается и прячется.

Когда игра заканчивается, обновляется список наивысших достижений. Если текущий наивысший счет входит в первую пятерку результатов, к счету добавляется класс `current` и этот счет записывается на страницу. Управление добавлением и удалением имен классов осуществляется с помощью кода JavaScript, а создание и выполнение анимации, основанной на принадлежности элемента к тому или иному классу, — исключительно средствами CSS. В случае с наивысшим счетом выполняется анимация `winner`, которая показывает пульсирующий, то есть растворяющийся и восстанавливающийся восемь раз зеленый фоновый цвет: четыре раза от полностью непрозрачного к полностью прозрачному и четыре раза от полностью прозрачного к полностью непрозрачному. Управление этой анимацией осуществляется путем объявления `animation-iterations: 8;` с `animation-direction: alternate;` в краткой записи свойства `animation`. Текущий наивысший счет так и будет виден на зеленом фоне, пока результаты перерисовываются. Мы не помещаем эту анимацию в 3D-пространство: поскольку здесь происходит очень простая перерисовка без переформатирования, она должна неплохо выполняться и при исходных настройках.

Переходы, анимация и производительность

CSS-анимация позволяет описывать декларативные правила для анимации, заменяя тем самым большой объем трудоемкого в обслуживании кода JavaScript.

Браузеры оптимизированы для работы CSS-анимации. Как таковая CSS-анимация производительнее JavaScript-анимации. По возможности нужно всегда использовать для анимации CSS, а не JavaScript. За счет использования CSS браузеру разрешается оптимизировать создание промежуточных кадров и пропуск кадров, что дает ему возможность оптимизировать производительность.

Хотя CSS-анимация производительнее JavaScript-анимации, у нее есть и недостатки. Как и JavaScript-анимация, CSS-анимация приводит к повышенному расходу заряда аккумулятора. Но CSS не нагружает центральный процессор так сильно, как JavaScript, поэтому работа системы будет более плавной.

В потоке пользовательского интерфейса у CSS самая низкая приоритетность. Это означает, что при загрузке огромного файла JavaScript, занимающей 8 с, в течение этого времени анимация на странице не начнется. Возможно, это не такая уж большая проблема, но здесь есть одна неприятная особенность: хотя анимация еще не началась, задержка ее старта, заданная в свойстве `animation-delay`, уже отсчитывается. И если есть 15 анимаций, каждая из которых стартует через секунду после старта предыдущей благодаря использованию задержек старта анимации на 0, 1, 2 с и т. д., первые восемь анимаций будут стартовать и выводиться на экран вместе, когда наконец завершится загрузка страницы, а семь следующих анимаций будут запущены в соответствии с ранее определенным графиком.

Для решения этой проблемы к документу можно добавить класс `loaded` (загруженный) и поставить анимацию элемента в зависимость от того, является он или нет потомком элемента с классом `loaded`.

Также нужно отметить, что анимация некоторых свойств осуществляется с более высокой производительностью по сравнению с анимацией других свойств. Если изменение разметки страницы влечет за собой переформатирование текста, анимация не будет выполняться с такой же производительностью, как при простой перерисовке объекта. Например, анимация, связанная с увеличением размера шрифта, которую мы проделывали ранее, — не что иное, как глупость! Такая анимация вызывает повторяющееся переформатирование страницы: как только меняется размер шрифта, меняется и размер элемента. Как только меняется размер элемента, до перерисовки происходит переформатирование документа. При простом масштабировании элемента средствами CSS-преобразований и анимации перехода браузеры перерисуют только тот элемент, который подвергся анимации, что выполняется гораздо быстрее.

Нужно помнить, что переформатирование является весьма затратным действием и занимает много времени. Чтобы достичь плавности, нужно, чтобы кадры анимации полностью рисовались за время не более 16,67 мс.

Существует также API-интерфейс анимации, позволяющий перехватывать события из CSS-анимации. С каждым проходом анимации возникают такие события, как `animationStart`, `animationEnd` и `animationIteration`. Мы не будем останавливаться на касающихся этих событий подробностях, но ссылки на соответствующую информацию можно найти в онлайн-ресурсах к главе.

Мы еще не завершили наш экскурс в CSS, и нам есть еще что обсудить. Это и будет сделано в главе 11.

11 Использование свойств CSS в адаптивном веб-дизайне

Содержимое должно разрабатываться с учетом функционирования на любом устройстве, потому что оно предназначено для повсеместного просмотра на всех видах устройств. Сайты, создаваемые сегодня с прицелом на просмотр на настольных системах, смартфонах и планшетных устройствах, могут отображаться как на экранах 52-дюймовых телевизоров, так и на жидкокристаллических экранах портативных устройств размером 3 × 5 дюймов. Если изначально закладывать в построение сайта гибкую основу, его можно будет вполне элегантно расширять или сжимать независимо от оборудования, загружающего этот сайт.

Чтобы сайт мог адаптироваться под любой размер экрана, нужно придать ему как можно больше гибкости. На пути к достижению этой цели 90 % успеха может принести переход при выражениях показателей ширины и размера шрифта от пикселов к процентам и rem. Добавление ряда медиазапросов доведет показатель до 95 %.

Помимо медиазапросов и тех средств CSS, которые уже были рассмотрены, существует ряд CSS-свойств, которые пригодятся в разработке адаптивных сайтов и доведут отметку успешного решения поставленной задачи до 99 %. А почему до 99, а не до 100 %? Совершенству нет предела, можно стремиться сделать сайт более адаптивным, доступным, привлекательным, быстрым и т. д., но, как говорится, лучшее враг хорошего и путь к нему может стать слишком длинным.

Медиазапросы, контрольные точки и резиновая разметка

Хотя об этом уже говорилось, но стоит повторить: не создавайте разметку под конкретные размеры телефонных экранов. Лучше постепенно расширяйте (или

сжимайте) свой сайт в браузере. Как только разметка начинает терять оптимальный вид, настает именно тот момент, который требует изменения дизайна для следующего набора устройств. Может понадобиться восемь уровней разметки для совсем крошечных (tiny), маленьких (xx-small), небольших (small), средних (medium), больших (large), весьма больших (x-large), очень больших (xx-large) и громадных (huge) экранов, или же может быть создана единая разметка, работающая на всех устройствах. Пока не просмотрите разметку на экранах разных размеров, вы ничего не поймете, поэтому нужно устроить такой просмотр, чтобы убедиться, что разметка хорошо работает на экранах любых размеров.

Когда изменяются размеры браузера и принимаются решения о необходимости новой разметки, ту ширину, при которой должна меняться разметка, называют контрольной точкой. Не стоит выбирать в качестве контрольных точек значения 320, 480, 640 и 960 только потому, что все так делают. Лучше сделать разумный выбор, подходящий именно для вашего сайта.

Когда потребность в контрольной точке станет очевидной, используйте медиазапросы, чтобы нацелить конкретную разметку на определенный диапазон размеров окна просмотра. Не следует также ограничиваться одной контрольной точкой. Вносить изменения в разметку заголовков, нижних колонтитулов, панели навигации и основного содержимого нужно в нескольких контрольных точках, имеющих наибольший смысл. Здесь нет правильных и неправильных решений, оценивать нужно по принципу: лучше или хуже выглядит результат.

После определения контрольных точек, подходящих для вашего дизайна или выбранных с точки зрения удобства пользования приложениями, можно нацелиться на изменение разметки и на особенности, выделяемые медиазапросами. Можно также ориентироваться на дисплеи с высокой плотностью точек на дюйм с более крупными изображениями, помня о том, что большие изображения требуют файлов большего размера. Чтобы отправлять изображения с высоким разрешением только для дисплеев с высоким показателем DPI при высокой пропускной способности канала, можно в медиазапросах использовать JavaScript. Возможно, настанет время, когда мы сможем в медиазапросах определять соответствующую ширину полосы пропускания, но пока такой возможности нет.

Медиазапросы уже рассматривались в главах 2 и 7, поэтому их синтаксис должен быть вам понятен.

Использование нескольких столбцов

CSS-свойство `columns` позволяет создавать несколько столбцов для разметки текста, как в газете. Свойство `columns` является краткой формой записи свойств `column-count` и `column-width`. Свойство `column-width` задает оптимальную ширину столбца, как будто объявляется несуществующее свойство `min-column-width`. Свойство `column-count` задает максимальное количество столбцов, как будто объявляется также несуществующее свойство `max-column-count`.

Свойство column-count имеет преимущество. Браузер настраивает ширину столбца относительно значения, заданного свойством column-width, обеспечивая то количество столбцов, которое указано в виде целочисленного значения свойства column-count, пока каждый столбец имеет по крайней мере ширину, указанную в качестве значения свойства column-width. Столбцы позволяют создавать масштабируемые варианты дизайна с возможностью умещать информацию на экранах разной ширины.

Промежуток между столбцами устанавливается с помощью специального свойства column-gap, а разделительная линия между столбцами включается путем объявления линии в свойстве column-rule. Значение CSS-свойства column-gap задается в единицах длины, а по умолчанию используется значение в виде ключевого слова normal, которое в большинстве браузеров будет соответствовать значению 1 em.

Использование столбцов делает широкие области содержимого более разборчивыми. А если окно просмотра узкое, количество столбцов будет сокращено. Если установлен следующий стиль, на 24-дюймовом мониторе можно увидеть шесть столбцов, а на HTC One в книжной ориентации — только один:

```
columns: 240px 6;
```

Предыдущая строка читается так: «Разделить содержимое элемента максимум на шесть столбцов, обеспечив ширину столбца — не менее 240 пикселов». У iPhone ширина экрана — 320 пикселов, поэтому при книжной ориентации два столбца на нем не уместятся, но при отсутствии отступов и промежутков они могут уместиться при альбомной ориентации. На дисплее размером 1920 × 1080 могут уместиться восемь столбцов, если не делать промежутков между ними. Но браузер не станет выводить более шести столбцов, даже если на отведенном месте может поместиться больше столбцов.

Как и border, свойство задания линии, разделяющей столбцы, column-rule, также является краткой формой записи свойств column-rule-width, column-rule-style и column-rule-color и получает такие же значения, как и краткая форма записи border. Пока значение свойства ширины разделительной линии column-rule-width меньше значения column-gap, эта линия будет выводиться на экран. Свойства столбцов, за исключением column-rule-style, поддаются анимации:

```
p {
  margin: 0 0 1em;
}

div {
  padding: 1em;
  margin: 1em;
  border: 2px solid #ccc;
  columns: 240px 6;
  column-gap: 2em;
  column-rule: 2px dashed #ccc;
}
```

Этот фрагмент кода[1] при выполнении на большом устройстве приведет к созданию разметки в несколько столбцов, а на устройстве или в родительском элементе, имеющем ширину менее 480 пикселов + 4 ems, будет, как показано на рис. 11.1 и в примерах онлайн-ресурсов к главе, выведен один столбец.

Рис. 11.1. Столбцы, видимые на узком и широком экранах

Еще одним интересным моментом в предыдущем коде является установка полей для абзацев: одной из причин того, что в прошлом разработчики не спешили использовать столбцы, являлись иногда возникающие пробелы в верхней и нижней частях столбца. Неровная нижняя часть особых нареканий не вызывала, но нужны были гарантии, что пробела не будет в верхней части столбца. Если новый столбец начинался внутри узла с полем или отступом в верхней части, то и в верхней части этого столбца появлялся пробел. Чтобы гарантировать отсутствие пробелов в верхней части столбцов, нужно установить отступы и поля в нижней части абзацев и других дочерних элементов ваших столбцов.

Когда свойство `column-span` получает значение `all`, оно позволяет элементам распространяться по всем столбцам. Если у элемента имеется настройка `column-span: all;`, содержимое вокруг него будет разделено между всеми столбцами поровну, сам элемент станет проходить через весь родительский элемент, а последующее содержимое будет опять отображено в виде столбцов.

По умолчанию все столбцы будут иметь примерно одинаковую высоту и содержимое будет поровну разделено между ними. В результате установки высоты родительского элемента и придания свойству `column-fill` (заполнение столбцов) значения `fill` (а не `balance`, которое используется по умолчанию) столбцы будут заполняться последовательно. В отличие от прочих свойств столбцов, свойства `column-span` и `column-fill` не пользуются всеобщей поддержкой.

Чтобы установить точную ширину столбца, в расчет нужно брать свойства `width`, `column-width`, `column-gap` и `column-rule-width`.

---

[1] Различные свойства столбцов поддерживаются во всех мобильных браузерах, включая Opera Mini, начинают поддерживаться в IE10 и должны иметь префиксы для WebKit и Firefox.

Для эффективного использования столбцов в адаптивной разметке нужно убедиться в том, что у их родительского элемента резиновая ширина. Объявляйте максимальное количество столбцов, которое нужно показывать на широком экране, и минимальную ширину, которую нужно видеть на любом экране, и содержимое станет адаптивным.

Для этой работы медиазапросы не нужны, однако если сужать и расширять окно браузера, доводя его до самой малой и самой большой ширины, можно оценить, имеет ли смысл использовать контрольные точки медиазапросов.

Изображения для границ

Свойство `border-image` позволяет использовать отдельное изображение для создания декоративных границ любого элемента независимо от его размера или соотношения сторон. С помощью отдельного файла изображения очень маленького размера или даже градиентов можно, помимо простых скругленных углов, создавать декоративные границы элементов.

При использовании свойства `border-image` изображение фактически делится на девять частей, при этом углы этого изображения помещаются в углы вашего элемента, принимая ширину его левой и правой границы и высоту верхней и нижней границы, а неугловые компоненты либо повторяются, либо растягиваются, чтобы охватить весь элемент. Можно взять одно относительно небольшое изображение и охватить им всю площадь небольшой кнопки или целой страницы.

На рис. 11.2 показаны три изображения для границ, а также то, как их можно было разделить в графическом редакторе до начала поддержки браузерами краткой записи свойства `border-image` и как сейчас браузеру задается порядок фактического разделения изображения.

Рис. 11.2. Небольшие изображения, которые делятся на части для создания эффектов границы и фона на элементах различных размеров

У собственных приложений iOS имеются кнопки, похожие на небольшую кнопку, показанную в центре рис. 11.2. Ранее мы создавали такие кнопки с помощью градиентов. Кнопку подобного вида можно было бы создать с помощью метода раздвижной двери или ряда других приемов, использовавшихся для создания кнопок в последнем десятилетии. Или же можно с помощью CSS-свойства `border-image` использовать для каждой кнопки одно небольшое изображение, не обращая внимания на то, каким будет размер кнопки, 10 × 10 или 200 × 300 пикселов.

В качестве изображений для границ элементов, показанных на рис. 11.3, использовались изображения, приведенные на рис. 11.2, при этом, чтобы охватить весь элемент, сохранялись углы и повторялась (в случае с маркой) или растягивалась средняя часть изображения для границы.

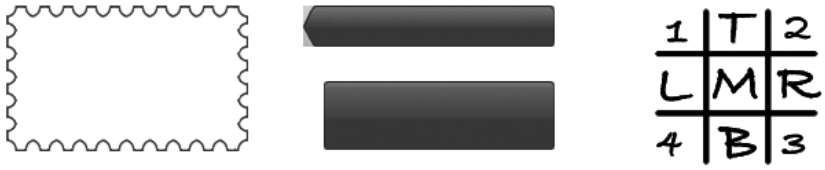


Рис. 11.3. Элементы с настроенным для них свойством border-image, с четырьмя угловыми частями, расставленными по углам, а также с верхней — top (T), правой — right (R), нижней — bottom (B) и левой — left (L) частями, которые были повторены или растянуты

В примере с маркой средние части (T, R, B и L) были повторены, чтобы получились зубцы марки. Чтобы обеспечить целостность изображения, ширину и высоту частей следует принимать кратными ширине (T и B) и высоте (R и L). Помимо повторений вверху, внизу и по бокам, мы создали четыре угла (перечисленные как 1, 2, 3 и 4), получив в результате эффект, похожий на почтовую марку. Свойство `border-image` является краткой формой записи, используемой для объявления свойств `border-image-source`, `border-image-slice`, `border-image-width`, `border-image-outset` и `border-image-repeat`.

Синтаксис краткой записи имеет следующий вид:

```
-prefix-border-image: <source>
  <slice {1,4}> / <width {1,4}> <outset> <repeat{1,2}>;
```

Браузеры, поддерживающие изображения для границ, поддерживают только краткую форму записи свойства `border-image`, а не отдельные свойства, значения которых составляют краткую запись. Мы рассмотрим все эти свойства, но я рекомендую вместо развернутой формы записи использовать краткую.

Следует заметить, что используемый в настоящее время синтаксис со времени начала реализации подвергался неоднократным изменениям. Поэтому, читая публикации в блогах на данную тему, нужно убедиться, что автор использует синтаксис, действующий в настоящее время.

Если границ нет, изображения для границ работать не будут. Поэтому сначала нужно объявить границы элементов. Из материалов главы 9 известно, что единственным обязательным свойством является `border-style`:

```
.button {
  border: solid;
}
.stamp {
  border: solid;
}
.arrow {
  border: solid;
}
```

При отсутствии объявления `border-style` со значением, отличным от `none` или `hidden`, попытка отобразить изображение для границы будет безуспешной.

border-image-source

Значением свойства источника изображения, которое нужно использовать для границы, `border-image-source`, являются URL-адрес, градиент или URI-данные этого изображения. В примерах, показанных на рис. 11.3, несмотря на то что развернутая форма свойства еще не получила всеобщей поддержки, все выглядит так, будто мы использовали объявление `border-image-source: url(stamp.gif)`, однако три наших объявления свойства `border-image` в краткой форме начинаются со следующего кода:

```
.button {
  border: solid;
  border-image: url(button_bi.png) ...
}
.stamp {
  border: solid;
  border-image: url(stamp.png) ...
}
.arrow {
  border: solid;
  border-image: url(arrow.png) ...
}
```

В качестве изображений для границ можно использовать те же типы изображений, которые используются при создании фона элемента: градиенты, base-64-, GIF-, JPEG-, PNG- и даже SVG-изображения.

border-image-slice

Свойство `border-image-slice` определяет от одного до четырех значений длины, устанавливающих дистанцию от каждой из сторон изображения, создающую область, которая будет использоваться для разрезания или нарезания на части изображения для границы (см. рис. 11.2). Свойство `border-image-slice` определяет также среднюю часть `border-image`, помеченную на рис. 11.3 буквой M, которая отбрасывается или заполняет среднюю часть элемента.

Значения свойства `border-image-slice` представляют собой внутренние смещения соответственно от верхнего, правого, нижнего и левого краев изображения (правило TRouBLe). Определяются четыре воображаемые линии, которые браузер затем использует для деления изображения, предназначенного для границы, на девять областей: четыре угла, четыре края и середину (см. рис. 11.3). Четыре угловые области помещаются в соответствующие углы, проходя масштабирование, чтобы поместиться в пространство, предназначаемое для них значениями свойств ширины границы. Четыре стороны получают путем растягивания изображения или его многократного повтора или же используют комбинацию этих двух приемов (подгонку) в зависимости от значений других свойств `border-image`.

Кроме четырех значений, беспрефиксная версия свойства `border-image-slice` получает еще одно необязательное значение `fill` для сохранения средней части

изображения, предназначенного для границы. Если ключевое слово `fill` отсутствует, средняя часть изображения, предназначенного для границы, отбрасывается. А если оно присутствует, средняя составляющая растягивается, повторяется или подгоняется при повторении в зависимости от значения рассматриваемого далее свойства `border-image-repeat`.

В наших примерах изображение разрезается на части линиями, отстоящими на 5 пикселов от каждой из сторон для кнопки, на 9 пикселов — для почтовой марки, на 0 пикселов от верхней и нижней сторон, на 5 пикселов от левой стороны и на 20 пикселов от правой стороны — для стрелки. Среднюю часть нужно показать для стрелки и кнопки, но не для марки. В развернутой форме мы бы написали соответственно `border-image-slice: 5px fill;`, `border-image-slice: 9px;` и `border-image slice: 0 5px 0 20px fill;`. Но вместо этого мы включили в краткую форму записи свойства значения без указания единиц длины, а также с указанием ключевого слова `fill` для кнопки и стрелки:

```
.button {
  border: solid;
  border-image: url(button_bi.png) 5 fill...
}
.stamp {
  border: solid;
  border-image: url(stamp.png) 9 ...
}
.arrow {
  border: solid;
  border-image: url(arrow.png) 0 5 0 20 fill...
}
```

Следует отметить отсутствие единиц длины. Если значение для разрезания на части задается в виде указания длины и значение будет интерпретировано в пикселах, единицу измерения длины нужно опустить. Если используется процентное отношение, указывается знак процента.

border-image-width

С помощью свойства `border-image-width` устанавливается ширина границы элемента. Если свойство `border image-width` объявляется как часть краткой формы записи свойства `border-image`, его значение получает преимущество над значением свойства `border-width`. Если будут опущены значение, а также объявление значения свойства `border-width`, для ширины границ большинством браузеров будет установлено значение 3 пиксела, являющееся средним, исходным значением свойства `border-width`.

Несмотря на причуды, связанные с реализацией значения `auto`, зачастую свойство `border-width` рекомендуется включать отдельно или в составе краткой записи свойства `border`, а не в составе краткой записи свойства `border-image`:

```
.button {
  border: solid 5px;
```

```
  border-image: url(button_bi.png) 5 fill...
}
.stamp {
  border: solid 9px;
  border-image: url(stamp.png) 9 / 9px ...
}
.arrow {
  border: solid;
  border-width: 05px 020px;
  border-image: url(arrow.png) 05020 fill / 05px 020px...
}
```

Четыре угла, помеченные на рис. 11.3 цифрами 1, 2, 3 и 4, получат ширину левой и правой границ и высоту верхней и нижней границ. Если значение свойства `border-image-width` будет таким же, как и значение свойства `border-image-slice`, изображение границы приобретет наилучший вид и будет выведено на экран без искажений. Но их одинаковые значения не являются обязательным требованием. Чтобы выделенная для границы часть изображения могла вписаться в другую ширину, заданную значением свойства `border-image-width`, она будет растянута (или сжата).

Между значениями свойств `border-image-slice` и `border-image-width` мы поставили слеш. Для определения ширины границ мы в беспрефиксной версии полагаемся на `border-width` — положительное значение, заданное без использования процентного отношения.

Но следует помнить об особенностях блочной модели! Свойство `border-width` является частью блочной модели и будет оказывать влияние на элементы. По мере увеличения значения свойства `border-image-width` ваш элемент будет увеличиваться в размерах, пока это не будет пресечено с помощью объявления `box-sizing: border-box;`.

border-image-outset

Значение свойства `border-image-outset` определяет величину, на которую область изображения границы будет выходить за границы блока со всех четырех сторон. По умолчанию используется значение 0.

Поскольку средняя часть представлена прозрачным изображением в формате PNG и мы ее ничем не заполнили, при добавлении свойства фонового цвета `background-color` этот цвет будет просматриваться сквозь среднюю часть и прозрачную часть границы. Решить эту связанную с границей проблему можно двумя способами: с помощью объявления `background-clip: padding-box;` или путем помещения изображения границы за пределы блока с помощью свойства `border-image-outset`. Первый способ не изменяет размеры блока. Второй способ приводит к их изменению, похожему на изменение, получаемое при использовании свойства `box-shadow`, увеличивая видимый размер элемента, но не оказывая влияния на блочную модель:

```
.stamp {
  border: solid 9px;
  background-color: #dedeef;
  border-image: url(stamp.png) 9 / 9px / 12px ...
}
```

border-image-repeat

Продолжая игру с построчным анализом кода, можно сказать, что на данный момент кнопка и стрелка выглядят неплохо, а марка — хуже. По умолчанию верхняя, правая, нижняя и левая части были растянуты, при этом отдельно взятая часть растянута на всю ширину или высоту элемента. Стрелку и кнопку такой прием не портит, собственно, для них именно это и требовалось, а что касается марки, все смотрится хуже некуда. Нам нужно, чтобы те части, из которых получаются стороны марки, не растягивались, а повторялись. Для этого у нас есть свойство `border-image-repeat`.

Свойств `border-image-repeat` позволяет описать способ повторения неугловых изображений (для сторон и середины) и/или их масштабирования в порядке «верх, правая сторона, низ, левая сторона» (TRouBLe). В спецификации определены четыре возможных значения, но хорошую поддержку имеют только два. Значение `stretch` показывает, что для заполнения области изображение должно быть растянуто, а не многократно повторено, значение `repeat` — что изображение для заполнения области выкладывается в виде плитки (или повторяется).

Если область, выделенная для повторяющегося изображения, не делится ровно на ширину изображения, последнее выкладываемое в виде плитки может быть обрезано. При использовании значения `round` изображение для заполнения области должно быть выложено в виде плитки (повторено), но при этом оно масштабируется, возможно, с изменением соотношения сторон, что позволяет обойтись без обрезания. Предполагалось, что неподдерживаемое значение `space` будет использоваться для повторения нарезанной части общего изображения в стольких экземплярах, сколько могло бы поместиться целиком в предоставленной области, чтобы плитки чередовались с пустыми пространствами между ними, если предоставленная ширина не является точным произведением, получаемым при умножении размера изображения. Но это значение (может быть, временно) было удалено из спецификаций.

В наших примерах для кнопки используется значение `stretch`, а для марки — `round`. Для градиентов неизменно используется растягивание (`stretch`), поскольку их повторение приведет к созданию резких переходов в тех местах, где заканчивается одна и начинается другая плитка. Для марки повторение (`repeat`), похоже, имеет смысл, но мы не можем знать, делится ли ширина нашей конструкции на ширину изображения без остатка. Значение подгонки (`round`) приводит к небольшому искажению изображения, но это все же лучше, чем его обрезание.

Поскольку значение `round` не имеет всеобщей поддержки, в запасе остается значение `repeat`[1].

[1] WebKits-браузеры не поддерживают значения round и space, заменяя их значением repeat (что, по-моему, все же лучше простого отказа).

Особый интерес представляет стрелка. Конечно же, повторять ее мы не хотим. Ее можно растянуть, но только немного, пока изображение не стало искаженным. Из-за особенностей ее формы для высоты верхней и нижней частей разрезаемого изображения задано нулевое значение, поэтому, если мы применим растяжение той части, в которой находится стрелка, она не изменит своей формы:

```
.button {
  border: solid 5px;
  border-image: url(button_bi.png) stretch 5 fill;
}
.stamp {
  border: solid 9px;
  background-color: #dedeef;
  border-image: url(stamp.png) round 9 / 9px / 12px;
}
.arrow {
  border: solid;
  border-width: 05px 020px;
  border-image: url(arrow.png) stretch 05020 fill / 05px 020px;
}
```

Для стрелки или кнопки значение `stretch` можно не указывать, поскольку оно используется по умолчанию.

Краткая форма записи свойства border-image

Заметьте, в последнем примере кода вместо многоточий стоят точки с запятой. Ими завершаются различные свойства, из которых состоит краткая запись свойства `border-image`. Но мы не будем работать во всех браузерах. Префиксы ставятся для мобильных WebKit-браузеров, включая браузеры для Android 4.2 и iOS 5.1, и для браузера Opera вплоть до версии 12.1. Свойство `border-image` поддерживается в IE (начиная с IE11) без префикса:

```
.stamp {
  background-color: #ccc;
  border: solid 9px transparent;
  -webkit-border-image: url(stamp.png) 9 / 9px / 12px round;
  -o-border-image: url(stamp.png) 9 round;
  border-image: url(stamp.png) round 9 / 9px / 12px;
  }

.button {
  border: solid 5px transparent;
  -webkit-border-image: url(button.png) 5;
  -o-border-image: url(button.png) 5;
  border-image: url(button.png) 5 fill;
  }

.arrow {
  border: solid transparent;
```

```
border-width: 1px 5px 1px 20px;
-webkit-border-image: url(arrow.png) 15120 / 05px 020px stretch;
-o-border-image: url(arrow.png) 15120 / 05px 020px stretch;
border-image: url(arrow.png) stretch 05020 fill / 05px 020px;
}
```

Надеюсь, что теперь вы уже хорошо разобрались в том, как создавать границы с использованием изображений. Помощь в этом деле можно получить благодаря нескольким инструментальным средствам. Ссылки на эти средства и на ряд демонстрационных примеров кнопок, стрелок и марок можно найти в онлайн-ресурсах к главе.

Режим flexbox

Ожидаемый окончательный синтаксис режима разметки гибкого блока flexbox (flexbox layout mode) в данный момент поддерживается всеми современными браузерами, а некоторые версии flexbox поддерживаются всеми мобильными браузерами, поэтому об этом режиме стоит упомянуть, в частности, по той причине, что flexbox позволяет разработчикам без особых усилий создавать гибкие многоколоночные разметки (рис. 11.4).

Режим разметки flexbox придает разметке веб-страниц особую гибкость. Дочерние элементы гибкого контейнера могут размещаться горизонтально, вертикально, в исходном порядке или не соблюдая этот порядок. Дочерние элементы могут гибко изменять свою ширину и высоту и избегать выхода за пределы своих родительских элементов или пустого пространства.

Для работы CSS дает нам несколько новых свойств (в данный момент это может быть неокончательная их версия). Существующие сейчас новые flexbox-свойства состоят из свойств порядка и ориентации (куда входят свойство направления `flex-direction`, свойство переносов `flex-wrap`, свойство заливки `flex-flow` и свойство порядка `order`), свойств гибкости (это свойства разрастания `flex-grow`, сжатия `flex-shrink`, сохранения основы `flex-basis` и краткая форма записи свойства `flex`), свойств выравнивания (к ним относятся свойства выравнивания содержимого по одной из сторон `justify-content`, выравнивания элементов `align-items`, самовыравнивания `align-self` и выравнивания содержимого `align-content`, а также новые значения свойства `display`).

Спецификации flexbox добавляют к свойству `display` два значения: `flex` и `inline-flex`. Значение `flex` или `inline-flex` (в IE10 `-ms-flexbox`) применяется к свойству `display` того родительского элемента, чьи дочерние элементы нужно позиционировать[1]. По умолчанию используется значение `flex`, при котором

[1] Префикс следует добавлять не к названию свойства, а к значению, поскольку экспериментальным для некоторых браузеров является не свойство display, а новые значения. В WebKit-браузерах следует использовать объявление display: -webkit-flex;, а в IE10 — объявление display: -ms flexbox;. Для Opera и Firefox 20+ значения с префиксами не нужны.

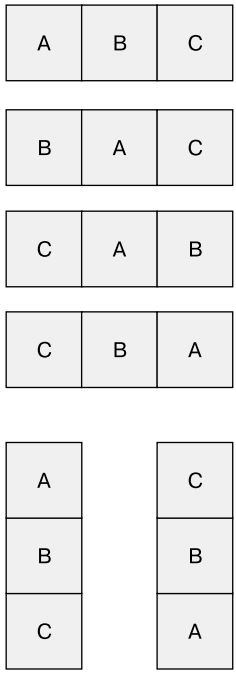


Рис. 11.4. Разметка и визуальный порядок могут изменяться при нетронутом исходном коде HTML

создается четное количество столбцов из получивших гибкость дочерних элементов. Дополнительные свойства позволяют изменить порядок на обратный, включить охват, изменить порядок, создать вместо столбцов отцентрованные строки и т. д., и все это без вмешательства в исходное HTML-содержимое. Позволяя CSS обеспечивать гибкость в разметке в сочетании с медиазапросами, мы можем отправлять различную разметку одного и того же содержимого разным конфигурациям окна просмотра.

Основой для различной разметки, показанной на рис. 11.4, послужил один и тот же код HTML:

```
<article>
  <div>A</div>
  <div>B</div>
  <div>C</div>
</article>
```

Итак, как же удалось изменить разметку, не касаясь ее основы? Все далось не так уж просто. Нам по-прежнему пришлось работать с разными синтаксисами для разных браузеров:

```
article {
  display: -webkit-box;
  display: -moz-box;
```

```
  display: -webkit-flex;
  display: -moz-flex;
  display: -ms-flex;
  display: flex;
}
```

Предыдущий код является основой для создания столбцов из гибких дочерних элементов. Если посмотреть в онлайн-ресурсы к главе, можно увидеть, что теперь `<div>`-контейнеры следуют друг за другом по горизонтали, как будто мы превратили их в плавающие элементы с перемещением к левому краю; независимо от количества содержимого, добавляемого к каждому вложенному `<div>`, все они будут одинаковой высоты.

К сожалению, все еще приходится поддерживать разнообразные спецификации — как с префиксами, так и без них. Поскольку свойство `display` имеет давнюю историю, префиксы производителей добавляются не к нему, а к значению: для старых WebKit-браузеров и Firefox вплоть до версии 17 мы включили значения `-webkit-box` и `-moz-box`. Затем включили префиксный вариант, рекомендованный для Chrome и BB10, IE10 и Firefox 17–19. На момент написания данной книги FF 20+, Opera, IE11 beta и Opera Mobile уже не использовали префиксную версию. IE10 поддерживает появившийся в феврале 2012-го синтаксис tweener, немного отличающийся от варианта спецификации, поддерживаемого другими браузерами и IE11 beta.

Предыдущий код предназначен исключительно для создания столбцов. А можно было бы создать строки. Можно было бы объявить ровные столбцы. Можно было бы изменить порядок представления этих столбцов на обратный. Все это можно было бы сделать с помощью других свойств flexbox.

ПРИМЕЧАНИЕ

Следует заметить, что дочерние элементы flexbox с заданным абсолютным позиционированием не могут быть flexbox-элементами, поскольку абсолютно позиционированные элементы исключаются из потока данных документа.

В браузерах модуль разметки гибких блоков реализуется в условиях развития спецификации. Из-за этого в разных браузерах реализованы разные синтаксисы. Далее в разделе используется только тот синтаксис, который был в текущей спецификации и может совпадать, а может и не совпадать с тем синтаксисом, который станет действующим на момент прочтения книги. Я включила в книгу рассмотрение технологии flexbox, несмотря на ее нестабильное состояние, потому что в сочетании с медиазапросами эта технология очень эффективна для разработки программного обеспечения мобильных устройств. И даже притом, что включенный в книгу синтаксис работает только в бета-версиях браузеров (IE11, Chrome 29+) и Opera Mobile с Presto (Opera 12.1), основная идея того, как он работает, не изменится. Для получения новейших примеров синтаксиса свойств и значений обратитесь к онлайн-ресурсам к главе.

Для выравнивания гибких дочерних элементов не по горизонтали, а по вертикали можно воспользоваться свойством `flex-direction`, указывающим направление

flexbox-разметки. Поскольку мы опустили это свойство, ему было задано значение по умолчанию `row`, при котором из дочерних элементов создается строка. Другие варианты включают значения `row-reverse`, `column` и `column-reverse`.

По умолчанию flex-контейнер представляет собой единую строку. Чтобы сохранить эту исходную разметку в одну строку, для свойства `flex-wrap` можно явно указать значение, исключающее перенос, — `nowrap`, а чтобы сделать возможной многострочную разметку — установить для этого свойства значение `wrap` или `wrap-reverse`.

Свойство `flex-flow` является краткой формой записи свойств `flex-direction` и `flex-wrap`, комбинация которых определяет основу разметки flexbox.

Если нужно изменить порядок отображения элементов, которым придана гибкость, может быть использовано свойство `order`. Значение свойства `order` задается в отношении не родительских, а дочерних flexbox-элементов. Для изменения порядка на противоположный можно воспользоваться объявлением `flex-direction: row-reverse;`:

```
article {
  display: flex;
  flex-direction: row-reverse;
}
```

Для перераспределения отдельно взятого дочернего элемента нужно применить объявление `order: -1;` или использовать какое-нибудь другое, еще меньшее значение, чтобы заставить дочерний элемент, в отношении которого оно применено, появиться первым. А чтобы элемент появился последним, нужно применить объявление `order: 1;` или использовать какое-нибудь другое, еще большее значение, превосходящее значения для элементов одного с ним уровня:

```
article {
  display: flex;
}
div:nth-of-type(2) {
  order: -1;
}
```

Если есть элемент `<article>` с тремя `<div>`-контейнерами (A, B и C), все три столбца будут выстроены в ряд, но B появится для наблюдателей первым, как будто использовался порядок B — A — C. А если воспользоваться объявлением

```
div:nth-of-type(2) {
  order: -1;
}
```

то будет применен порядок A — C — B.

flex

Свойство `flex` определяет значение свойств `flex-grow`, `flex-shrink` и `flex-basis`. Вместо трех развернутых определений свойств нужно использовать краткую форму `flex`.

Гибкость — это способность контейнера изменять свою ширину и высоту, чтобы поместиться в доступном пространстве, позволяя установить размеры для наших элементов. Свойство `flex` применяется к дочерним, а не к родительским элементам flexbox. При настройке дочерних элементов flexbox браузер построчно устанавливает размер элементов, в отношении которых объявлено свойство `flex`, равномерно распределяя оставшееся свободное пространство между элементами, для которых не установлено свойство `flex`.

Свойству `flex` можно передать три значения. Компоненты свойства `flex-grow` определяют, до какой степени гибкие элементы будут увеличиваться в размерах относительно одноуровневых с ними родственных элементов внутри родительского элемента flexbox при распределении свободного пространства. По аналогии с этим (или в противоположность этому, но это не самое подходящее слово) коэффициент `flex-shrink` определяет, до какой степени элемент будет уменьшаться в размерах относительно одноуровневых с ними родственных элементов внутри родительского элемента flexbox при распределении, связанном с отсутствующим пространством. Свойству `flex-basis` передаются те же самые значения, что и свойству `width`, определяющие исходный основной размер элемента перед тем, как свободное пространство будет распределено в соответствии со значениями свойств `flex-shrink` и `flex-grow`. По умолчанию используется установка `flex: 110;`.

Если нужно, чтобы элемент A был в два раза шире элемента B, а B — в два раза шире элемента C, можно воспользоваться следующим кодом (см. онлайн-ресурсы к главе):

```
div:nth-of-type(1) {
  flex: 4;

}
div:nth-of-type(2) {
  flex: 2;

}
div:nth-of-type(3) {
  flex: 1;
}
```

Если при выполнении разработки под различные размеры окна просмотра использовать свойства разметки flexbox в сочетании с медиазапросами, это может упростить процесс разработки. Для широкого экрана можно создать на странице три столбца, помещая сопутствующую информацию слева, основное содержимое — по центру, а сноски — справа. А на небольшом телефоне можно, не затрагивая содержимого страницы, поместить основное содержимое наверху, за ним — сопутствующие данные, а снизу — ссылки. С точки зрения семантики я бы поставила содержимое первым. Чаще всего начинать нужно с разработки под мобильное устройство:

```
<article>
  <div>Основные данные — в центре широкого экрана и первые на узком
```

```
  экране</div>
  <aside>С левой стороны на широком экране и после основных данных –
  на узком</aside>
  <footer>С правой стороны на широком экране и в нижней части –
  на узком</footer>
</article>
@media screen and (min-width: 600px) {
  article {
    display: flex;
    flex-direction: row;
  }
  article > * {
    flex: 1;
  }
  aside {
    order: -1;
  }
  article > div {
    flex: 3;
  }
}
```

Обратите внимание на то, что порядок, установленный для элемента `<aside>`, заставляет его появиться первым, как видно в примере, имеющемся в онлайн-ресурсах к главе. Все три родственных элемента того же уровня получают установку `flex: 1;`, которая для основного содержимого переписывается установкой `flex: 3;`. Значит, статья для элементов `aside`, `div` и `footer` будет разбита в пропорциях 20, 60 и 20%. Также обратите внимание на то, что свойство `display: flex;` задано в отношении родительского элемента, а все остальные свойства (кроме `flex-direction`) — в отношении дочерних элементов.

В данном случае объявлять отдельную разметку для экранов меньшего размера не нужно, поскольку исходная разметка браузера очень похожа на ту, которая получается при объявлении `flex-direction: column`. Тем не менее можно было бы воспользоваться следующим объявлением:

```
@media screen and (max-width: 600px) {
  article {
    display: flex;
    flex-direction: column;
  }
}
```

Свою раскладку нужно всегда просматривать и в небольших, и в больших форматах, расставляя контрольные точки медиазапросов, сообразуясь с предназначением или необходимостью.

Обнаружение возможностей с помощью @supports

Хотя в мобильном пространстве эт-правило (то есть правило, начинающееся с символа «эт» — @) `@supports` не поддерживается и существует только на уровне вари-

анта рекомендаций W3C, оно уже поддерживается в Firefox и Opera. Эт-правило `@supports` позволит создавать одну разметку для браузеров, поддерживающих flexbox, а другую — для браузеров, которые эту технологию не поддерживают, не прибегая при этом ни к каким трюкам.

Пройдет еще немало времени, пока flexbox станут поддерживать все пользовательские устройства. Мы конечно же увидим другие новые CSS-свойства, которые, как и flexbox, будут использовать новые значения для поддерживаемых CSS-свойств вроде `display: flex;`. Правило `@supports` похоже на правило `@media`, но вместо определения соответствия браузеров количественным показателям браузера и устройства соответствие будет определяться на основе поддержки браузером тех или иных свойств CSS:

```
@supports (display: flex) and (background-color: red) {
  h1 {color: green;}
}
```

Этот код обусловит то, что все элементы `<h1>` в браузерах, поддерживающих как объявление `display: flex;`, так и объявление `background-color: red;`, будут выводиться зеленым цветом. И ни один из элементов при этом не будет выведен на фоне красного цвета. Будет лишь проверена поддержка свойств и значений свойств. В промежуточный период некоторым браузерам был добавлен ряд возможностей обнаружения свойств с помощью правил `@media`:

```
@media screen and (-webkit-transform-3d) {
  h1 {
    -webkit-transform: translateZ(0) rotate(5deg);
    -webkit-animation: makemedizzy 1s infinite;
  }
}
```

Предыдущий код предназначен для всех WebKit-устройств, поддерживающих 3D-преобразования, он задает вращение и анимацию элементов `<h1>` внутри документа. Компонент обнаружения свойства, входящий в этот медиазапрос, использует префикс. Этот медиазапрос будет предназначен для WebKit-браузеров, которые все еще поддерживают префиксную форму для CSS-свойства `transform`[1]. Чтобы он мог работать в браузерах, которые больше не нуждаются в префиксах, нужно использовать следующий код:

```
@media screen and (transform-3d) {
  h1 {
    transform: translateZ(0) rotate(5deg);
    animation: makemedizzy 1s infinite;
  }
}
```

[1] Он также может предназначаться для некоторых браузеров Opera, выпущенных еще до перехода на основу Blink, таких как Opera Presto, в котором была добавлена ограниченная поддержка некоторых WebKit-свойств и значений с указанием префикса.

Этот медиазапрос будет понятен только тем браузерам, которые поддерживают `transform-3d`.

Запрос `@media` можно использовать также для обнаружения свойств, поддерживающих анимацию и переходы, как с префиксами, так и без них. Со временем этот способ обнаружения свойств будет заменен ранее рассмотренным правилом `@supports`. После того как поддержка правила `@supports` будет реализована, это правило сможет обнаруживать поддержку всех свойств и значений. Правило `@media` ограничено только тремя свойствами и не различает значения свойств.

Итак, почему интерес проявляется к `@supports`, а не к тому, чтобы просто позволить браузерам проигнорировать те свойства, которые они не поддерживают? Правило `@supports` позволит вам, к примеру, создать разметку сайта, используя flexbox, если эта технология поддерживается, и столбцы, если она не поддерживается, и не позволит по невнимательности применить к столбцам гибкую разметку.

Адаптируемая информация

Гибкая разметка позволяет без особого труда создавать гибкие макеты. К сожалению, свойства flexbox не имеют полной поддержки во всех браузерах. Спецификация CSS 2.1 предоставляет все инструменты для создания гибкой разметки, в то время как flexbox упрощает решение этой задачи.

До тех пор пока flexbox не будет поддерживаться на подавляющем большинстве устройств, для создания разметки, адаптирующейся к размерам экрана, проще будет по-прежнему работать с процентными отношениями вместо пикселов. При вводе элементов с фиксированной шириной создание гибкой разметки представляется более трудной задачей, но существует несколько приемов, упрощающих решение, казалось бы, весьма сложных проблем.

В качестве характерного примера можно привести необходимость в гибком изображении в заголовке: нужно, чтобы независимо от размеров устройства это изображение занимало всю ширину экрана, без необходимости увеличения масштаба страницы, делающего текст неразборчивым на небольших устройствах. Нужно, чтобы изображение имело 100 % ширины независимо от того, какова ширина экрана, 440 или 640 пикселов. Решение проблемы настолько простое, что оно фактически описывается предыдущим предложением:

```
header img {
  max-width: 100%;
  height: auto;
}
```

Материалы с фиксированной шириной, например изображения и видеосюжеты, можно объявить с любой шириной относительно ширины их родительского контейнера. В предыдущем случае рисунок будет отображен со своей исходной шириной (`width: auto;`), но если он не больше родительского элемента. Поскольку вместо значения свойства `width` было определено значение свойства `max-width`, изо-

бражение прекратит увеличиваться, как только достигнет фактической ширины своей среды распространения.

Если вы не против показа изображения в низком разрешении, можете воспользоваться объявлением `min-width: 100%` или просто `width: 100%;`. Пока дело не будет касаться растягивания градиента или какого-нибудь другого растягиваемого изображения, не объявляйте фактическое значение высоты, поскольку такое объявление, скорее всего, приведет к искажению соотношения сторон изображения. Именно поэтому мы включили в код объявление `height: auto;`.

Обслуживание изображений

Расширение и сжатие изображений не является панацеей, решающей все проблемы, связанные с изображениями на мобильных устройствах. Что касается памяти, то на мобильных устройствах ее объем, как правило, весьма ограничен. Тем не менее уже имеются устройства с высокими показателями DPI, которые неплохо справляются с большими изображениями, используя более высокую пропускную способность и больший объем памяти. Ограничения, накладываемые на объемы памяти, задержки и различные разрешающие возможности устройств, приводят к тому, что обслуживание изображений выходит за круг тех тривиальных вопросов, которые нас волновали в эпоху настольных устройств.

Retina®: дисплеи с высокой плотностью пикселов

Смартфон iPhone 4, выпущенный в 2009 году, был первым устройством с так называемым дисплеем Retina Display, имевшим разрешение 326 точек на дюйм (DPI). Планшетный компьютер iPad третьего поколения, выпущенный в 2012 году, имел удвоенное по сравнению с предыдущей версией разрешение 264 DPI. Первым ноутбуком с высоким DPI был Retina Display MacBook.

У исходного iPhone экран был 320 × 480 пикселов а у исходного iPad — 768 × 1024 пиксела. Первые Retina-версии этих устройств имели разрешение экрана 640 × 960 и 1536 × 2048 пикселов. Размер экранов остался прежним, но на площади, которую обычно занимал 1 пиксел, теперь отображались 4 пиксела, создавая экран с более высоким разрешением и плотностью.

Retina Display является устройством с высоким разрешением. Это торговая марка от Apple, означающая двойное разрешение. Оно не связано с каким-то конкретным значением DPI. Высокое разрешение применяется не только на устройствах Apple. На современном рынке есть устройства и с более высоким разрешением, чем у iPhone, но их производителям придется придумать их описания, не имеющие отношения к торговым маркам. Правильным будет неторговый термин «высокое разрешение».

С выпуском iPhone 4 веб-разработчикам пришлось работать с дисплеями Retina Display, то есть с дисплеями высокого разрешения.

Пиксел устройства является самой маленькой точкой цвета, отображаемой устройством, не всегда в точности совпадающей с пикселом CSS. Понимание того, в чем заключается разница, может помочь избавиться от связанной с этим путаницы.

Пиксельная плотность представляет собой число или соотношение пикселов устройства, приходящихся на один пиксел CSS. Устройство может быть способно отображать более одного (или менее одного) пиксела устройства в пикселе CSS. В то же время *разрешение* является произведением ширины устройства на его высоту, выраженным в пикселах.

Количество точек на дюйм, или DPI, является плотностью дисплея. DPI является частным от деления количества отображаемых пикселов на размер экрана устройства в дюймах. Например, устройство с экраном шириной 4 дюйма отображает 800 пикселов, тогда:

800 пикселов/4 дюйма = 200 пикселов/дюйм.

Устройство с показателем DPI от 200 пикселов/дюйм и выше считается устройством с высоким DPI. Если, к примеру, взять iPad, то стандартное устройство изначально имело разрешение 768 × 1024 пиксела, а устройство стандартного размера (не мини) версии с высоким DPI имело разрешение 1536 × 2048 пикселов, и хотя оба устройства имели одинаковую высоту 7,75 дюйма[1], их показатели DPI были равны 132 и 264 соответственно:

1024 пиксела/7,75 дюйма = 132 пиксела/дюйм;

2048 пикселов/7,75 дюйма = 264 пиксела/дюйм.

Чем выше DPI, тем меньше пиксел устройства, что позволяет получить более качественное изображение. Высококачественные изображения, как правило, просто более крупные изображения. Продолжая рассматривать пример, можно сказать, что новейший iPad имеет удвоенное количество пикселов и, следовательно, получает преимущество за счет отображения изображений, имеющих в четыре раза больше пикселов, чем при отображении тех же изображений на исходном iPad.

Изображения, созданные с помощью программ редактирования изображений, имеющие форматы JPEG, PNG и GIF и отправляемые по Сети, имеют одно и то же разрешение 72 пиксела/дюйм. Изображение с низким разрешением, демонстрируемое на экране с высоким разрешением, может казаться нерезким.

Для заполнения фона всего окна браузера устанавливаем высоту и ширину изображения первого плана или фона равными размерам экрана с низким DPI. Но мы можем обслуживать изображения в четыре раза больше (если изображение вдвое выше и вдвое шире, значит, оно в четыре раза больше исходного размера), чтобы на устройствах с высоким разрешением они выглядели более яркими и красивыми.

Это вызывает ряд проблем: отправлять изображения с высоким DPI на устройства с низким показателем DPI не имеет смысла, поскольку чем больше изображение, тем выше нужна пропускная способность, больше размер файла и больше объем занимаемой им памяти, а мобильные устройства, как известно, имеют ограниченный объем памяти.

---

[1] Когда про размер экрана говорится, что он равен 9,5 или 9,7 дюйма, имеется в виду размер по диагонали от верхнего левого до нижнего правого угла.

Хотя мы пока не располагаем API-функциями, позволяющими определять объем оставшейся памяти, мы можем определить пропускную способность. Используя комбинацию JavaScript и CSS, можно выборочно отправлять изображения высокого разрешения только пользователям, применяющим высокоскоростную сеть и дисплей с высоким разрешением, который может использовать более крупные изображения. Медиазапросы на основе скорости соединения уже были предложены, но поддержки пока не получили (а может быть, уже получили?).

Скорость соединения. Тип текущего соединения можно получить благодаря свойству JavaScript `navigator.connection.type`, которое возвращает значения `UNKNOWN`, `ETHERNET`, `WIFI`, `CELL_2G`, `CELL_3G` или `CELL_4G`:

```
var connection, speed;
var connection = navigator.connection ||
      navigator.mozConnection ||
      navigator.webkitConnection ||
      {'type':'0'};

// установка скорости загрузки
switch(connection.type) {
  case connection.CELL_3G: // 3G
    speed = 'medium';
    break;
  case connection.CELL_2G: // 2G
    speed = 'slow';
    break;
  default:
    speed = 'fast';
}

document.body.classList.add(speed);
```

Можно изменить класс тела документа и цель и на основе этого класса импортировать изображения в высоком разрешении:

```
@media screen and (-webkit-min-device-pixel-ratio: 2),
  screen and (min--moz-device-pixel-ratio: 2),
  screen and (-min-moz-device-pixel-ratio: 2),
  screen and (-o-min-device-pixel-ratio: 2/1),
  screen and (min-device-pixel-ratio: 2) {

body.fast {
    background-image: url(../hidpi/bgimg.jpg);
  }
}
```

Следует учесть, что на некоторых устройствах изображения с параметром одного из размеров больше 1024 пикселов будут размещаться в памяти.

background-size

При добавлении изображений с высоким разрешением хочется, чтобы они занимали то же самое физическое пространство, что и на устройствах, не изготовленных

по технологии Retina. Гарантию появления одинаковых изображений независимо от разрешения может дать свойство background-size, рассмотренное в главе 9.

Фактически разные дисплеи показывают 72, 96 или 144 DPI. При отправке обычного изображения или изображения для дисплея Retina, которое больше по размерам в четыре раза, нужно, чтобы фоновые изображения отображались так, будто они одинакового размера, только выглядели четче, если устройство способно их обработать.

Для обеспечения отображения изображения размером 100 × 100 пикселов как на экранах 100 × 100 пикселов, так и на экранах 200 × 200 пикселов с технологией Retina нужно использовать свойство background-size:

```
.icon {
  -size: 100px 100px;
    background-image(../lodpi/icon.jpg);
}

@media screen and (-webkit-min-device-pixel-ratio: 2),
  screen and (min--moz-device-pixel-ratio: 2),
  screen and (-min-moz-device-pixel-ratio: 2),
  screen and (-o-min-device-pixel-ratio: 2/1),
  screen and (min-device-pixel-ratio: 2) {
  .fast .icon {
    background-image(../hidpi/icon.jpg);
  }
}
```

В предыдущем коде, даже притом что изображение в высоком разрешении может быть в четыре раза больше изображения в низком разрешении, оба они будут занимать одинаковое пространство.

URI-идентификаторы данных

Иногда использование URI-идентификаторов данных может обеспечить более высокую производительность, чем отправка дополнительных HTTP-запросов на обслуживание обычных изображений. URI-идентификаторы данных являются строкой, представляющей двоичные файлы изображений.

Поскольку для представления двоичных данных в URI-идентификаторах данных используется строка, их размер может стать довольно большим, если не громадным. Для небольших изображений, например аватаров и значков, URI-идентификаторы данных, вероятнее всего, повысят производительность загрузки за счет сокращения количества HTTP-запросов (и, возможно, DNS-поисков). Для больших изображений, например полноэкранных фоновых изображений с высоким разрешением, возможно, стоит воспользоваться DNS-поиском и HTTP-запросом. Однозначно верного решения не существует.

На сайте, подобном Twitter, где отображаются аватары всех отслеживающих ваши сообщения людей, всех людей, за сообщениями которых следите вы сами, и всех людей, которым они отвечают, вряд ли удастся успешно создать кэшируемый спрайт (который рассматривается далее в подразделе «Спрайты») аватаров.

Люди могут изменять свои Twitter-аватары, когда им заблагорассудится. Поэтому даже если бы Twitter и создавал спрайты, они должны были бы обновляться с каждым запросом. В случае с сайтами типа Twitter применение URI-идентификаторов данных может иметь вполне определенный смысл. Такие сайты требуют небольших, некэшируемых и не собираемых в спрайты изображений. Отдельный HTTP-запрос для каждого из таких изображений будет исполняться намного медленнее, чем включение изображений в качестве URI-идентификаторов данных в виде отдельного текстового файла с использованием единственного HTTP-запроса.

Спрайты

Спрайты — это большие файлы изображений, содержащие несколько изображений меньшего размера. Спрайты используются для сокращения количества HTTP-запросов и DNS-поисков и повышения скорости загрузки фоновых изображений веб-страниц. Спрайты могут использоваться и в анимации.

Например, если на сайте используется множество красочных значков или отображаются значки множества рейтинговых сайтов и вам известен ограниченный круг отображаемых или повторяющихся значков, все эти значки можно поместить в одно изображение. Затем, используя свойство `background-position`, можно будет показывать пользователю только отдельные части этого изображения. Если все значки имеют один и тот же цвет, можно будет воспользоваться рассматриваемыми далее шрифтовыми значками. Пример спрайта показан на рис. 11.5.

Рис. 11.5. Спрайт, состоящий из значков популярных приложений и сайтов

Спрайты могут использоваться также в анимации в сочетании со значениями `steps()` свойства `animation-timing-function`, как мы уже делали в случае с анимированием лемминга в главе 10.

Чтобы создать танцующий значок, можно использовать спрайт, показанный на рис. 11.6, со следующим кодом CSS:

```
.psy {
  width: 22px;
  height: 40px;
  background-image: url(sprite.png);
  animation:
    dance 4s steps(23, start) infinite,
    movearound 9s steps (23, start) infinite 45ms;
}

@keyframes dance {
  0% {
```

```
    background-position: 0 0;
  }
  100% {
    background-position: -506px 0;
  }
}
@keyframes movearound {
  0% {
    transform: translatex(-300px);
  }
  100% {
    transform: translatex(300px);
  }
}
```

Рис. 11.6. Спрайт для символьной анимации

В танцующей анимации изменяется позиция фона. Анимируемые символы имеют размер всего лишь 22 пиксела в ширину и 40 пикселов в высоту. Приблизительно каждые 45 мс фоновое изображение смещается на 22 пиксела влево, показывая символ, расположенный справа от предыдущего символа. Таким образом мы можем создать видимость того, что содержимое `<div>`-контейнера танцует. Анимация со смещением в три раза быстрее была бы более плавной, но для нее потребовалось бы свыше 40 кадров иллюстраций.

Небольшие спрайты сокращают количество HTTP-запросов, снижают вероятность мерцания, связанного с задержками в загрузках изображений, а кроме того, они могут быть кэшированы и использованы в анимации. Но большие спрайты создают риск возникновения проблем из-за ограниченного объема памяти мобильных устройств. Спрайты нужно использовать осмотрительно, и желательно, чтобы их длина не превышала 1024 пикселов.

image-set()

Браузеры Safari 6 и Chrome 21 поддерживают значение `image-set()`, позволяющее обслуживать различные фоновые изображения для дисплеев с различной плотностью пикселов:

```
body > header {
  background-image: url(images/header.png);
  background-image: -webkit-image-set(url(images/header.png) 1x,
    url(images/header_2x.png) 2x);
  height:60px;
}
```

В соответствии с заявлениями рабочей группы CSS Working group, это значение еще не готово для реализации. Я включила его сюда только лишь для того, чтобы

вы знали о его существовании. Будем надеяться на то, что вскоре это значение будет реализовано, поскольку изображениями легче управлять с использованием синтаксиса `srcset`, чем с помощью блоков медиазапросов.

Шрифтовые значки

На многих сайтах и во многих приложениях используется большое количество небольших изображений, повсеместная распространенность которых привела к представлению в файлах шрифтов. Например, цветок на тыльной стороне карт CubeeDoo и более мелкие образы в теме «образы» на самом деле, как показано на рис. 11.7, являются исходным шрифтом CubeeDoo.

Рис. 11.7. Значки, используемые в CubeeDoo

На большинстве устройств в исходных наборах символов имеется свыше 10 000 различных символов. Скорее всего, вам удастся найти нужный образ, будь то конверт, стрелка или что-нибудь другое. Поскольку шрифтовые значки являются всего лишь символами, их размер и цвет можно легко изменять с помощью кода CSS. Шрифтовые значки масштабируются без искажений, изменяют цвет без использования программ редактирования изображений и, если входят в состав шрифтов, которые можно найти на большинстве устройств, загружаются, не требуя для этого дополнительных HTTP-запросов.

Когда будет найден нужный символ, его можно включить непосредственно в HTML в качестве сгенерированного содержимого, используя для этого псевдоэлементы `::before` и/или `::after` или добавля символ в SVG и включая эту конструкцию в фоновое изображение или в URI-идентификатор данных.

При любой возможности следует выбирать значки, а не изображения. Многие компании создают собственные наборы символов с собственной уникальной иконографией.

CSS-маскирование: создание прозрачных изображений формата JPEG

Хочется рассмотреть еще один прием: маскирование. Иногда изображение в формате PNG требуется использовать только из-за того, что нужна прозрачность. Но для детализированных PNG-изображений создаются намного более длинные файлы, чем для JPEG-изображений. Но в JPEG-формате нет прозрачности, поэтому может показаться, что для предоставления прозрачности без PNG-формата не обойтись. Прозрачные JPEG-изображения можно создавать с помощью маскирования, поэтому вместо PNG-формата можно использовать JPEG-формат, экономя при этом трафик и память.

CSS-маскирование позволяет нам накладывать небольшие файлы JPEG с восьмиразрядным прозрачным PNG-изображением, чтобы создать на экране дисплея прозрачные участки исходного JPEG-изображения. Маскируя JPEG-изображение прозрачным PNG-изображением, мы можем создавать прозрачные участки на основе исходного изображения, существенно сокращая необходимое количество байтов:

```
div {
  background-image:url(images/smallerFileThanPNG.jpg);
  -webkit-mask: url(images/partToShow.png);
}
```

Хотя загрузка двух изображений — фонового и маски — может добавить дополнительный HTTP-запрос, экономия байтов вполне может превысить стоимость этой потери. В примере, имеющемся в онлайн-ресурсах к главе, исходное PNG-изображение с высоким разрешением занимает файл размером 551 Кбайт. Конвертирование из PNG-формата в JPEG-формат без прозрачности приводит к появлению изображения, занимающего файл размером 88 Кбайт, а монотонная маска яркости занимает файл размером всего 4 Кбайт, что в сумме составляет 92 Кбайт и приводит к существенной экономии объема по сравнению с прозрачным PNG-изображением.

Маскирование изначально было реализовано только в WebKit-браузерах, но свойство введено в стандарт консорциумом W3C. В исходном синтаксисе попрежнему используется префикс, и свойство поддерживается только в браузерах WebKit; базовая поддержка имеется во всех мобильных WebKit-браузерах.

Клиентские подсказки

Свойства клиентских подсказок `Client-Hints` пока еще не реализованы и не фигурируют даже в черновой форме спецификации. Но, поскольку они вполне могут появиться, что произведет грандиозный эффект, я все же решила о них упомянуть.

Свойства `Client-Hints` являются подсказками, которые браузер будет отправлять серверу вместе с заголовком запроса. Когда появится их поддержка, ожидается, что будут передаваться три значения, `dpr`, `dw` и `dh`, предназначенные для задания пиксельного соотношения устройства, ширины экрана устройства и его высоты:

```
Client-Hints: dh=1280, dw=768, dpr=2.0
```

Браузер сможет информировать сервер с помощью заголовка запроса. Затем на основе спецификаций браузера сервер сможет подобрать изображения наиболее подходящих размеров.

Это может показаться созвучным с мониторингом информации от пользовательского агента. Производители браузеров копируют UA-строки друг у друга, чтобы пройти несовершенные процедуры UA-мониторинга, развернутые на множестве сайтов. Поскольку разработчики создали слишком много процедур UA-мониторинга, поставщики браузеров включают строки друг друга, чтобы получить хоть какое-то подобие поддержки. Остается надеяться, что подобная проблема минует свойства `Client-Hints`, поскольку от них, кроме сокращения потребления трафика и сообщения о высоте и ширине экрана вашего устройства, ничего другого не требуется.

12 Разработка приложений для мобильных устройств

Разработка сайтов и приложений с целью их использования на мобильных устройствах отличается от разработки программных средств для настольных компьютеров, и не только визуальным дизайном. Характерные для мобильной среды размеры экрана, отсутствие указательного устройства, ограниченная скорость загрузки и разница в целях пользователей оказывают существенное влияние на дизайн и решение вопросов реализации.

Хотя многие утверждают, что сайты, предназначенные для мобильных устройств, должны иметь ограниченные возможности, это не так. Вся информация, которую нужно сделать доступной для пользователей настольных компьютеров, должна быть доступна и пользователям мобильных устройств. Вы просто не сможете прикрепить все это содержимое к главной странице мобильной версии сайта.

Да, в отношении некоторых сайтов цели обычного посетителя, сидящего за экраном настольного компьютера, и такого же посетителя, пользующегося мобильным устройством, могут быть разными, но имеется множество посетителей, которых нельзя отнести к обычным. Доказательством может послужить то, что у посетителя мобильной версии сайта меньше времени и ему нужен всего лишь ваш номер телефона[1] или адрес. Следует учесть, что все большее количество людей выходят в Интернет только через мобильные устройства. И посетителю мобильной версии сайта может понадобиться только ваш телефонный номер. Или же он может искать ваш сайт в биржевом зале, желая сделать огромные инвестиции. Ему нужно со своего смартфона получить доступ к вашим годовым отчетам, пресс-релизам и совету директоров. И то, что у него меньше экран, не означает, что ему нужно меньше информации или функциональных возможностей. У мобильной версии сайта должно быть такое же содержимое и такие же функциональные возможности, как и у «полноценной» версии сайта, хотя разметка, иерархия и пути обнаружения информации могут быть другими.

---

[1] Номер телефона должен быть ссылкой на этот номер, например 415.555.1212, о чем уже упоминалось в главе 3, в теме обработки ссылок, характерных для мобильных версий сайтов.

Если в мобильную версию будет включена вся информация из полной версии сайта, то польза от него будет гарантирована независимо от способа доступа пользователя к Интернету.

Нужно не давить на посетителей слишком большим объемом информации, а обеспечивать ее доступность. По аналогии с этим не нужно чем-то удивлять и тех пользователей, которые сидят за дисплеями настольных компьютеров. Следовательно, не нужно ограничивать возможности мобильной версии вашего сайта. Вместо этого лучше создавать сайт сначала в мобильной версии, включая в него всю информацию, необходимую для любых пользователей, но при этом не стараться кого-нибудь чем-нибудь удивить. Создавая сайт с прицелом на работу на мобильных устройствах или даже создавая сначала его мобильную версию, нужно делать так, чтобы версия, предназначенная для настольных систем, производила наиболее яркое впечатление. Если внести небольшую корректировку в информационно полноценную мобильную версию сайта, ваш вариант сайта для настольного компьютера будет проще с точки зрения навигации и не окажется перегружен лишними элементами, чего трудно избежать, начиная разработку с настольной версии.

Для мира мобильных устройств характерны такие ограничения, как разнообразные размеры экранов, порой весьма скромные, разнообразные средства указания (мышь, стилус, тонкие пальцы, толстые большие пальцы), ограниченная скорость загрузки, прерывающееся или непостоянное подключение к Сети, дозированная ширина полосы пропускания и различные методы ввода данных. Но при этом, в отличие от настольных компьютеров, отсутствуют ограничения, связанные с не самыми высокими возможностями браузеров. На настольных компьютерах приходится мириться с недостаточной поддержкой CSS3, HTML5 и сопровождающего API-интерфейса JavaScript в более ранних версиях получившего почти повсеместное распространение браузера Internet Explorer. Работая с браузерами iPhone, BlackBerry, Firefox mobile, Android и mobile Chrome, мы не испытываем подобных ограничений. Поддержка новых веб-технологий обеспечивается мобильными версиями WebKit-браузеров, IE10, Opera, Firefox для Android и Boot2Gecko, а также почти всеми браузерами смартфонов. Что касается HTML5, то эти мобильные браузеры, безусловно, относятся к классу A. В отношении CSS3 следует отметить, что мобильные браузеры являются наиболее совершенными из всех доступных в настоящее время браузеров.

Вопросы, требующие рассмотрения перед началом работы

Учет интересов целевой аудитории был важен и при работе для настольных компьютеров, а из-за ограниченных размеров мобильных устройств он приобретает еще большую актуальность. Перед тем как приступить к разработке, нужно определить, кем будет представлена ваша аудитория и каковы будут ее цели при посещении вашего сайта. Основной упор нужно делать на то, что эти люди пользуются мобильными

устройствами, имеющими средства беспроводной связи, работающими от автономных источников питания, постоянно включенными, все время находящимися при пользователях и, как правило, используемыми одним пользователем.

Нужно решить вопрос, кто именно будет посещать ваш сайт, на кого вы ориентируетесь: на юношей, девушек, мужчин или женщин. Создается ли сайт с прицелом на молодежь, профессионалов, неформалов, родителей? На кого именно он рассчитан? Зная свою аудиторию и то, каких действий с вашим приложением или на вашем сайте от нее ожидать, можно точнее определить внешний вид, круг предоставляемых функциональных возможностей и пути реализации пользовательского интерфейса.

Низкие текущие показатели статистики посещений определенной категории пользователей еще не означают, что этой категории не интересен ваш продукт, — они могут означать, что этот продукт просто не производит на них должного впечатления. Улучшив производительность и повысив удобство пользования продуктом, можно пробиться с ним на еще не освоенные рынки.

Никакого типичного пользователя не существует. Растет число людей, у которых нет ноутбука или настольного компьютера и которые входят в Интернет только через мобильные телефоны. Главным устройством для них является именно мобильный телефон. Разумеется, мобильными устройствами пользуются не только те, кто постоянно находится в движении, но и те, кто использует браузер телефона от случая к случаю, и их тоже нельзя упускать из виду.

Пользователи мобильных телефонов могут решать с их помощью сразу несколько задач. Они могут находиться в Twitter, проверять поступившие новости на Facebook, выступать в поддержку каких-нибудь движений, интересоваться сортами пива, проверять биржевые котировки или спортивные результаты, находясь в это время на какой-нибудь встрече или в учебном классе. Они могут стоять в очереди в магазине или находиться в автобусе, в 15 минутах езды до работы или школы, коротать время в вестибюле отеля или терминале аэропорта или просто маяться без дела.

В зависимости от типа приложения внимание пользователей к нему может быть весьма недолгим. Наиболее успешные привычные приложения как на мобильных, так и на настольных устройствах выполняют именно то, что нужно пользователю, вызывая у него чувство полного удовлетворения, при этом у них нет никаких лишних привлекательных элементов, которые могут отвлечь пользователя от поставленной задачи или сбить с намеченной цели. Мобильные версии сайтов и приложений должны делать то же самое. Независимо от типа устройства, ваше приложение или сайт должны быть *простыми*, *понятными* и *адекватными поставленной задаче*.

- **Простота.** У пользователей нет времени на то, чтобы долго в чем-то разбираться, — они должны сразу же понять, как пользоваться приложением или перемещаться по сайту и как с их помощью можно выполнить поставленную задачу. В противном случае они переключатся на использование другого приложения.
- **Понятность.** Работу приложения нужно сделать понятной, сведя к минимуму элементы управления и абсолютно точно обозначив их функции при минимуме

текста и узнаваемой иконографии. Не стоит изобретать колесо: выберите те свойства, элементы управления и значки, которые уже стали стандартом[1].

- **Адекватность.** Поместите наиболее важную информацию в верхней части экрана, где она будет заметнее и ее будет легче прочитать, и постепенно, по мере продвижения по странице наращивайте информационное наполнение.

Если говорить о конкретных действиях, поместите наиболее важные вызовы тех или иных действий ближе к верхней части экрана. На портативных устройствах наиболее часто используемые элементы управления нужно поместить на левой стороне панели управления в нижней части экрана.

Размышления по поводу дизайна

Бывший исполнительный директор компании Apple Стив Джобс (Steve Jobs) был бы счастлив, если бы все веб-приложения в iPhone выглядели так же хорошо, как исходные приложения Apple iPhone. Команда Apple, занимающаяся вопросами пользовательского восприятия, определила цветовые решения, систему навигации и графические стандарты для каждого приложения в зависимости от его типа. Эти стандарты по-прежнему считаются лучшими для iPhone, iPod и iPad. Аналогичные подходы к дизайну есть и в других операционных системах. Ссылки на дизайнерские стандарты Apple, Android, BlackBerry, Windows Phone, Firefox OS и других операционных систем можно найти в онлайн-ресурсах к главе.

Чтобы определить, какой из стилей приложений нужно использовать, следует рассмотреть несколько вопросов: Какова была мотивация пользователя для использования приложения? Каковы назначение, цель или область применения приложения? На создание какого пользовательского восприятия оно рассчитано?

Определение типа создаваемого приложения и его дизайна основано на стандартах, определенных для приложений данного типа. Для чего создается приложение: для серьезной работы или развлечения? Чем оно является по своей сути: рабочим приложением, развлекательной или вспомогательной программой (утилитой)?

Большинство приложений создается либо для работы, либо для решения вспомогательных задач, либо для развлечений. Одни приложения могут иметь интересно оформленный внешний вид, а другие — серьезный.

Большинство мобильных приложений можно отнести к одной из пяти категорий: интересно оформленные или серьезные рабочие приложения, интересно оформленные или серьезные приложения в режиме погружения и вспомогательные приложения.

---

[1] При всей важности сохранения максимально возможного постоянства и универсальности графического языка не следует полагаться на то, что пользователи знакомы со значками, характерными для той или иной операционной системы. Например, значок iOS, позволяющий поделиться информацией, пользователю другой операционной системы может показаться похожим на значок раскрытия окна на весь экран.

Эти типы приложений могут помочь вам уточнить свои дизайнерские решения. Хотя строгой схемы классификации, которой должно соответствовать все программное обеспечение мобильных устройств, не существует, эти рекомендации помогут вам создавать весьма эффективные приложения. Стилевое оформление приложения нужно выбирать в зависимости от степени его серьезности.

Инструментальные средства: рабочие приложения

Рабочие приложения, или инструментальные средства, используются для выполнения важных задач, например для чтения, набора и сортировки сообщений электронной почты. Успешные рабочие приложения или инструменты сосредотачивают внимание пользователей на выполняемой задаче, обеспечивают быстрое нахождение нужных функций и предоставляют пользовательский интерфейс, позволяющий быстро и просто выполнить необходимые пользователю задачи.

В рабочих приложениях прослеживается склонность к иерархической организации информации. Благодаря ей люди могут находить информацию путем постепенной конкретизации выбора, пока не достигнут желаемого уровня детализации, после чего выполняют задачи с информацией на этом уровне.

Рабочие приложения чаще всего используют несколько представлений, каждое из которых обычно отображает один из уровней иерархии. Пользовательский интерфейс должен быть простым, лаконичным и составленным из стандартных представлений и элементов управления. Основное внимание должно быть сосредоточено на информации и задаче, а не на окружении или восприятии.

Рабочие инструменты должны включать страницу предпочтений или настроек, определяемых пользователем. Эти настройки могут храниться на стороне сервера, или, как было сказано в главе 6, в `LocalStorage`. Рабочие приложения могут работать с большим объемом информации и потенциально иметь различные способы доступа к информации и работы с ней.

Исходные приложения смартфона позволяют получить доступ к инструментам его настройки, и вам, может быть, захочется заново создать такую же страницу настроек, чтобы запомнить предпочтения пользователя: сохраняйте настройки на локальном устройстве в `LocalStorage`, если приложение не требует идентификации пользователя, и на серверной стороне, если требует. Сохраняйте эту информацию в виде cookie-файла, если ее нужно передавать на сервер и обратно, или в `LocalStorage`, если этого делать не нужно.

Если добавляется страница настроек или предпочтений, нужно помнить, что необходимость доступа к ней и внесения изменений будет возникать довольно редко. Простые изменения конфигурации должны вноситься на экране основного пользовательского интерфейса. Изменения предпочтений должны быть выделены в отдельные экраны, доступные через ссылку в меню.

В *серьезных* рабочих программных средствах обычно используется палитра с ограниченным числом цветов, подобная сине-серой палитре исходных приложений iPhone. Серьезные программные средства концентрируются на данных. В них

нужно использовать стандартные средства навигации и стандартное оформление верхней и нижней частей экрана. Нужно создать в конструкции четкие разграничения и включить в нее блоки из взаимосвязанных данных и/или поведения.

ПРИМЕЧАНИЕ

Независимо от того, используется монохроматическая палитра или многоцветная, цвета нужно проверять на контраст. Это связано с тем, что значительная часть населения страдает различными уровнями цветовой слепоты.

К примерам серьезных рабочих программ можно отнести страницы настройки и/или учетных записей, такие как эмулируемое нами средство выбора языка iPhone, Dropbox и календарь Yahoo! для мобильных устройств. При разработке серьезных рабочих программ основное внимание уделяется содержимому, а не внешнему виду приложения, о чем весьма убедительно свидетельствует непривлекательный внешний вид различных мобильных версий приложений Yahoo! и Google.

Но наряду с серьезными рабочими программами есть и *интересно оформленные* рабочие программы. В этих программах нужно использовать умеренное количество цветов и графики. В них допускается менее высокая производительность, и они похожи на серьезные рабочие программы, но используют другую, более привлекательную цветовую схему. Интересно оформленные рабочие программы, как и серьезные программы, должны разрабатываться с прицелом на простую иерархию информации.

Развлечения: приложения в режиме погружения

Приложения в режиме погружения представляют собой полноэкранные визуально богатые среды, сконцентрированные на пользовательском впечатлении от содержимого приложения. Задачи, представляющие уникальную среду, минимизируют текстовую информацию и награждают пользователей за их внимание продуманным погружением.

К приложениям в режиме погружения относятся игры, аудиовизуальные материалы и другие развлечения. Под эту категорию подпадает и наша игра CubeeDoo. Внимание пользователя концентрируется на визуальном содержимом и впечатлении от него, а не на данных, создающих это впечатление. Хотя игры и входят в эту категорию, приложения в режиме погружения не обязательно являются играми.

В приложениях в режиме погружения наблюдается стремление скрыть основную часть пользовательского интерфейса, имеющегося на устройстве, заменив его специальным пользовательским интерфейсом, укрепляющим ощущения пользователя при вхождении в мир приложения. Некоторые мобильные браузеры позволяют скрывать или изменять панели управления и состояния браузера, чтобы усилить чувство погружения. Пользователи ожидают, что поиск и обнаружение будут частью их восприятия приложения в режиме погружения, поэтому использование нестандартных элементов управления может быть вполне приемлемым.

Приложения в режиме погружения могут работать с большим объемом данных, но обычно они их не показывают. Взамен эти приложения представляют информацию в контексте сценария компьютерной игры, сюжета или восприятия. Также

по этой причине приложения в режиме погружения зачастую предоставляют специальные навигационные методы, дополняющие среду, а не стандартные, управляемые данными методы, используемые в утилитах или рабочих приложениях. В приложении могут быть тонны информации, но пользователям не стоит ожидать ее последовательного просмотра или переходов через нее.

ПРИМЕЧАНИЕ

Поскольку в среде мобильных устройств большое неудобство доставляют ограниченная и/или дозированная пропускная способность и работа от аккумулятора, загрузка больших ресурсов должна быть последовательной, с четко обозначенным пользовательским разрешением или возможностью установки предпочтений.

Простые развлечения

Развлечения, к примеру игры, зачастую насыщены графикой. Это единственный тип веб-приложения, который может использовать звук без получения явного разрешения (путем нажатия кнопки проигрывания) от пользователя[1]. Теги `<audio>` и `<video>` рассматривались в главе 5. Функцию `play()` без специального запроса разрешения можно применять только в развлекательных веб-приложениях. Как и во всех приложениях любого типа, иерархия информации в них должна быть простой.

Серьезные развлечения

К серьезным можно отнести развлечения, имеющие определенную цель. В отличие от простых развлечений, в которых может присутствовать стремление добиться наивысшего счета, серьезные развлечения позволяют что-то сделать. К серьезным развлечениям можно отнести iTunes Store, Netflix, Flickr, YouTube и другие приложения для просмотра фото- и видеоматериала и его выкладывания в Сеть. Серьезные развлечения должны включать умеренное использование графики. Вместо полной концентрации на графике, как в играх, серьезные развлечения концентрируются на содержимом. В эти приложения часто включаются данные на вкладках. Для этого типа приложений может иметь смысл наличие стандартных элементов навигации как в верхней, так и в нижней части экрана.

Утилиты

Последняя категория приложений является единственной, которая служит одной цели, сразу же выдавая всю необходимую пользователю информацию. Утилиты являются графически богатыми одноэкранными приложениями с небольшой или вовсе отсутствующей иерархией (что не требует углубляться в данные в поиске нужной информации), в качестве примера можно привести приложения, информирующие о погоде или биржевых котировках.

Приложения-утилиты отображают узконаправленный объем информации с четко выраженными суммарными данными. Конечно, есть приложения, имеющие

[1] Почему? Во-первых, потому, что я так сказала, а во-вторых, потому, что вам не нужны плохие отзывы или ненавистники.

более одного экрана, но вспомогательные экраны служат для установки пользовательских предпочтений. Пользовательские настройки могут сохраняться на локальном устройстве с помощью `LocalStorage` (см. главу 6). А при наличии механизма идентификации можно задать и хранение настроек на серверной стороне.

Приложения-утилиты обычно ориентированы скорее на быстрый взгляд, чем на взаимодействие с ними. Состояние погоды можно проверять на смартфоне по пять раз в день, а курсы акций на бирже — и по 100 раз. Путешествуя, вы разве что меняете место своего нахождения. А если вы не трейдер, то что-то менять в своих акциях можете значительно реже.

Приложения-утилиты включают информационную кнопку (обычно находится в нижнем правом углу), с помощью которой открывается дополнительный экран, где могут быть обновлены настройки. Не следует забывать и о кнопке Готово (Done) на втором экране, предназначенной для возвращения на главный экран. В гибридных приложениях частично контролировать расход энергии батареи позволит возможность задания пользователями частоты обновления приложения.

Приложения-утилиты могут быть визуально привлекательными, но, как и во всех других приложениях, информация, представляемая при этом, должна не затмеваться, а выделяться дизайном. Пользовательский интерфейс должен быть лаконичным, с простыми стандартными представлениями и элементами управления.

А что подойдет конкретно вам?

Прочитав о стилях рабочих приложений, утилит и приложений в режиме погружения, подумайте о типе той информации, которая отображается в вашем приложении, и о задаче, решаемой с его помощью. Перед принятием решения о стиле нужно определить, чем именно будет занято ваше приложение.

Вы можете подумать, что с типом будущего приложения уже все ясно и можно приступать к его созданию, но сделайте шаг назад. Обычно все не так просто. Вы не ограничены выбором лишь одного стиля приложения. Может обнаружиться, что в соответствии с замыслом приложения наибольший смысл имеет сочетание характеристик из нескольких различных стилей приложений. Независимо от выбранных действий не нужно ничего усложнять. Сведите перечень свойств к минимуму и создайте приложение, которое способно на что-то одно. Проследите за тем, как оно используется и какова реакция на приложение, и переделайте его на основе этих наблюдений.

Если есть уже существующее компьютерное приложение, не занимайтесь простым переносом его на мобильную веб-основу. Использование приложений для телефонов отличается от использования приложений для настольных компьютеров и ноутбуков. Кроме того, опыт показывает, что у людей при пользовании разными устройствами разные ожидания. Подумайте, как они могли бы взаимодействовать с приложением на небольшом экране мобильного устройства. Не удаляйте какие-либо функции из варианта приложения для настольного компьютера. Вместо этого при наличии функций, определенно более приоритетных для пользователей мобильных устройств, выведите эти функции на первый план.

ПРИМЕЧАНИЕ

Выведите на первый план те функции, которые, вероятнее всего, будут более приоритетными для пользователей мобильных устройств.

Примените к разработке приложения правило «80 — 20». Предположите, что большинство пользователей (как минимум 80%) будут использовать весьма ограниченное количество функций, имеющихся в приложении, и только немногие (не более 20%) станут применять все функции. Затем нужно тщательно продумать, стоит ли заранее загружать мобильное приложение с объемными функциями, которые требуются небольшому проценту пользователей, или лучше сделать эти функции доступными только по требованию (я рекомендую второй вариант). Имейте в виду, что у приложения для настольного компьютера может быть более подходящая среда окружения, в которой можно открыто предложить те или иные функции, и, как правило, в мобильном приложении лучше сосредоточиться на функциях, соответствующих запросам наибольшего числа людей.

Конечно, нужно сделать все функции, имеющиеся в настольном приложении, доступными и в мобильном приложении. Только вот заранее загружать их не стоит. Все больше пользователей получают доступ к Интернету только через мобильные устройства. Им нужно предоставить возможность делать то же самое, что доступно пользователям настольных компьютеров. Нужно обеспечить быстрый поиск и практичность, но они не должны занимать центральное место в начальной загрузке. Их нужно загружать только по необходимости, помня об управлении электропитанием, памяти и пропускной способности. Если в этом есть смысл для вашего приложения, можно позволить пользователю изменять настройки, чтобы могли появляться менее востребованные на мобильном устройстве функции.

ПРИМЕЧАНИЕ

Возрастающее меньшинство в США и приличное большинство на некоторых развивающихся рынках пользуется только мобильными устройствами.

Мобильная платформа: богатые возможности

Планируя создание мобильного веб-приложения, нужно брать в расчет характеристики дизайна и устройства. Приложения для смартфонов отличаются от приложений для настольного компьютера, хотя мобильная версия может быть (или будет) наилучшей отправной точкой для разработки сопутствующего сайта-компаньона для настольных систем. И об этом важно помнить при разработке веб-приложения для небольшого экрана.

Небольшой экран

Несмотря на то что у мобильных устройств может быть весьма высокое разрешение экрана, сами экраны имеют все же очень небольшие размеры. Учитывая это, важно

сконцентрировать пользовательский интерфейс на самом важном. Не стоит включать в него те элементы, в которых нет необходимости. Научитесь говорить «нет» своим клиентам, кем бы они ни были — руководителями, акционерами или просто вашим внутренним голосом. Перенасыщенный элементами управления пользовательский интерфейс делает сайты для настольных компьютеров и приложения запутанными и непривлекательными. Еще более актуальной эта проблема становится для смартфонов, у которых площадь экрана составляет менее 25% от площади экрана настольного компьютера. Умение сказать «нет» функциям при конструировании и разработке мобильного интерфейса поможет получить менее захламленную версию и для настольных компьютеров.

Меньший объем памяти

На настольном компьютере может быть от 4 до 16 Гбайт оперативной памяти, а на iPad вашего пользователя памяти такого объема может и не быть. На моем iPad 256 Мбайт оперативной памяти. А на Android-телефоне, являющемся одним из самых мощных моих мобильных устройств, — 768 Мбайт.

Следует заметить, что наши веб-приложения запускаются в браузере и их нельзя отнести к самостоятельным приложениям, коим является браузер. Память потребляют и запущенный браузер, и запущенное веб-приложение, и запущенная на мобильном устройстве операционная система. Следует помнить, что операционная система, браузер и веб-приложение — не единственные приложения, которые могут быть запущены на устройстве. И все эти приложения, которые вы можете и не контролировать, используют память устройства. А кроме этого, потребляют энергию аккумулятора. Чем больше процессов, загружающих оперативную память и микросхемы устройства, тем выше нагрузка на устройство и быстрее разряжается его аккумулятор.

Когда я приобрела телефон HTC с 768 Мбайт, у него в исходном состоянии доступны были лишь 222 Мбайт оперативной памяти, поскольку остальную память занимало программное обеспечение, запущенное на нем по умолчанию. Такой свободный объем памяти нельзя назвать большим. И когда у телефона заканчивалась свободная память, ничего нельзя было поделать. Большинство мобильных устройств не способно перераспределять память. Когда браузеру не хватает памяти, он входит в аварийный режим.

Хотя вам мало что удастся сделать, чтобы возложить на пользовательские браузеры обязанность своевременно предупреждать о недостаточном объеме памяти и очищать ее, более важной задачей является предупреждение утечек памяти в коде при ограниченных возможностях центрального процессора.

Вам следует обеспечить наименьший возможный объем ваших файлов с ресурсами: не загружайте те ресурсы в которых нет необходимости, и подумайте, нужны ли вам потребляющие память малоизвестные пользователю функции.

В предыдущих главах при представлении функций, которые могут потреблять большие объемы памяти или надолго загружать центральный процессор, таких как радиальные градиенты, внутренние тени и изображения размерами свыше

1024 пикселов, их недостатки уже рассматривались. Поставщики браузеров постоянно совершенствуют возможности и повышают производительность своих продуктов. Вскоре радиальные градиенты, внутренние тени и большие изображения смогут более экономно расходовать память телефона. Ссылки на источники, в которых перечислены некоторые узкие места ныне существующих мобильных браузеров, можно найти в онлайн-ресурсах к главе.

Управление памятью

Ограниченность ресурсов памяти нужно учитывать в ходе всего процесса разработки. Мы все занимаемся разработкой веб-приложений на настольных компьютерах, не испытывающих недостатка в оперативной памяти, поэтому можем просто забыть о существовании данной проблемы, а этого делать не следует!

При ведении разработки на настольном компьютере обращайте внимание на потребление памяти вашим веб-приложением. Потребление памяти и производительность можно отслеживать в браузерах Chrome (рис. 12.1).

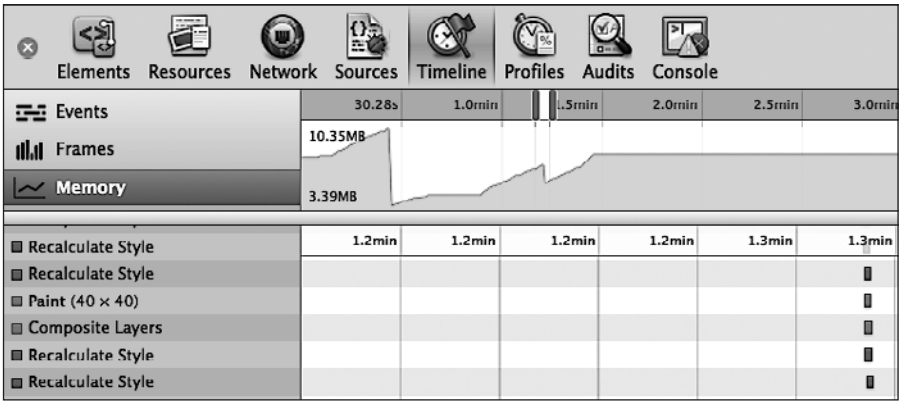


Рис. 12.1. Показатель потребления памяти в инструментарии разработчика в браузере Chrome

Chrome представляет информацию о том, какой объем памяти используется каждой вкладкой браузера. Чтобы просмотреть сведения об объемах используемой памяти, нужно открыть инструментарий разработчика в браузере Chrome для настольных компьютеров (View ▸ Developer ▸ Developer Tools (Вид ▸ Разработчик ▸ Инструменты разработчика)). Выберите вкладку Timeline (Временная шкала), затем выберите в окне просмотра Timeline вкладку Memory (Память) (см. рис. 12.1)[1]. Чтобы оценить расход памяти, нужно выбрать кнопку записи (серый кружок в нижней части Developer Tools (Инструменты разработчика)). Когда кнопка записи становится красной, это означает, что идет запись. Поработайте в своем приложении.

[1] Подобные инструменты имеются также в Safari Developer Tools, Opera DragonFly, Firebug the Firefox add-on и в IE F12, начиная с IE11.

Понаблюдайте, как растет расход памяти по мере добавления DOM-узлов или ресурсов и как он падает после сбора «мусора».

Не забудьте о том, что, кроме вашего великолепного веб-приложения, которое могло бы использовать 80 Мбайт, есть еще и другие программы, запущенные на устройстве пользователя. На нем работают операционная система, браузер, программное обеспечение телефона. Есть и другие приложения, работающие в фоновом режиме. И все эти приложения расходуют память.

В ходе разработки для тестирования будут использоваться инструменты настольного компьютера. А при тестировании устройства будут также работать приложения, характерные для его операционной системы, которые станут оценивать расход памяти и энергии батареи, связанный с работой приложения и использованием мобильной сети. Эти инструменты рассматриваются в главе 14.

Хотя свойство Timeline Memory в инструментарии разработчика может помочь определить объем памяти, расходуемый приложением, вам все же нужно протестировать это приложение на различных устройствах. У этих устройств должен быть ряд собственных приложений, установленных с включенным режимом уведомлений, поскольку именно в таких условиях у пользователей будет формироваться впечатление о вашем веб-приложении.

Как разработчику вам следует учитывать задержки при работе, использование памяти, расход энергии батареи и задействование трафика вашими сайтами. Не забудьте протестировать их в мобильных сетях, которые обычно не столь быстры, как Wi-Fi. Более подробно тема производительности будет рассмотрена в главе 14.

Одно окно — одно приложение

Мобильные пользователи одновременно могут видеть только одно окно браузера. На большинстве мобильных платформ, даже если веб-приложение содержит несколько экранов, пользователи будут видеть их последовательно, а не одновременно.

Хотя мобильное веб-приложение должно обеспечить доступ ко всему набору функций полноценного приложения, вместо одновременного воспроизведения широкого набора функций сконцентрируйтесь единовременно на одной задаче. В условиях ограниченных объемов памяти, пространства и возможности последовательного просмотра страниц представьте наиболее важную информацию на главной странице, предложив получать доступ к дополнительным функциям только по мере необходимости посредством дополнительных экранов, или кнопки More (Дополнительно), или i-кнопок со ссылками на расширенный набор функций.

При взаимодействии с большинством смартфонов (для некоторых планшетных компьютеров это менее актуально) на переднем плане единовременно можно увидеть только одно приложение. Когда пользователи переключаются с одного приложения на другое, приложения иногда завершают свою работу полностью, а иногда продолжают ее в фоновом режиме.

Некоторые устройства поддерживают многозадачность: приложение, утрачивающее фокус, переходит в фоновый режим и остается в этом режиме, пока не будет запущено снова или остановлено. Это означает, что при щелчке пользователя на ссылке на YouTube, телефонном номере или карте Googl, работа браузера, а стало быть, и всего веб-приложения может быть завершена или, что бывает чаще всего, продолжится в виде фонового процесса.

Поскольку вам неизвестно, работает пользователь на устройстве, допускающем многозадачность, или нет, нельзя выстроить предположение о том, будет завершена работа браузера и веб-приложения или нет.

Минимум документации

Как и большинство из людей, я заглядываю в инструкции IKEA только после сборки мебели, удивляясь, откуда взялись три лишние детали. Вероятность того, что я удосужусь прочитать справку или другие инструкции, касающиеся моего телефона, еще меньше, чем та, что я стану пересматривать печатное руководство по сборке мебели, которую только что сломала.

Вот и пользователи не станут читать инструкции или справочную документацию перед использованием приложения. Чтобы приложение работало успешно, оно должно быть простым в использовании, со сразу же заметными необходимыми пользователю функциями, а ожидания пользователя относительно совпадения его пользовательского опыта с возможностями, предоставляемыми приложением, должны быть полностью оправданны. И эти утверждения справедливы независимо от используемого устройства.

Используйте стандартные элементы управления. Предоставляйте информацию логичным и предсказуемым способом. Обеспечьте в приложении такой же очевидный путь назад, как и путь вперед. Выберите такую конструкцию приложения, чтобы его поведение было последовательным и предсказуемым.

Даже если вы работаете для дизайнерской фирмы, а точнее, если вы работаете для дизайнерской фирмы, не пытайтесь использовать инновационные подходы к способам взаимодействия с приложением и к пользовательскому интерфейсу: основой для практических решений и дизайна должен послужить стандартный пользовательский опыт. Тогда приложение станет интуитивно понятнее, а такие приложения чаще пользуются успехом. Если от вас требуются инновации (и даже если нет), приложение нужно тестировать, тестировать и еще раз тестировать на разных пользователях. Инновации пользовательского интерфейса (UI) и взаимодействия с пользователем (UX) могут быть удачными и даже весьма удачными, если все переходы основательно протестированы на пользователях с опытом работы на широком диапазоне мобильных устройств.

Вопросы разработки

Не нужно забывать, что пользователи могут работать в мобильном режиме. У них должна быть возможность загружать веб-приложение быстро и тут же видеть наиболее важное содержимое в просматриваемом окне браузера, независимо от того,

в каком режиме ориентации оно находится, в книжном или альбомном. Это не значит, что нужно запретить прокрутку. Скорее, нужно сделать самую важную информацию и главный призыв к действию видимыми без прокрутки.

Пользователи должны получить возможность достижения своих целей с помощью всего нескольких движений пальцами. Не забывайте о том, что вашим пользователям могут быть недоступны мышь, колесо прокрутки или клавиатура для навигации по приложению и, чтобы выполнять переходы при взаимодействии с сайтом, они могут использовать только один или два пальца. Задействуя только одну руку, поскольку другой нужно будет держать устройство, они, может быть, вообще не будут пользоваться пальцами. Возможно, для навигации они воспользуются голосовыми командами, которые доступны не только в Google Glass.

Пользовательский интерфейс должен быть сконцентрирован на предоставлении нужных категорий и облегчении выполнения общих задач без запроса подробностей, не занимающих центральное место в задаче.

Сведите список функций в одну инструкцию, которая определяет характер конечного продукта и не только дает описание решения, предлагаемого вашим продуктом, но и определяет целевой рынок. Не допускайте выхода основных функций за пределы этого списка.

Вы можете и должны включить в приложение редко востребованные функции, но их нужно сделать второстепенными. Контекстом всегда должны быть те действия, которых ожидают от пользователя, с прицелом на определенную вами пользовательскую аудиторию.

Если функция не является основной для достижения целей, поставленных пользователем, ее не нужно включать в целевую страницу веб-приложения: скажите «нет» командам, занятым маркетингом и продажами. Пространства, доступные в мобильных телефонах, слишком малы, скорость загрузки слишком низка, а расширять взаимодействие с пользователем в вашем скромном приложении слишком сложно. В пространстве мобильного телефона нет места для функций, не предназначенных для решения основной задачи. А от разработки, ведущейся в первую очередь для мобильных устройств, версия приложения для настольных компьютеров только выиграет.

Очертив аудиторию пользователей, вы сможете уточнить определение продукта приложения, удовлетворяющего основные потребности этой аудитории. Уточнение определения продукта позволит лучше отфильтровать список функций. Нужно будет убрать те функции, пусть в чем-то весьма полезные, которые не вписываются в определение продукта приложения. Сделайте одно дело. Но сделайте его хорошо. Перенесите эту логику на разработку приложений для настольного компьютера: простота понравится пользователям и на этом устройстве!

Разработка с прицелом на мобильные WebKit-браузеры

Чтобы создать веб-приложение, похожее на стандартные мобильные приложения, следует учесть ряд особенностей. Существуют взаимоотношения между тегами

<meta> и <link>, являющиеся фирменной принадлежностью компании Apple, но некоторые из них работают и на Android, Chrome для Android, BlackBerry 10 и iOS. Эти взаимоотношения тегов <meta> и <link> уже рассматривались в главе 2, но давайте взглянем на них еще раз.

В iOS можно указать браузеру на то, что приложение должно быть автономным:

```
<meta name="apple-mobile-web-app-capable" content="yes"/>
```

Это указание сработает только в том случае, если пользователь сохранил веб-приложение на главном экране в виде закладки со значком и обратился к веб-приложению с помощью этой закладки, а не перешел к нему, воспользовавшись адресной строкой браузера. Хотя это обстоятельство может показаться неким ограничением, благодаря ему ваше приложение, будучи в действительности веб-приложением, получит возможность работать в полноэкранном режиме, ничем не отличаясь от стандартного приложения. Использование этого метатега приводит в iOS WebKit-браузерах к удалению стандартных элементов навигации и управления, когда пользователь обращается к вашему веб-приложению через сохраненную закладку.

Хотя браузеру указывается на то, что приложение должно выглядеть как стандартное, но со своими собственными функциями, ему все же нужно указать, в чем именно состоят эти присущие приложению функции. Когда приложение запускается в автономном режиме, у вас есть возможность управлять цветом панели состояния и навигационной панели.

Панель состояния

На панели состояния отображается важная информация о пользовательском устройстве, включая мощность сигнала, состояние сетевого подключения и степень разряженности батареи. Когда ваш сайт или приложение просматривают в браузере мобильного устройства, вы не можете скрыть панель состояния, имеющую высоту 20 пикселов (рис. 12.2).

Рис. 12.2. Панель состояния iPhone

Возможность скрыть панель состояния не должна становиться решающим фактором при принятии решения о том, какое приложение нужно создавать — стандартное или веб-приложение. Хотя некоторые устройства позволяют скрыть панель состояния при разработке стандартного приложения, этого делать не следует. Пользователю вряд ли понравится то, что посмотреть на остаток заряда батареи или наличие соединения они смогут, только завершив работу приложения.

В iOS можно, отказавшись от исходного цвета, изменить цвет панели состояния (см. рис. 12.2) на черный или полупрозрачный черный. Настроив приложение на запуск в полноэкранном режиме и задав значение метатега default,

black или black-translucent, получим отображение панели состояния в сером (исходном) цвете, непрозрачном черном цвете или полупрозрачном черном цвете (rgba(0,0,0,0.5)):

```
<meta name="apple-mobile-web-app-capable" content="yes"/>
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Этот метатег будет работать только в том случае, если страница будет настроена на web-app-capable.

Когда для атрибута content метатега apple-mobile-web-app-capable установлено значение yes, веб-приложение запускается в полноэкранном режиме, в противном случае этого не происходит. Но эта настройка не действует, если пользователь переходит на ваш сайт через браузер. Поведение, похожее на то, что присуще стандартному приложению, проявляется только при обращении к нему как к стандартному приложению.

Значение black-translucent делает панель состояния прозрачной на 50% и располагает ее над содержимым веб-страницы или приложения, освобождая немного дополнительного экранного пространства, которое пригодится в том случае, если ваше приложение представляет собой игру вроде «Тетриса», где элементы появляются в верхней части экрана.

Мы включили эти метатеги в CubeeDoo. Но результаты этого вам могут быть не видны. Чтобы увидеть эту функцию в действии, нужно добавить CubeeDoo в виде ссылки на закладку на главный экран устройства, которое поддерживает данную функцию, и обратиться к веб-приложению, щелкнув на этом значке.

Панель навигации

Панель навигации представляет собой адресную панель, появляющуюся в верхней части экрана, сразу под панелью состояния (рис. 12.3). По умолчанию некоторые мобильные браузеры, например Safari на iOS и браузер Firefox OS, показывают содержимое тега <title> вашей веб-страницы вместе с поисковой и адресной панелями на панели навигации высотой 60 пикселов. Браузер Chrome для мобильных устройств показывает URL-адрес, инструменты, содержащие ссылки на дополнительную информацию о странице, другие открытые вкладки и раскрывающуюся панель инструментов. Chrome для более крупных мобильных устройств, таких как Nexus Galaxy, отображает такие же вкладки, как и браузер настольного компьютера.

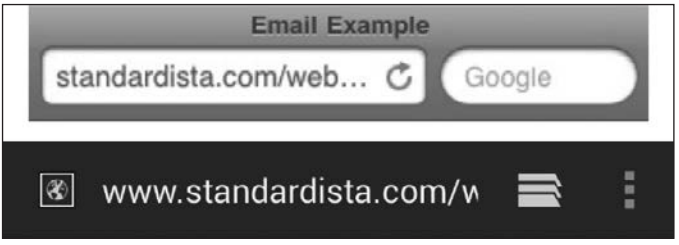


Рис. 12.3. Панели навигации Safari *(вверху)* и Android Chrome *(внизу)*

Чтобы создать веб-приложение, имеющее внешний вид как у стандартного приложения, можно скрыть исходную панель навигации и добавить панель навигации самого приложения. В примере, приведенном в главе 7, для эмуляции исходной стандартной панели навигации iPhone (рис. 12.4) использовались средства CSS.

Рис. 12.4. Навигационная панель, эмулирующая исходную панель навигации стандартного приложения iPhone с помощью средств CSS

Применив небольшую хитрость, можно скрыть большую навигационную панель Safari, даже если пользователь не сохранил закладку на ваше приложение на экране. Для того чтобы скрыть навигационную панель, нужно воспользоваться запасным методом с использованием JavaScript.

Чтобы скрыть навигационную панель Safari[1], нужно включить в веб-приложение строку кода `window.scrollTo(0, 1);`. Следующий сценарий скроет навигационную панель Safari при загрузке страницы:

```
<script>
addEventListener("load", function() {
  setTimeout(hideURLbar, 0);
  }, false);

function hideURLbar() {
  window.scrollTo(0,1);
}
</script>
```

Взаимодействие с пользователем (UX): панели навигации

При создании панели навигации, эмулирующей внешний вид такой же панели стандартного приложения, на исходном или главном представлении должен быть показан только заголовок приложения, поскольку пользователю навигация по приложению пока не нужна. Если в приложении имеется только одна страница, панель навигации главной страницы может состоять только из элементов управления содержимым в представлении. Если в приложении несколько страниц, на всех последующих экранах должен быть новый заголовок с кнопкой возврата, на которой написан заголовок предыдущего экрана, или со словом «Назад» слева от него.

Кнопка Назад используется в качестве стандартного способа возврата к предыдущему экрану. Это соответствует ожиданиям пользователя, поэтому без веских

[1] iOS 7 не позволяет скрывать навигационную панель.

причин ничего менять не стоит. Не забывайте, что отчаявшийся пользователь может оказаться вынужден воспользоваться кнопкой перехода на главную страницу и может не вернуться в приложение. На панели навигации справа от заголовка может находиться еще и вторая кнопка, управляющая содержимым в представлении.

В продуктах Apple предусматривается стандартизация кнопок. Как показано на рис. 12.4, кнопки на панели навигации окружены желобком. Все элементы управления Apple iOS, имеющиеся на панели навигации, используют стили с заданием границ. Все значки пользовательского интерфейса iOS имеют размер 30 × 30 пикселов, для значков панелей вкладок с областью, чувствительной к прикосновениям, используется размер 44 × 44 пиксела. Значки панели инструментов и навигационной панели Apple выполняются размером 20 × 20 пикселов.

Дизайнерские шаблоны Android рекомендуют шаблон 48-пиксельной независимой от устройства периодичности (48 DP), который соответствует приблизительно 9 мм (0,35 дюйма) с некоторыми вариациями, обеспечивая наличие областей, чувствительных к прикосновениям, в диапазоне рекомендованных 7–10 мм целевого размера.

Элементы дизайна, имеющие как минимум 48 DP в высоту и в ширину, гарантируют, что независимо от размеров экрана размер целей будет не меньше, чем минимальный рекомендуемый целевой размер 7 мм (48 DP предусмотрены для хорошей общей плотности информации и возможности нацеливания на элементы пользовательского интерфейса). Промежутки между элементами пользовательского интерфейса должны составлять 8 DP.

Элементы управления страницы. Элементы управления страницы должны находиться на панели высотой 44–48 пикселов, идущей по нижней части экрана, где слева направо должны располагаться элементы управления по степени убывания их востребованности. Не нужно создавать области для применения жестов прикосновений слишком близко к краям окна браузера, поскольку на некоторых мобильных устройствах станут фиксироваться жесты для устройства или для стандартных действий браузера.

Не размещайте на нижней панели те элементы управления, на которых пользователь может случайно щелкнуть вопреки вашему желанию. Нижняя панель должна использоваться для наиболее частых действий пользователя. Включайте те элементы, щелчки на которых выполняются редко, например относящиеся к настройкам или удалению, где-нибудь в другом месте, например на верхней панели навигации или на отдельном экране.

В CubeeDoo элементы управления страницей, даже те, которые в соответствии с нашими ожиданиями будут часто востребованы, мы помещаем в верхней части экрана. Почему? Потому что это веб-приложение является игрой. 99% времени будет занято игрой, а не проверкой наивысших показателей или других функций меню. Пользователи вряд ли обрадуются, случайно щелкнув на каком-нибудь элементе управления в ходе игры. Поэтому, несмотря на то что многим приложениям подходит именно нижняя панель управления, не стоит строго придерживаться этого правила. Единственным строгим правилом является применение

здравого смысла (ну и обеспечение предпочтительного размера кнопок 44 × 44 пиксела, как минимум 22 × 22 пиксела, и областей взаимодействия, если нужно, чтобы пользователь успешно куда-нибудь попал).

Размер и цвет панели навигации

Изменение ориентации устройства с книжной на альбомную может автоматически изменить высоту панели управления (программно высоту задавать не нужно). При альбомной ориентации более тонкая панель навигации предоставляет больше места для содержимого вашего экрана. При разработке значков на панели навигации и разметки экрана не забудьте принять во внимание разницу в высотах.

Стремитесь сохранить последовательность во внешнем виде панелей навигации и других панелей вашего приложения. Если используется полупрозрачная панель навигации, то ее не нужно комбинировать с непрозрачной панелью инструментов. Также следует избегать изменений цвета или прозрачности навигационной панели на разных экранах в одной и той же ориентации.

Загрузочное изображение

Если пользователь щелкает на значке главного экрана, веб-приложение немедленно запускается, возможно еще до того, как с сервера будут получены все файлы. В некоторых браузерах, включая стандартные Safari-браузеры iOS, можно управлять тем, что показывают браузеры, пока ожидают загрузки сайта, предоставляя загрузочное изображение.

Браузер можно заставить показать конкретное изображение, пока он ожидает загрузки, проводит разбор кода и выкладывает все ваши объекты:

```
<link rel="apple-touch-startup-image" href="/screenshot.jpg"/>
```

Здесь приводится URL-адрес, указывающий на загрузочное изображение. По умолчанию используется копия экрана с последнего запуска веб-приложения, но с помощью данного тега можно определить свое собственное изображение.

Однако это не должна быть заставка. Ее лучше не использовать. Вашим пользователям хочется получить содержимое. Они не хотят ждать (расплачиваясь при этом за пропускную способность канала) только потому, что вашей маркетинговой команде или генеральному директору нравятся вступления в стиле флеш-анимации.

Значки главного экрана

```
<link rel="apple-touch-icon"…
```

Это указатель на изображение, которое мы собираемся сделать значком сайта, расположенным на главном экране устройства и обозначающим закладку, ведущую на сайт. Но не все устройства требуют одинакового размера или разрешения изображения. Различия могут быть даже среди устройств от одного производителя. Как уже говорилось в главе 2, для тега `<link>` у нас есть новый атрибут, помогающий справиться с этой ситуацией:

```
<link rel="apple-touch-icon" href="touch-icon-iphone.png" />
<link rel="apple-touch-icon" sizes="72x72"
  href="touch-icon-ipad.png" />
<link rel="apple-touch-icon" sizes="114x114"
  href="touch-icon-hiresolution.png" />
```

Устройство превратит ваш значок в более подходящий для той или иной операционной системы. На iOS 6 и более ранних версий к нему добавятся скругленные углы и подсветка. Если нужно создать собственные углы и подсветку или убрать подсветку, следует воспользоваться в значении для атрибута `rel` ключевым словом `precomposed`:

```
<link rel="apple-touch-icon-precomposed" href="path/image.png"/>
```

Потратьте время на создание красивого значка. Пользователи должны быть в состоянии с первого взгляда на ваш значок определить, чем занимается сайт или приложение. И советую: если вы не представляете хорошо известную компанию с узнаваемым брендом (например, CNN), не включайте в значок текст.

Исходный значок iPhone (до выхода iOS 7) имеет размер 57 × 57 пикселов с закруглением границы 10 пикселов. Размер значка iPad составляет 74 × 74 пиксела с радиусом 12 пикселов. Размер значка для iTunes Store — 512 × 512 пикселов.

Если вы на самом деле надеетесь попасть в App Store, нужно учесть следующее: в компании Apple любят фоновые изображения с иллюзией осязаемости, тонкие тени, выделенный текст, глянцевые кнопки, едва различимые градиенты и качественные яркие значки. Отправите вы приложение в App Store или нет, не стоит в каких-либо приложениях использовать значки, группы изображений или торговые марки Apple.

Сведите к минимуму ввод с клавиатуры

Ввод данных является весьма затратной операцией на всех устройствах, тем более на устройствах с сенсорным вводом. Если требовать от пользователя ввода данных, то нужно быть уверенными в том, что на это стоит потратить время. Разумеется, иногда без информации не обойтись. Если команда по маркетингу проявляет инициативу и требует пользовательский почтовый код, когда он совершенно не нужен, скажите «нет». Не ставьте барьеры на вход, в которых нет абсолютной необходимости. Это важно как для мобильных, так и для настольных устройств!

Когда требуется ввод данных, его нужно максимально упростить. Если это не запрещено по соображениям безопасности, то требуемые имя пользователя и пароль нужно запомнить как информацию пользователя!

Если пользователь должен ввести данные, то по возможности создайте список выбора или другую форму отбора, чтобы для представления информации он мог использовать любой метод, кроме ввода с клавиатуры. Если пользователю нужно ввести телефонный номер или адрес электронной почты, нужно использовать надлежащие типы ввода (см. главу 4), чтобы он увидел клавиатуру, соответствующую вводимым данным.

Обычно в формах перед полями ввода помещаются надписи, а после них — подсказки. На устройствах с экранами более скромных размеров нужно обеспечить их переносы. Или, что еще лучше, поместить надписи над областью ввода, а подсказки — под этой областью. Когда устройства выводят виртуальную клавиатуру, на многих из них одновременно увеличивается то поле формы, на котором находится фокус, при этом на узких экранах оказывается заслонено все, что находится слева и справа от этого поля.

Проявляйте краткость во всем

Этим все сказано. Будьте краткими.

Пользователи всматриваются в мелкие шрифты на небольших устройствах. Но даже на настольных компьютерах пользователи не любят читать. Излагайте мысль по существу. Проявляйте краткость.

И в качестве краткого подведения итогов.

- **Старайтесь все сделать очевидным.** У пользователей вечно не хватает времени или внимания, чтобы разобраться в сложных приемах взаимодействия и приложениях. Нужно сделать приложение сразу же понятным для пользователей.
- **Сведите к минимуму ввод данных.** Ввод информации требует от пользователей времени и внимания. Если ваше приложение, перед тем как принести пользу, потребует большого объема пользовательского ввода, то пользователи, скорее всего, предпочтут перейти на другой сайт или поработать с другим приложением.
- **Сведите к минимуму количество текста.** Если текст в пользовательском интерфейсе краток и конкретен, пользователи могут усвоить его быстро и легко. Наиболее важную информацию нужно выделить, лаконично выразить и поместить на видном месте.

Другие вопросы взаимодействия с пользователем

Избегайте скрытого содержимого, которое становится видимым (и, возможно, закрыто ладонью пользователя) только при проходе указателя мыши или прикосновении. Пальцы — слишком толстый инструмент. Ладони слишком большие. Когда информация демонстрируется только между началом и концом прикосновения или только при проходе указателя мыши, она может оказаться не полностью доступной для тех, кто пользуется сенсорным устройством.

Существуют определенные соглашения, вредные для настольных систем, но еще более вредные для мобильных устройств. Меню с подменю, которые раскрываются при проходе над ними указателя мыши, плохо подходят для настольных компьютеров, но еще меньше — для мобильных устройств.

Когда от пользователя требуется использовать для навигации по сайту раскрывающееся или появляющееся при проходе указателя мыши меню, от него требуется находиться в верхней части меню, не покидая его или не уводя в сторону указатель, пока не будет достигнут нужный пункт. Но на сенсорных устройствах пользователи не направляют в нужное место указатель мыши — они прикасаются к экрану. На некоторых устройствах код CSS, показанный в блоке `:hover`, будет работать, а на других для этих меню придется разрабатывать сценарии обработки прикосновений, чтобы все устройства отображали меню и не позволяли внезапно прятаться подменю.

На рис. 12.5 можно заметить, что самый короткий путь перехода от Dropdown до Pick Me проходит за пределами появляющегося меню, что вызовет его закрытие. И это в том случае, если толстый палец пользователя или его ладонь не закрывают содержимое страницы. Даже при использовании мыши, когда содержимое видно лучше всего, прямая линия между двумя ссылками частично проходит за пределами меню, а выход за пределы меню навигации обычно приводит к тому, что подменю скрывается.

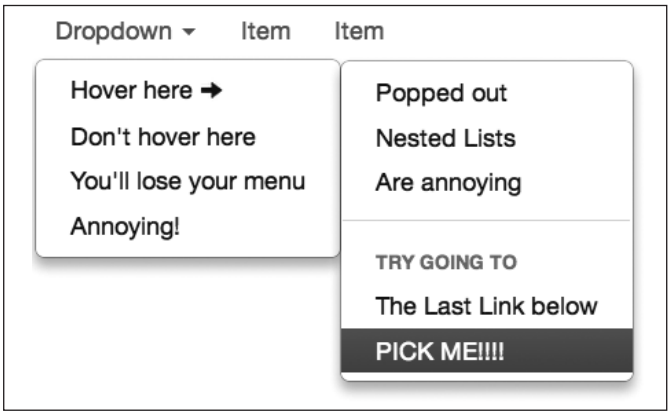


Рис. 12.5. Пример обычного раскрывающегося меню

Держитесь подальше от раскрывающихся и всплывающих меню, отдавая предпочтение тем меню, которые работают от прикосновений или щелчков. Кроме того, при слишком длинных списках или элементах навигации лучше использовать динамически загружаемое дополнительное содержимое. Кроме того, неплохо было бы избавиться от лишнего содержимого, выходящего за рамки видимого на данный момент списка, особенно если это связано с бесконечной прокруткой[1], чтобы сократить потребление памяти и количество DOM-узлов, замедляющих перерисовку данных.

Принципы взаимодействия с пользователем, рассмотренные в данной главе, актуальны не только для мобильных сайтов и приложений, но и для сайтов, предназначенных для настольных компьютеров и стандартных мобильных приложений.

[1] Положение может улучшиться, если применить предлагаемую компанией Google «ленивую» загрузку блоков.

13 Ориентация на мобильные устройства и сенсоры

Надеюсь, вы уже поняли, что разметка для мобильных браузеров имеет тот же код, что и для браузеров настольных компьютеров. Основным различием является размер окна просмотра и способ взаимодействия пользователей с теми устройствами, на которых они работают. На настольных устройствах используются клавиатура и мышь, большой экран и браузер, размеры окна которого можно изменять. На сенсорных устройствах мы используем наши пухлые пальчики, иногда на небольших экранах с окнами просмотра, размер которых, как правило, изменить нельзя.

Но это всего лишь обобщения! У меня есть настольный компьютер с сенсорным экраном размером 23 дюйма. И есть планшетный компьютер с внешней клавиатурой и мышью, связанными с ним по Bluetooth. Все веб-содержимое должно быть доступно с помощью сенсора и мыши как на больших мониторах, так и на маленьких экранах. Занимаясь разработкой программ, нужно помнить, что не все получают доступ к создаваемому нами содержимому одинаковым способом.

Масштабирование до нужного размера

Когда дело касается совсем небольших окон просмотра, хочется, чтобы ширина сайта была равна ширине экрана устройства. Исходный размер отображения страницы для большинства мобильных браузеров равен 980 пикселам, что, как правило, не является шириной экрана устройства.

Пока `@viewport` поддерживается повсеместно, можно воспользоваться тегом `<meta>` с атрибутом, задающим параметры окна просмотра. Браузеры настольных компьютеров этот тег игнорируют:

```
<meta name="viewport" content="width=device-width;"/>
```

Атрибут `content` этого метатега может иметь несколько значений. Если не разрабатывается какая-нибудь интерактивная критическая по времени игра, то нужно включать именно такое окно просмотра. У пользователей должна быть возможность

масштабировать страницу в обе стороны. Предыдущий код позволяет им увеличивать изображение страницы, что играет важную роль с точки зрения доступности информации.

Если вернуться к CubeeDoo, то мы создаем полноэкранную интерактивную критическую по времени игру, в которой пользователю нельзя разрешить случайно увеличить или уменьшить изображение страницы. В отличие от большинства приложений других типов, в играх пользователям не нравится, когда игровое поле не помещается в окне. К предотвращению возможности изменения размеров страницы с помощью следующего метатега настройки окна просмотра нужно прибегать только при наличии веских причин (что, собственно, мы и делаем в случае с некоторыми играми):

```
<meta name="viewport" content="width=device-width;
  initial-scale=1.0; maximum-scale=1.0; minimum-scale: 1;
  user-scalable=0;"/>
```

Этот пример приведен с избытком настроек. Он читается так: «Установить ширину окна просмотра равной ширине экрана устройства. Сделать это значение начальным размером, минимальным разрешенным размером при масштабировании и максимальным разрешенным размером при масштабировании и запретить масштабирование». Чтобы показать значения, в пример включено больше свойств, задающих параметры содержимого, чем нужно. Логичнее было бы написать следующее:

```
<meta name="viewport" content="width=device-width;
  initial-scale=1.0; user-scalable=0;"/>
```

Вообще-то вам понадобится настройка `width=device-width`. Но если у вашего сайта для разных контрольных точек имеется конкретно указанная ширина (чего я не рекомендовала бы делать), можно объявить конкретную ширину. Например, если средняя контрольная точка дизайна сайта установлена точно на отметке 550 пикселов, можно написать следующий код:

```
<meta name="viewport" content="width=550">
```

Я не могу припомнить ни одного случая, когда объявление единой ширины содержимого имело бы под собой какой-то здравый смысл. Этого делать не следует. И здесь этот код показан для того, чтобы вы видели, с чем можно встретиться и по какому коду можно сразу отсеивать плохих разработчиков из числа кандидатов в свою команду.

@viewport. В метатеге `viewport` используются возможности HTML по управлению презентацией, что должно было бы относиться к области, регулируемой с помощью CSS. Смешивание управления презентацией с уровнем содержимого нельзя признать верным решением. Но на момент написания книги доступно было только лишь оно. Часть браузеров (Opera, IE10 и «ночные сборки» WebKit) поддерживают эт-правило `@viewport`. Но пока `@viewport` не получит более широкой поддержки, решением будет оставаться метатег `viewport`.

Прикоснись ко мне

Наше внимание сосредоточено на мобильных устройствах и, следовательно, только на том, что поддерживается современными браузерами. А все современные браузеры поддерживают DOM-метод добавления отслеживателя события `addEventListener`. Поскольку речь идет о мобильных устройствах (в обобщенном представлении), мы перехватываем прикосновения, а не перемещения указателя мыши и щелчки ее кнопками.

Два основных отличия прикосновений от щелчков заключаются в размерах области, в отношении которой выдается событие, и в количестве событий, которые могут выдаваться одновременно. Области прикосновений намного больше, чем области, в которых фиксируется щелчок кнопкой мыши: площадь прикосновения пальца довольно велика, а указатель мыши выделяет всего один пиксел. К тому же сенсорные устройства поддерживают события нескольких прикосновений, поскольку к сенсору устройства можно прикасаться несколькими пальцами.

Разные устройства поддерживают разные жесты и фиксируют разное количество прикосновений. К примеру, iPad может фиксировать до 11 прикосновений одновременно. Стандартные события, связанные с мышью, не обслуживают множественные щелчки: одинарный щелчок кнопкой мыши выдает одно событие щелчка в одной точке.

Каждое прикосновение пальцем или стилусом является событием щелчка, но некоторые устройства будут выжидать от 300 до 500 мс, прежде чем отреагировать на прикосновение, чтобы убедиться, что жест или прикосновение можно рассматривать как одно, а не несколько прикосновений. Соответствующий вопрос будет рассмотрен в следующем разделе.

Следует учесть, что при использовании координат указателя мыши или прикосновения палец нельзя в точности сравнивать с указателем мыши! Мышь может быть очень точным устройством. А палец? Он не может обеспечить такую же точность.

Области прикосновений

С точки зрения конструкции и удобства пользования сенсорные устройства обладают уникальными свойствами. При том же уровне прилагаемых усилий пользователь может получить доступ к каждому пикселу на экране. Для выбора пользователь использует собственные пальцы, у которых куда более значительный радиус указания, чем у указателя мыши.

Ваша конструкция должна отражать эти различия — следует предусматривать более крупные зоны попадания и бóльшие промежутки между этими зонами. Для кнопок рекомендуется выбирать высоту 44 пиксела, минимальная их высота — 22 пиксела, наиболее приемлемо пространство между чувствительными зонами 20 пикселов (минимум 10 пикселов).

Когда пользователь прикасается к экрану, та его часть, которая находится под пальцем или ладонью, может быть скрыта. Пользователь может пользоваться как правой, так и левой рукой. Нужно брать в расчет, что именно может быть скрыто под ладонью, в зависимости от того, какой рукой он будет пользоваться, и насколько важным является то содержимое, которое может быть скрыто.

Палец, прикасающийся к экрану, и даже часть ладони могут закрывать некоторые области экрана. Нужно сделать так, чтобы надписи находились выше связанных с ними полей формы, а при сенсорных событиях не выводились никакие временно появляющиеся диалоговые окна. А если без таких окон не обойтись, они должны располагаться выше области прикосновения, а не сбоку или ниже ее.

Операционной системой устройства используются несколько пальцевых жестов, которые могут различаться от системы к системе. Эти жесты следует знать, особенно когда разрабатывается система взаимодействия с пользователем для вашего сайта. Некоторые устройства под управлением iOS используют для переключения между приложениями обнаружение прикосновения четырьмя пальцами. Может быть, понадобится отказаться от выполнения жестов близко к краям окна просмотра, поскольку некоторые мобильные устройства осуществляют переходы между окнами, вкладками или приложениями, когда пользователь делает скользящее движение пальцем по экрану от края или к краю экрана. При конструировании и разработке приложения и пользовательского взаимодействия с ним нужно учитывать все характерные особенности операционных систем мобильных устройств.

События мыши, сенсорные события

Работа в Интернете происходит благодаря мыши. Если бы у каждого документа (ну или почти у каждого) не было ссылок, реагирующих на щелчки кнопкой мыши и ведущих на другие документы, это не был бы Интернет. Игр не существовало бы, если бы вы не могли с ними взаимодействовать. Эти взаимодействия осуществляются, как правило, с помощью щелчков кнопкой мыши.

За последние примерно 20 лет разработчики добавили к своим страницам события щелчка кнопкой мыши. Когда мы прикасаемся к экранам мобильных устройств, слегка ударяем по ним и выбираем элемент нажатием пальца, то щелчком это не считается. Для взаимодействия со многими мобильными устройствами и некоторыми ноутбуками также можно использовать легкие удары по экрану. Но вообще-то Интернет стал Интернетом благодаря событиям щелчка. И если бы сенсорные устройства не поддерживали эти повсеместно используемые события мыши, мобильный Интернет вряд ли состоялся бы.

Поскольку работа в Интернете построена на событиях мыши, эти события работают на сенсорных устройствах, на которых нет никаких устройств для указания. События мыши имитируются после запуска сенсорного события. Эти события выдаются в эмулируемой среде, но их порядок не гарантируется. Каждое прикосновение выдает событие щелчка, нажатия клавиши мыши, входа, выхода и т. д., но мы никогда не можем быть уверены в том порядке, в котором они происходят.

Каждое устройство эмулирует события мыши при использовании прикосновения и предоставляет нам другие характерные для прикосновений события, которые мы можем захватить. Что же касается сенсорных событий, есть две реализации, в которых нужно разобраться: во-первых, это события прикосновений и жестов, разработанные в Apple и представляющие собой незавершенную спецификацию, которая была отменена из-за патентов Apple, а во-вторых, это события указателя и жестов, разработанные в Microsoft и являющиеся свободными от патентов спецификациями, которые станут стандартами и вскоре будут реализованы в Chrome и Firefox. Стандарты и реализации все еще развиваются.

События указателя

Корпорации, особенно Apple, патентуют все, включая обычные повседневные вещи, такие как скругленные углы и приемы взаимодействия с пользователем. Были предприняты попытки сделать собственностью некоторые жесты: Apple фактически запатентовала сенсорные события. Спецификации представлены открытыми стандартами. В связи с этим возникла проблема. На помощь пришли события указателя! Microsoft создала собственную версию событий — события указателя (pointer events) и предложила их консорциуму W3C в качестве основы для стандарта[1].

Эти события не следует путать с CSS-свойством `pointer-events`, поскольку события указателя — это модель событий для указателя мыши, пера, прикосновения (включая множественное прикосновение) и всех остальных указывающих устройств. В случае поддержки мы будем располагать не только событиями JavaScript, к которым так привыкли, таким как `mouseover` и `mouseout`, но и событиями `pointerdown`, `pointerup`, `pointercancel`, `pointermove`, `pointerover`, `pointerout`, `pointerenter` и `pointerleave`. В дополнение к событиям, которые мы можем отследить, устройство будет перехватывать подробности сенсорных событий или событий указателя, такие как размер прикосновения, его тип, сила давления и угол. В данный момент в IE10 применяется единственная реализация событий указателя с префиксом Microsoft `MS`, следовательно, событие `pointermove` в IE10 кодируется как `MSpointermove`, а в IE11 префикс не используется.

Сенсорные события

Общеизвестно, что мыши и пальцы отличаются друг от друга. При использовании мыши один и тот же указатель проходит над элементами, позволяет войти и выйти и щелкнуть на отдельном пикселе. Пальцы выбирают более обширные области, к тому же у людей по пять пальцев на каждой руке! Устройство и обработчики событий должны отслеживать количество пальцев, взаимодействующих с экраном. Создавать и обрабатывать сложные жесты можно с помощью исходных сенсорных событий и событий мыши в сочетании с функцией `preventDefault()`.

Пока производители браузеров не согласуют и не станут поддерживать открытый стандарт событий указателя, у нас есть сенсорные события.

---

[1] Дополнительные сведения можно найти по адресу http://blog.jquery.com/2012/04/10/getting-touchy-about-patents/

Сенсорные устройства и их браузеры, включая Android Browser, Chrome, BlackBerry Browser, Opera и Firefox, поддерживают имеющиеся в iOS события `touchstart`, `touchend`, `touchmove` и иногда дающее сбой `touchcancel`. Четыре события возвращают объект `TouchEvent`, коллекцию `changedTouches` и объект `Touch`.

Объект `Touch` предназначен только для чтения и содержит свойства координат точки прикосновения, включая `pageX`, `pageY`, `screenX`, `screenY`, `clientX`, `clientY`, `target` и `identifier`. `TouchList` является списком отдельных точек контактов для сенсорного события. Объект `TouchEvent` содержит коллекции `touches`, `targetTouches` и `changedTouches`, а также логические свойства `altKey`, `metaKey`, `ctrlKey` и `shiftKey`.

Когда устройства касаются одним пальцем или стилусом, выдается одно событие. А если в касании задействованы несколько пальцев, выдаются несколько событий. Когда происходит нажатие на экран, выдается событие `touchstart`. При движении пальца по экрану многократно выдается событие `touchmove`. Когда давление на экран прекращается, выдается событие `touchend`. Событие `touchcancel` выдается, когда другое приложение, например поступивший телефонный звонок, прекращает прикосновение.

Если телефонный звонок раздается в то время, когда пользователь играет в игру, слушает подкаст или смотрит видеоклип, есть ли смысл на время звонка ставить на паузу игру или видео или останавливать звук? Мы не хотим расстраивать наших пользователей истечением срока игры и потери ее результатов при каждом ответе на звонок. В CubeeDoo при возникновении события `touchcancel` игра ставится на паузу:

```
document.addEventListener('touchcancel', function() {
  if (!qbdoo.game.classList.contains('paused')) {
    qbdoo.pauseGame();
  }
});
```

Сенсорные устройства поддерживают множество жестов, перехват которых может потребоваться. Люк Врублевский (Luke Wroblewski) составил справочное руководство по жестам на сенсорных устройствах — Touch Gesture Reference Guide, в котором даны определения различных жестов прикосновений, различаемых операционной системой. Я рекомендую его распечатать и повесить на стену над столом (рядом со схемой уровней конкретизации из приложения).

Функция обнаружения сенсорных событий

Если для браузеров сенсорных устройств и настольных компьютеров используется один и тот же код, то, наверное, нужно будет расширить область прикосновений для ссылок и сократить задержку между одиночным прикосновением и его событием.

При этом можно подумать, что выходом из положения могут стать медиазапросы: небольшие экраны, скорее всего, будут мобильными, а мобильные экраны — сенсорными. Но ведь есть еще и планшетные компьютеры, экран которых может иметь более высокое разрешение, чем у многих ноутбуков и небольших мониторов.

Похоже, решением может стать функция обнаружения, но она несовершенна. Функция обнаружения прикосновения определяет, поддерживает ли браузер сенсорные события, но не определяет, поддерживает ли их устройство. Свойства сенсорного события придется протестировать с помощью JavaScript:

```
var isTouchEnabled = 'ontouchstart' in window ||
  'createTouch' in document ||
    (window.DocumentTouch && document instanceof DocumentTouch);
```

Затем можно будет воспользоваться логическим свойством `isTouchEnabled`, чтобы справиться с устройствами как поддерживающими, так и не поддерживающими сенсорное управление, помня о том, что некоторые устройства, например мобильные телефоны для пользователей с ослабленным зрением или двигательными функциями, могут вообще не иметь никаких устройств-указателей.

Для имитации событий одиночного прикосновения в среде разработки на основе настольного компьютера попробуйте воспользоваться утилитой Phantom Limb.

Псевдо- или не-совсем-псевдособытия щелчков

При щелчке пальцем по экрану получить событие щелчка правой кнопкой мыши невозможно. Поэтому в мобильных устройствах реализована реакция на длительное прикосновение. Из-за отсутствия клавиатуры, мыши или щелчка правой кнопкой мыши мобильные браузеры обладают особым встроенным поведением.

Цвет выделения при прикосновении

На сенсорных устройствах отсутствует эффект прохода указателя над элементом. Поэтому есть такое понятие, как цвет выделения ссылки при прикосновении, которым можно управлять с помощью свойства `tap-highlight-color`. Стиль цвета выделения можно подобрать в соответствии с общим дизайном. Хотя значение `transparent` часто позволяет избавиться от всяких неприглядных эффектов, следует помнить, что избавление от проявления эффекта прикосновения негативно сказывается на обозначении доступности элемента:

```
#content a {
  -webkit-tap-highlight-color: #bada55;
}
#board a {
  -webkit-tap-highlight-color: transparent;
}
```

На нашей доске ссылки отсутствуют, а если бы они были, этот код вызвал бы изменение фона любой ссылки в верхней части экрана на цвет `#bada55`, но это не коснулось бы ссылок на игровой доске и они не реагировали бы на прикосновения ничем, кроме эффекта переворачивания карты, управляемого отдельным кодом.

Избавление от диалоговых окон в ходе выбора

При прикосновении с последующим удержанием копируемого текста или при прикосновении и перетаскивании могут появляться диалоговые окна с запросами разрешения на копирование или определение выбираемого текста. Этот процесс на WebKit-браузерах поддается управлению с помощью объявления `-webkit-user-select: none;`. Когда для свойства `user-select` такого DOM-узла, как абзац или даже `<body>`, устанавливается значение `none`, диалоговые окна, связанные с копированием или определением, не появляются.

А вот пара «свойство — значение» `pointer-events: none;` в данном случае не подойдет. Она, конечно, избавит пользователя от появления диалоговых окон, связанных с копированием или определением, но при этом будет создано препятствие для возникновения любых других сенсорных событий в отношении целевого DOM-узла, предназначенного для пользовательского выбора.

Избавление от диалоговых окон в отношении изображений

Наряду с диалоговыми окнами, появляющимися при выборе элементов, когда пользователь прикасается к изображению и удерживает на нем палец, появляется панель с предложениями сохранения или копирования изображения. Чтобы избавиться от диалоговых панелей, появляющихся при прикосновениях к изображениям, нужно добавить ко всем изображениям объявление `touch-callout: none;`:

```
img {
-webkit-touch-callout: none;
}
```

Чтобы не испортить впечатления пользователя и не снизить доступность элементов, на сайтах с информационным содержимым показанные ранее свойства CSS применять не следует. Ими лучше воспользоваться в играх и других развлечениях, а также в рабочих и инструментальных приложениях.

Избавление от появления меню операционной системы

Не хотелось бы, чтобы пользователи случайно вызвали появление меню операционной системы. А CSS позволяет отключить его появление. Но полное отключение на все время нам не нужно, поэтому можно воспользоваться объявлением `touch-action: none;`, чтобы предотвратить возможное случайное появление меню:

```
.active #board {
  -ms-touch-action: none; /* отключение появления меню */
  }
```

Может возникнуть вопрос: «А почему бы не воспользоваться функцией JavaScript `preventDefault()`?» Наверное, можно воспользоваться и этой функцией. Но четыре только что рассмотренных свойства CSS будут работать производительнее функции `preventDefault()`. Код CSS почти всегда работает производительнее кода JavaScript. В данном случае на запуск сенсорных событий будет затрачено 400 мс, поэтому

лучше до того, как это произойдет, избавиться от появления меню, диалоговых окон, панелей и т. д.

onTouchEnd

Устройству неизвестно о ваших намерениях коснуться экрана одним или сразу двумя пальцами, поэтому после первого касания выжидают 300–500 мс, прежде чем выдается сенсорное событие. Браузеры на сенсорных устройствах, способные работать с прикосновениями, будут ждать с момента прикосновения до выдачи события щелчка от 300 до 500 мс в зависимости от устройства. Причина ожидания браузера связана с определением возможного двойного касания сенсора. Поэтому может появиться желание с помощью обработчика события захватить первое касание (не дожидаясь истечения времени между предполагаемыми двумя касаниями).

Если пользователь вызывает сервер или другой медленный процесс, ему нужно дать понять, что касание было принято. В зависимости от скорости соединения до получения ответа от сервера может пройти довольно много времени. Если ответ сервера на действие пользователя занимает более 100–200 мс, нужно уведомить пользователя о текущем состоянии дел.

В CubeeDoo обращения к серверу отсутствуют, поэтому в добавлении свойства «ожидания» нет необходимости. А вот ждать 300 мс после прикосновения к экрану того, что карта будет перевернута, нет никакого желания. Мы знаем, что в разрабатываемом приложении нет никаких двойных щелчков, требующих обработки, поэтому столь долгое ожидание до начала действия, инициируемого щелчком, будет напрасной тратой времени пользователей.

Прежде чем что-то менять во взаимодействии с пользователем на желаемое, нужно хорошенько все взвесить. В нашем примере нет никакого смысла ожидать от пользователя двойного щелчка: здесь не разрешается увеличивать или уменьшать изображение на экране или же осуществлять какие-либо другие функции, связанные с двойным касанием. Захват прикосновения к картам мы осуществляем с помощью события `touchend`. Ускорение реакции браузера на сенсорные события потребует применения небольшого фрагмента кода на JavaScript, который позволит приложению откликаться не на события `click`, а на события `touchend`. События `touchend` запускаются сразу же после того, как палец убирается с экрана, поэтому они выдаются значительно быстрее, чем события щелчка (`click`), до выдачи которых проходит 300–500 мс[1].

В тех браузерах, которые не поддерживают сенсорные события, для работы с картами придется использовать обработчик события `onclick`, но при этом не хотелось бы обрабатывать `touchend`, а затем на 300 мс позднее выдавать событие щелчка. Если бы это была кнопка или ссылка, нужна была бы гарантия того, что мы случайно не запустили два события в отношении одного и того же узла, вызвав в отношении события `touchstart` функцию `preventDefault`. Этот вызов не даст в результате текущего прикосновения осуществляться щелчкам и прокруткам:

---

[1] В Firefox и Chrome, если увеличение или уменьшение изображения экрана отключено, событие щелчка выдается немедленно, без ожидания в течение 500 мс.

```
eventHandlers: function() {
  if ('ontouchstart' in window ||
    'createTouch' in document ||
      (window.DocumentTouch && document instanceof DocumentTouch)) {
      qbdoo.btn_pause.addEventListener('touchend',
        qbdoo.pauseGameOrNewGame);
      qbdoo.btn_mute.addEventListener('touchend',
        qbdoo.toggleMute);
      qbdoo.clearScores.addEventListener('touchend',
        qbdoo.eraseScores);
      document.addEventListener('touchcancel',
        qbdoo.pauseGameOrNewGame);
  }
  qbdoo.btn_pause.addEventListener('click', qbdoo.pauseGameOrNewGame);
  qbdoo.btn_mute.addEventListener('click', qbdoo.toggleMute);
  qbdoo.clearScores.addEventListener('click', qbdoo.eraseScores);
  qbdoo.themeChanger.addEventListener('change', qbdoo.changeTheme);
},
```

Еще одним решением будет добавление к тегу <body> прослушивателей событий click и touchend, прослушивающих фазу захвата. При инициализации прослушивателя события определяется, были ли щелчок или прикосновение результатом взаимодействия с пользователем, которое уже обработано. В таком случае в отношении его вызываются функции preventDefault и stopPropagation. Следует учесть, что некоторые настольные компьютеры поставляются с сенсорными экранами, поэтому они всегда включают как события щелчка, так и события прикосновения, предотвращая в случае прикосновения исходное событие щелчка.

Прикосновение для прокрутки

В нашей игре прокрутка не используется. Обычно для прокрутки нужно прикоснуться к экрану, а когда мы убираем палец, логика сообщает нам о том, что произошло событие touchend. В настоящее время при прокрутке событие touchend выдается в большинстве мобильных браузеров, за исключением Chrome для Android. В спецификации не указывается, что при прокрутке сенсорные события должны быть отменены, но в этом есть определенный смысл.

Chrome для Android ведет себя несколько иначе, и это поведение было добавлено в спецификацию событий указателя. Спецификациям для сенсорных событий с этой проблемой не справиться, зато с ней могут справиться события указателя. При поддержке событий указателя прокрутка, сжатие, расширение и другие приемы взаимодействия с устройством (а не со страницей) будут выдавать событие отмены.

Доступ к оборудованию

Управление прикосновениями является одним из заметных отличий мобильных устройств, но не является их прерогативой. Появляется все больше мониторов

ноутбуков и настольных компьютеров, а также других устройств, воспринимающих прикосновения. Сенсорные экраны к тому же не являются единственными новыми свойствами оборудования, с которыми можно выстраивать взаимодействие. В зависимости от операционной системы, устройства, браузера, использования CSS, JavaScript и HTML5 можно создавать браузерные приложения, взаимодействующие с оборудованием системы тем способом, который предусмотрен для установленных на устройстве стандартных приложений.

Перемещение телефона и его направление

На многих мобильных устройствах имеются сенсоры, с которых можно получать данные, используя JavaScript. К числу таких сенсоров относятся акселерометр, магнитометр и гироскоп. Для обработки данных об ориентации устройства имеется спецификация свойства `deviceOrientation`, которая предоставляет три оконных события, подробно рассматриваемых в следующих абзацах.

Акселерометр измеряет ускорение или линейное движение по трем осям. Используемый для обнаружения движения, наклона и встряхивания, он измеряет ускорение (м/с^2), прикладываемое к устройству по трем физическим осям (*X*, *Y* и *Z*), а также силу притяжения. Для обнаружения данных акселерометра можно обработать событие `devicemotion`:

```
window.addEventListener('devicemotion', function( ) {
  // сюда нужно добавить ответ на событие
});
```

Магнитометр, подобно компасу, определяет, куда направлено устройство, но не обязательно указывает на север. Магнитометр измеряет силу магнитного поля в трех направлениях, определяя индукцию окружающего геомагнитного поля по трем физическим осям (*X*, *Y*, *Z*) в микротеслах (мкТ). Событие `compassneedscalibration` выдается, когда устройство определяет, что компас нуждается в калибровке для повышения точности данных. Для калибровки пользователь проходит с устройством по маршруту, напоминающему цифру 8:

```
window.addEventListener('compassneedscalibration', function( ) {
  // сюда нужно добавить ответ на событие,
  // обычно о необходимости пройти с устройством по восьмерке
});
```

Гироскоп измеряет скорость вращения устройства (рад/с) вокруг всех трех физических осей (*X*, *Y* и *Z*). Поскольку скорость вращения вокруг одной оси измеряется на основании углового момента за вычетом силы притяжения, гироскоп может предоставить информацию о вращении устройства и ориентации, если нужно измерить степень вращения или поворота устройства пользователем. Если событие `deviceorientation` поддерживается, его можно перехватить:

```
window.addEventListener('deviceorientation', function( ) {
  // сюда нужно добавить ответ на событие
});
```

Когда пользователь перемещает устройство, выдается событие deviceorientation, включающее свойства α (0... 360), β (–90... 90) и γ (–180... 180) для вращения устройства вокруг его осей *Z*, *X* и *Y* соответственно. Показатели свойств относятся, как правило, к направлению, в котором удерживалось устройство, когда данные о его ориентации были получены впервые, и поэтому свойство deviceorientation особенно полезно для определения перемещений относительно исходной позиции.

Состояние устройства

Можно не только выяснить, как именно пользователь держит устройство, но и определить, в каком оно состоянии. Подключено ли оно к сети? Если подключено, то каков тип подключения? Есть ли заряд у батареи устройства?

Сетевое подключение

Сетевой API-интерфейс (Network API) позволяет воспользоваться атрибутом navigator.connection.type со строковым значением unknown, ethernet, wifi, 2g, 3g, 4g или none. Некоторые браузеры возвращают для этих значений целые числа или константы: WIFI, CELL, CELL_3G, CELL_2G, CELL_4G и UNKNOWN. Тип подключения возвращается API при первом подключении. Но устройства не находятся в состоянии постоянного подключения к Интернету, а тип подключения может измениться.

Самый последний API полагается на качество соединения, а не на тип подключения. Памятуя о том, что все телефонные компании лукавят насчет качества того соединения, которое они выводят на рынок, в таком подходе усматривается больше смысла, чем в какой-то обобщенной логике. Эта версия поддерживается не так хорошо, как предыдущая, но степень ее поддержки начинает расширяться. В самой последней спецификации вместо типа подключения объект navigator.connection предлагает атрибуты bandwidth и metered, а также событие change.

В свойстве navigator.connection.bandwidth возвращаются 0 (если устройство вне сети), бесконечность (неизвестно) или количество мегабайт в секунду в виде числа с двойной точностью. Свойство navigator.connection.metered возвращает значение true или false. Если значение равно true, значит, его провайдер накладывает на него какие-то ограничения и к пропускной способности канала нужно относиться осмотрительно. Например, если при поддержке данного свойства выясняется, что соединение не очень надежное, то пользователя можно спросить, не желает ли он отключить изображения и настроить для этого соответствующий cookie-файл.

Измененная обработка события может иметь следующий вид:

```
navigator.connection.addEventListener('change', function() {
  // Обработка события. Обычно проверка значения свойства bandwidth
});
```

Учтите, что объект connection все еще имеет префикс и может быть перехвачен с помощью следующего кода:

```
var connection = navigator.connection ||
        navigator.webkitConnection ||
        navigator.mozConnection;
if (connection.bandwidth != undefined) {
  if (connection.bandwidth <= 0) {
    // Вне сети
  } else if (connection.bandwidth <= 1) {
    // Менее 1 Мбайт/с / Соединение низкого качества
  } else if (connection.bandwidth > 1) {
    // Более 1 Мбайт/с / Соединение высокого качества
  } else {
    // Неизвестно
  }
} else {
  // API-интрефейс недоступен
}
```

Аккумулятор

API состояния аккумулятора (Battery Status API) позволяет определить текущее состояние аккумуляторной батареи с помощью объекта `navigator.battery`. С помощью логического свойства `navigator.battery.charging` можно определить, не находится ли батарея на зарядке.

Свойство `navigator.battery.chargingTime` вернет количество секунд, которые, предположительно, пройдут до полной зарядки аккумулятора. Свойство `navigator.battery.dischargingTime` вернет в секундах время, оставшееся до перевода системы в режим ожидания. А число с плавающей точкой в диапазоне от 0 до 1 в значении свойства `navigator.battery.level` покажет уровень заряда аккумулятора:

```
var percentBatteryLeft = navigator.battery.level * 100
```

Можно также перехватить события `chargingchange`, `chargingtimechange`, `dischargingtimechange` и `levelchange`.

Другие API

В число других API-интерфейсов для мобильных устройств входят:

- Pointer Lock;
- MediaStream Recording;
- Ambient Light Events;
- Proximity Events;
- Vibration;
- Web Intents.

API-интерфейсы Calendar, Messaging, Sensor и System Information были отставлены на полку. О состоянии различных API-интерфейсов можно узнать из списка, который ведет рабочая группа Device API Working Group.

Стандартные веб-приложения, пакетные приложения и гибриды

На устройствах под управлением iOS можно добавлять метатеги, позволяющие создавать на HTML веб-приложения, работающие в полноэкранном режиме наподобие стандартных приложений. В Apple приложения такого типа называются Web.app. Если при создании Wep.app пользователь устанавливает веб-приложение путем добавления значка сайта на главный экран, то при обращении к сайту с помощью этого значка он получит приложение, работающее в полноэкранном режиме. При обращении к веб-приложению с помощью соответствующих метатегов через соответствующий значок на главном экране пользовательский интерфейс браузера будет скрыт. Пользуясь этим, а также API-интерфейсом HTML5, рассмотренным в данной книге, мы можем по крайней мере на некоторых устройствах создавать похожие на стандартные, автономно работающие приложения, конкурирующие с любыми стандартными приложениями.

Иногда созданных по ранее приведенным описаниям искусственных стандартных приложений недостаточно. Хотя каждая операционная система требует, чтобы ее стандартные приложения создавались на различных языках программирования, их можно создавать на HTML5, CSS и JavaScript и конвертировать свои веб-приложения в стандартные.

Гибридными называются приложения, созданные на основе HTML5, CSS и JavaScript, которые были конвертированы или скомпилированы в стандартные, зачастую просто с использованием в качестве контейнера приложения полноэкранного веб-представления. Использование уже известных вам веб-технологий наряду с теми, которые были изучены благодаря данной книге (HTML, CSS, JavaScript), позволяет составлять пакеты и компилировать исходный код в стандартное приложение для операционных систем различных устройств. Полученное стандартное приложение можно распространять в различных магазинах приложений.

PhoneGap или Apache Cordova

Apache Cordova, ранее называвшийся PhoneGap, — это проект с открытым кодом и среда стандартных веб-приложений для нескольких платформ. PhoneGap позволяет нам экспортировать веб-приложения в стандартные приложения для большинства или для всех мобильных платформ.

PhoneGap не только позволяет создавать пакеты из веб-приложений в виде стандартных приложений для различных мобильных операционных систем, но и обеспечивает доступ к компонентам устройства, пока недоступным через браузер.

Например, наряду с тем, что функция `getUserMedia()` полностью поддерживается в версии Google Chrome для настольных компьютеров, запись видео из браузеров не имеет пока полной поддержки в мобильной среде. PhoneGap позволяет программировать в JavaScript то, что не имеет полной поддержки в мобильных браузерах. При экспорте такого веб-приложения в гибридное приложение с помощью PhoneGap упаковщик конвертирует код JavaScript в родной код, понятный операционной системе, предоставляя нашему гибридному приложению доступ

к функциям устройства, которые в данное время поддерживаются только в его родном пространстве.

Adobe PhoneGap Build является компилятором Cordova основанным на облачных технологиях, поэтому нам не нужно работать с исходными SDK.

Sencha Touch

Хотя Sencha Touch считается средой пользовательского интерфейса, начиная с версии 2.0 в нее включен упаковщик, подходящий для iOS и Android. Он доступен в средах разработки как для Windows, так и для Mac. Упаковщик и другие инструменты разработки могут быть загружены с сайта Sencha.

Appcelerator Titanium

Среда Appcelerator Titanium дает возможность создавать стандартные веб-приложения для iOS, BlackBerry и Android. Titanium обеспечивает своеобразный мост, позволяющий использовать родные компоненты пользовательского интерфейса из JavaScript. В ходе компиляции Titanium конвертирует JavaScript в родной код. Свободно распространяемую версию Appcelerator Titanium Studio IDE можно загрузить из Интернета.

Тестирование

В число главных инструментов разработчика должны входить интегрированная среда разработки (IDE) настольного компьютера и его браузер. Для первого шага в разработке вполне подойдет браузер Chrome: если приложение не действует в Chrome (исключая такие характерные для конкретного устройства особенности, как сенсоры и вызовы), оно не будет работать и на вашем телефоне. При разметке и кодировании браузер настольного компьютера будет вашим основным инструментом — он должен использоваться только для разработки и первичного тестирования, а затем сайты должны быть протестированы на множестве браузеров множества устройств.

Получение нескольких устройств может обойтись довольно дорого. В лотерее везет не всем. Понимая это, вы не должны стыдиться того, что не можете позволить себе иметь экземпляр каждого из устройств.

Конечно, проверить работу сайта нужно на максимально доступном количестве устройств различных размеров с различными версиями браузеров и операционных систем и при различных условиях подключения к Сети, однако протестировать его при всех допустимых комбинациях невозможно. Недорогим и простым решением для тестирования могут стать эмуляторы мобильных устройств. Тестирование на настоящих мобильных устройствах может быть медленным и утомительным, но необходимым процессом, однако эмуляторы могут сделать отладку более терпимой. Если код работает на настольном компьютере, но не работает на эмуляторе, скорее всего, он не будет работать на каком-либо устройстве.

Тестирование на эмуляторе проходит значительно быстрее, чем на телефоне. Но следует помнить, что эмуляторы — это не мобильные устройства. Они похожи

на эмулируемые ими устройства, но у них имеется ряд ограничений: у мобильного устройства, скорее всего, весьма ограниченный объем памяти. На настольном же компьютере, а стало быть, и у эмулятора избыток оперативной памяти. Различий между эмулятором и настоящим устройством, конечно, много, но эмулятор дает неплохую стартовую позицию.

Без тестирования на устройствах все равно не обойтись, а вопрос с лотереей до сих пор не решен. Между тем вам нужен доступ к настоящим устройствам. Если вы не в состоянии купить, одолжить или украсть множество различных устройств, нужно провести довольно широкое выборочное тестирование с прицелом на предполагаемых основных пользователей.

Учтите, что текущая и возможная пользовательская аудитория — это не всегда одно и то же. Если видно, что количество мобильных пользователей сайта составляет всего 1 %, то причина может быть в его недостаточной привлекательности на мобильных устройствах. Эта аудитория может сильно отличаться от той, которая будет пользоваться вашим сайтом, если он станет выглядеть привлекательнее для пользователей мобильных устройств. Текущая статистика мобильных пользователей отражает лишь сиюминутное впечатление от вашего сайта в мобильном пространстве.

При таком подходе к тестированию ряд нерешенных проблем все же остается. Существуют сотни отличий одних реальных устройств от других, а для некоторых платформ эмуляторов вообще не существует. Тут без тестирования на настоящих устройствах просто не обойтись.

Невозможно протестировать приложение во всех браузерах и на всех устройствах или даже в одном браузере и на каждом устройстве. Устройств слишком много, и постоянно появляются все новые и новые.

Я рекомендую взять несколько различных устройств с разными размерами экранов, операционными системами, ограничениями по объемам памяти и различными браузерами. Разумеется, вам вряд ли удастся заиметь все устройства. По возможности получите доступ как минимум к одному устройству, работающему под управлением каждой операционной системы, включая самые новые сенсорные версии iOS на планшетных компьютерах, телефонах или iPod touch versions, BlackBerry (предпочтительно версии 10), Windows 8 для телефонов или планшетных устройств, и по крайней мере к двум устройствам под Android версий 2.3 и 4+ или более поздней.

У вас не может быть, да и не должно быть всех устройств. Но эта выборка может обеспечить неплохой их диапазон. Android все еще продается, почему я и рекомендую приобрести одно из устройств под его управлением. Приобретать пользовательские устройства можно на eBay. Они там очень дешевы, если поцарапан экран или вышла из строя телефонная функция. А вам нужно лишь, чтобы работал и выводился на экран браузер. Хорошо сохранившиеся устройства для этого не нужны.

Этот набор устройств подойдет для основного тестирования. Но хочется, конечно, протестировать программу на более широком круге устройств. Если добиться от Samsung, BlackBerry, Nokia или Motorola отправки бесплатного

устройства для тестирования не получится, то, возможно, вы найдете в своем городе или через удаленный доступ исследовательские лаборатории того или иного устройства. Apple, скорее всего, никогда не предоставит свое устройство бесплатно, но если у вас устройств этой компании нет, может быть, они есть у одного из ваших друзей.

Тестирование на отдельном устройстве отнимает много времени. Тестирование на всех устройствах не представляется возможным. Но тестировать-то все равно нужно. Следует обеспечить гарантии качества вашего кода на множестве реальных устройств.

В процессе разработки можно использовать инструменты для тестирования веб-приложений таким образом, чтобы они наиболее точно отражали мобильную среду, что позволит исключить потребность в кропотливой проверке на реальных устройствах. На первых порах для тестирования могут применяться симуляторы и эмуляторы, которые рассматривались в главе 1.

Конечно же, хочется, чтобы сайт хорошо выглядел и не сбоил при выполнении задач, востребованных пользователем. Мы только что об этом говорили. Но только привлекательного вида и работоспособности для сайта недостаточно. Нужно обеспечить его хорошую работу или производительность. Поэтому далее разговор пойдет о производительности.

14 Производительность мобильных устройств

Не важно, предназначены ваши дизайн или разработка в первую очередь для мобильных устройств или нет, верхней строкой списка вопросов, требующих решения в процессе разработки, всегда должна быть производительность.

Хотя браузеры современных мобильных устройств обычно совершеннее браузеров настольных компьютеров, которыми мы все еще должны довольствоваться, сами устройства могут обладать такими же объемом памяти и ограничениями пропускной способности, как и компьютеры на Pentium III из далекого 1999 года.

Сами мобильные устройства, в отличие от мобильных браузеров, накладывают различные ограничения, с которыми приходится считаться в ходе разработки.

Объем среднего сайта составляет более 1 Мбайт. Хотя наиболее горячей темой большинства конференций по мобильным устройствам является адаптивный веб-дизайн, на самом деле намного важнее производительность приложений для мобильных устройств: кому будет интересно, на что похож ваш сайт на телефоне, если пользователи не смогут его загрузить или отобразить на своем устройстве?

Помимо контрольных точек нашего адаптивного дизайна, есть еще много вещей, над которыми нужно призадуматься. Следует озаботиться обнаружением функций, API-интерфейсами устройства, сенсорными событиями, стратегией вывода содержимого, условной загрузкой ресурсов и реальной производительностью устройства. На всем протяжении процесса разработки нужно брать в расчет продолжительность работы от аккумулятора, задержки, лимит оперативной памяти и степень отзывчивости пользовательского интерфейса.

Продолжительность работы от аккумулятора

В отличие от настольных компьютеров, постоянно привязанных к одному месту, и даже от ноутбуков, которые обычно используются в стационарных условиях, мобильные устройства не подзаряжаются в течение всего дня. Пользователи мобильных устройств ожидают от них работы между подзарядками как минимум в течение 24 часов.

Пользователи понимают, что на звонки и использование GPS тратится энергия батареи. Но если они думают, что используют свой браузер только для переходов по сайтам, то не учитывают того, что одни сайты сажают батарею их устройства

быстрее, чем другие. Управлять расходом энергии на выполнение нашего кода должны мы, разработчики.

Вы, наверное, заметили, что, когда ноутбук отключен от сети электропитания, его батарея разряжается из-за работы центрального процессора. Точно так же центральный процессор разряжает батарею и на мобильных устройствах. Все, что крутится, греется или заставляет включаться вентилятор ноутбука, будет расходовать энергию и на мобильных устройствах (если они не подключены к электросети). При программировании нужно исходить из худшего: предположить, что пользователи мобильных устройств работают на них без подключения к электросети.

Для управления энергопотреблением кода нам нужно управлять использованием центрального процессора. Избегайте перекомпоновок. Сведите к минимуму объем и активность использования кода JavaScript. Не нужно постоянно беспокоить радиоканал излишними AJAX-вызовами. Для анимации вместо JavaScript нужно всегда использовать CSS. И даже притом, что команды, продвигающие на рынке устройства, поддерживающие WebGL, уверяют, что их устройства оптимизированы, не используйте WebGL на мобильных устройствах. По крайней мере пока, поскольку энергоэффективность WebGL постоянно совершенствуется.

Используйте темные цвета

Чем светлее цветовые оттенки в вашем дизайне, тем ярче будет экран. На телефонах с AMOLED-экранами чем ярче экран, тем больше энергии он потребляет, разряжая тем самым батарею. Экраны AMOLED (active-matrix organic light-emitting diode — активная матрица на органических светодиодах) изготавливаются из тонкого слоя органических, испускающих свет полимеров. Поскольку подсветка отсутствует, экраны могут быть очень тонкими. Черные пикселы фактически выключаются, экономя энергию батареи. На этих экранах, не использующих жидкокристаллическую технологию, более светлые тона потребляют при выводе больше энергии, чем более темные.

Разумеется, и помимо энергопотребления есть проблемы, влияющие на цветовые решения в дизайне приложения. И все же следует отметить, что количество энергии, потребляемой сайтами, может существенно разниться в зависимости от цветовых оттенков, используемых в дизайне на конкретных устройствах. Цвета — далеко не единственная деталь, влияющая на энергопотребление. Свой вклад в энергопотребление вносят и такие медиаэлементы, как фоновые изображения, видео, аудио, анимация и JavaScript. По возможности нужно выбирать более темные цвета и оптимизировать энергопотребление других ваших функций.

Используйте JPEG-изображения

Вместо PNG-изображений нужно использовать изображения в формате JPEG. В этом формате изображения сильнее сжимаются и быстрее выводятся на экран, поэтому их использование более эффективно с точки зрения экономии энергии.

При выводе изображений потребляется энергия. В зависимости от количества, размера и типа изображений на вашем сайте на вывод изображений может тратиться существенный процент потребления электроэнергии. Количество энергии, необходимое для вывода изображений, пропорционально количеству и размеру выводимых изображений. На вывод изображений в формате JPEG тратится меньше энергии, чем на вывод GIF- и PNG-изображений: в соответствии с исследованием Who Killed My Battery: Analyzing Mobile Browser Energy Consumption[1] («Кто сажает мою батарею: анализ энергопотребления мобильного браузера») JPEG является самым энергоэффективным форматом для изображений всех размеров.

Использование JPEG-изображений приводит не только к сохранению заряда батареи, но и к уменьшению затрат памяти и ускорению перерисовок. Тип используемого формата изображения влияет на потребление энергии и при выводе этого изображения на экран. Это влияние распространяется и на процесс перерисовки изображения для получения другого размера. Как отмечалось ранее, при длительном отображении более светлых цветовых оттенков потребляется больше энергии. Когда речь заходит о цене вывода изображения на экран, имеется в виду аппаратное декодирование, изменение размера и вывод изображения на экран, а не затраты энергии на показ статического изображения.

Сокращайте объем кода JavaScript

Хотя растровые изображения являются самым серьезным потребителем трафика и все изображения потребляют много памяти, они являются не единственным виновником потребления больших объемов памяти и разрядки батареи. В этом виноват и код JavaScript. Для экономии заряда батареи и объема используемой памяти нужно свести к минимуму как размер, так и активность вашего кода JavaScript.

Когда браузеру встречается тег `<script>`, он прекращает загрузку дополнительных информационных ресурсов и вывод тех ресурсов, которые уже были загружены, до тех пор, пока не будет загружен, разобран и выполнен код JavaScript. Кроме того, браузер также не приступает к разбору и выполнению файла сценария, пока тот не будет полностью загружен, о чем вы уже знаете.

Но о чем вы, наверное, никогда не задумывались, так это о памяти и энергии, потребляемой кодом JavaScript. При каждом AJAX-вызове для выдачи запроса задействуется радиоканал устройства, разряжая при этом батарею. Энергия тратится и на каждый разбор кода JavaScript. Хотя сайт может кэшировать файл JavaScript, он все равно проводит разбор и выполнение кода JavaScript при каждой загрузке страницы. Динамический JavaScript, например XMLHttpRequest, повышает стоимость вывода информации на экран и не может быть кэширован. Обработчик выполняет код JavaScript при каждой обработке события. JavaScript выполняется также при каждой итерации функции `setTimeout`. И на все это тратится энергия.

---

[1] Who Killed My Battery: Analyzing Mobile Browser Energy Consumption. Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh? and Jatinder Pal Singh // http://mobisocial.stanford.edu/papers/bonehwww2012.pdf, c. 41–50. ACM (2012).

Загрузка, разбор и выполнение кода JavaScript может быть наиболее энергозатратным компонентом веб-страницы. Иногда использовать код JavaScript просто не нужно! Среду JavaScript нужно использовать только в случае крайней необходимости.

Мне попадались сайты, включавшие библиотеку jQuery просто для того, чтобы выбрать элемент и выполнить другие простые операции, с которыми запросто справляются селекторы и/или обычный код JavaScript. Например, чтобы добавить к первому элементу каждого неупорядоченного списка класс `first`, можно использовать jQuery, но делать это не обязательно:

```
$('ul li:first').addClass('first');
```

Этот код выполняет практически то же самое, что и следующий:

```
var firstLIs = document.querySelectorAll('ul li:first-of-type');

for (var i = 0; i < firstLIs.length; i++) {
  firstLIs[i].classList.add('first');[1]
}
```

Но последний код не приводит к добавлению на ваш сайт 34 Кбайт или выполнению дополнительного HTTP-запроса. И хотя 34 Кбайт — не так уж много, особенно по сравнению с размерами изображений, добавляемых людьми на свои сайты, если включить библиотеку jQuery, то, несмотря на то что файл jQuery может быть кэширован, он вдобавок должен быть обработан парсером (быть разобран) и выполнен при каждой загрузке страницы. Конечно, одна загрузка страницы не сможет полностью израсходовать остаток заряда батареи, но затраты 4 Дж[2] на каждую загрузку сделают это довольно быстро. И для пользователя может стать неожиданным тот факт, что не только применение GPS или просмотр фильма, но и сайт сможет разрядить батарею их мобильного устройства.

Речь не идет об отказе от использования среды JavaScript. Я всего лишь призываю включать ее в разработку, только если вы убеждены в необходимости ее использования, поскольку необдуманное включение приводит не только к неоправданным расходам памяти и трафика, но и к повышенному расходу энергии батареи.

Не следует импортировать библиотеку только для нацеливания на элемент с помощью CSS-селекторов. Для этого есть функции `querySelector()` и `querySelectorAll()`. И не следует ее импортировать лишь для привязки событий: функция `addEventListener()` неплохо работает во всех современных браузерах. Не нужно создавать сценарий только для улучшения работы прокрутки. Вместо этого следует воспользоваться объявлением `-webkit-overflow-scrolling: touch`. А если нужно получить идеальную прокрутку вниз с небольшим возвратом, воспользуйтесь сценарием. Не стоит изобретать колеса — все равно не получится. Используйте библиотеку в крайнем случае, но перед этим хорошенько подумайте, нужны ли вам лишние байты, HTTP-запрос, дополнительный расход памяти и разряд батареи.

---

[1] classList поддерживается мобильными браузерами, начиная с IE10, Android 3, iOS 5.

[2] http://mobisocial.stanford.edu/papers/boneh-www2012.pdf

Избегайте сетевых запросов

Файлы, необходимые для работы вашего веб-приложения, загрузить, конечно, нужно. На это расходуется энергия батареи, но без загрузки не обойтись. А вот опрашивать серверы Facebook, Twitter и Pinterest каждые 15 секунд, чтобы посмотреть, не набрала ли ваша страница еще больше лайков, совсем не обязательно, поскольку на это напрасно тратятся трафик и энергия батареи. Фактически это самая бестолковая трата обоих ресурсов.

Следует определить, нуждается ли ваше приложение в постоянных или редких опросах, или не нуждается в них вообще. Если нужно, чтобы приложение работало в системе реального времени, например в чате или спортивной игре, значит, вам необходимо тратить энергию батареи на поддержку постоянного соединения. Если ваше приложение не проводит целевой опрос (например, Facebook вам не нужен и уже надоел), позвольте своему мобильному устройству разорвать соединение с вышкой сотовой связи.

Установка и поддержка радиосвязи с сотовыми вышками приводит к расходу энергии батареи. Когда устройство не делает запросов, оно отключает процессы подключения для экономии заряда батареи. И это хорошо.

Хотя большинство споров о производительности ведутся вокруг ввода и вывода данных, основным потребителем энергии в мобильном телефоне является радио. Для экономии энергии батареи мобильные устройства после завершения передач переводят радио в режим сохранения энергии, а после нескольких секунд сетевой пассивности — в режим глубокой спячки. После простоя в течение 5 секунд радиоканал переводится в режим половинного потребления энергии и существенного снижения трафика. По прошествии следующих 12 секунд пассивности он переводится в состояние ожидания.

Чтобы перейти из состояния ожидания в режим полной мощности и полноценного трафика, требуется время. Если каждые 15 секунд выполняется опрос сервера, радиоканал пробуждается от глубокой спячки. Это пробуждение может занять от 2 до 3 секунд, при этом для перехода в то состояние, при котором приложение может вести передачу данных, должно пройти несколько циклов установки связи.

Если вашему приложению необходимо поддерживать постоянное соединение, то нужно так и сделать. При этом следует понимать, что батарея разряжается, и оповестить об этом пользователя. Если опрос с постоянными интервалами не нужен, то для сохранения энергии батареи делайте сообщения как можно короче и ограничьте количество и частоту сетевых запросов после загрузки страницы.

Аппаратное ускорение

Обычно, размышляя об управлении использованием центрального процессора, люди думают о своем сервере. Разумеется, об этом тоже следует думать. Но когда речь заходит об ограниченном сроке работы от аккумуляторной батареи, возникает потребность управления использованием центрального процессора со стороны брау-

зера при выполнении вашего веб-приложения. Все, что принуждает к включению системы охлаждения на вашем ноутбуке, также будет разряжать батарею на любом устройстве.

Одно из решений заключается в аппаратном ускорении всех анимаций. Под аппаратным ускорением понимается визуализация вашей анимации с помощью графического, а не центрального процессора. Графическая микросхема требует меньше энергии, чем центральный процессор устройства, что увеличивает продолжительность работы от батареи. Аппаратное ускорение берет на себя все операции рисования и выполняется на пространствах представлений (View's canvas) с использованием GPU. Изображения при аппаратном ускорении комбинируются, используя в четыре раза больше памяти, чем исходные. В результате возросшего объема ресурсов, необходимого для включения аппаратного ускорения, ваше приложение будет потреблять больше оперативной памяти, но меньше энергии батареи. В связи с ограниченными объемом памяти и сроком работы от батареи при конструировании и разработке приложений всегда нужно брать в расчет как расход энергии, так и расход памяти.

У аппаратного ускорения есть свои преимущества и недостатки. Ваша анимация на графическом процессоре будет идти более плавно, и на нее будет затрачено меньше энергии. Но при этом ограничивается объем оперативной памяти. Иными словами, объявление `transform: translatez(0);` нельзя считать панацеей от всех бед, и поэтому использовать такой код не следует:

```
* {
  transform: translatez(0);
}
```

поскольку можно выйти за рамки памяти, выделенной графическому процессору, особенно на тех устройствах, где ее объем и так ограничен. Но опасаться применения аппаратного ускорения в отношении анимируемых элементов не стоит. Фактически для снижения объемов трафика между центральным и графическим процессором рекомендуется возлагать визуализацию всех элементов, которые предполагается анимировать, на графический процессор:

```
.spinner {
  transform: translatez(0);
  animation: spin 1s linear infinite;
}
@keyframes spin {
  100% {
    transform: translatez(0) rotate(360deg);
  }
}
```

Обратите внимание на то, что в предыдущем примере мы добавили 3D-преобразование даже при отсутствии анимации. Если собираетесь когда-нибудь визуализировать элемент с помощью графического процессора, то пусть он получит постоянное графическое ускорение. Вы же не захотите, чтобы элемент на короткое время исчез при перемещении из центрального в графический процессор.

Избегайте перерисовок и переформатирований. Перерисовки и переформатирования являются одной из основных причин медленно работающего кода JavaScript и основной причиной некачественной анимации.

Перерисовка означает обновление изображения экрана при изменении местоположения элемента без воздействия на разметку. Перерисовку вызовет изменение цвета, видимости или фонового изображения элемента. Перерисовки обычно обходятся дешево, но могут быть и дорогими, если понадобится заново определять видимость всех узлов в DOM-дереве и всех слоев каждого узла. Еще дороже перерисовка обойдется при использовании альфа-прозрачности.

Визуализация альфа-прозрачных размытостей, например теней или альфа-прозрачных градиентов, всегда будет занимать больше времени, поскольку браузеру нужно вычислить итоговый цвет каждого пиксела на основе прозрачности по отношению к цвету нижнего уровня. Это вычисление выполняется даже в том случае, когда цвет не виден из-за находящегося над ним элемента дизайна, поскольку такие CSS-свойства, как фоновое изображение и тени, рисуются от заднего плана к переднему.

Времени на перерисовку затрачивается совсем немного. Как правило, оптимизация других областей может обернуться куда более весомой отдачей. Но при повторяющейся перерисовке, например при перемещении или анимации без аппаратного ускорения, минимизация времени перерисовки приобретает жизненную важность. Чтобы анимация не выглядела рваной, браузер должен перерисовывать анимированные узлы менее чем за 16,67 мс. При визуализации с использованием центрального процессора на пикселы, подвергающиеся перерисовке будет тратиться очень много процессорного времени.

Переформатирование влияет на производительность еще больше, поскольку связанные с ним изменения влияют на разметку части страницы (или всей страницы целиком). Переформатирование элемента вызывает последующее переформатирование всех его дочерних элементов и более глубоких потомков, а также тех элементов, которые следуют за ним в объектной модели документа. Переформатирование — это браузерный процесс по пересчету размеров и позиций всех DOM-узлов, когда браузеру нужно пересчитать размер элемента или когда он проводит повторную визуализацию части всего документа.

Когда браузеру нужно определить размеры или переформатировать один элемент документа, не имеющего абсолютного позиционирования или не находящегося на собственном уровне визуализации, он обычно заново форматирует не только соответствующий узел, но и элементы-потомки узла и все элементы, которые следуют за этим узлом.

ПРИМЕЧАНИЕ

Некоторые узлы, имеющие собственный уровень визуализации, включают сам документ, узлы с явно заданной CSS-позицией (относительной, абсолютной или трансформированной), прозрачные узлы, узлы с выходом за границы, альфа-маски или отражения, WebGL, содержимое, выводимое с аппаратным ускорением, и элементы <video>.

В ходе переформатирования взаимодействие пользователей со страницей блокируется. Поэтому предотвращение переформатирования и сведение к минимуму времени неизбежного переформатирования играют весьма важную роль. Переформатирование может быть вызвано выполнением сценариев и даже некоторого кода CSS. На скорость переформатирования могут влиять DOM-дерево, стили и цифровые объекты.

Переформатирование может быть вызвано многими факторами, в числе которых добавление, удаление, обновление или перемещение DOM-узлов, изменение свойств узла, относящихся к его отображению или к блочной модели, добавление стилей из таблицы или встроенных стилей, изменение размеров окна или изменение ориентации, прокрутка и запрос стилевой информации посредством кода JavaScript.

Чтобы сократить количество переформатирований, следует собирать запросы стилей в пакеты, изменять стили путем изменения CSS-класса, а не добавления встроенных стилей.

Вместо изменения отдельных стилей нужно изменять имя класса. Если стили имеют динамический характер, редактируйте свойство `cssText`, а не свойство `style`:

```
myNode.style.cssText += "; left: 50p%; top: 0;";
```

Сводите изменения DOM в пакеты и применяйте их разом к существующему DOM-дереву. Не требуйте без особой необходимости вычисляемых стилей. А если идете на это, сводите запросы в пакеты и кэшируйте результаты в локальных переменных, работая с копией. Вносите все изменения в клон содержимого, выполняя их в автономном режиме и возвращаясь назад по завершении.

Все должно делаться в `documentFragment` или в копии раздела редактируемого документа. Если нужно, можно даже скрыть элемент с помощью объявления `display: none`, внести множественные изменения, а затем переключить элемент в исходный режим отображения. Этот метод предусматривает лишь два случая переформатирования: при скрытии элемента и его повторном показе. Казалось бы, это тоже немало, но все же меньше, чем могло бы быть при внесении сотен изменений, вызывающих переформатирование существующего узла.

Чтобы переформатирование происходило быстрее, нужно уменьшить количество DOM-узлов, избавиться от слишком сложных CSS-селекторов и перевести всю анимацию на аппаратное ускорение.

Чем глубже находится DOM-узел, тем больше времени будет занимать каждое его переформатирование. Изменения на одном уровне DOM-дерева могут вызвать изменения на каждом его уровне, от самых последних потомков узлов на пути изменений до корневого элемента документа. Чем больше узлов, тем больше времени займет их переформатирование.

Если вы вносите сложные изменения, касающиеся визуализации, например, анимации, делайте это вне общего потока данных документа. Чтобы выполнить эту задачу, создайте с помощью объявления `position: absolute;`, `position: fixed;` или `transform: translatez(0);` отдельный уровень визуализации.

Задержки

Скорости загрузки или выкладывания данных очень редко соответствуют скорости трафика, объявляемой провайдерами Интернета (Internet service providers — ISP) при продаже услуги. Объявленные мегабиты в секунду представляют собой наивысшую скорость соединения, на которую можно надеяться, но отнюдь не скорость среднестатистического соединения. На скорость, с которой сайт, включая разметку, таблицы стилей, медиаматериал, сценарии приложения и сценарии сторонних производителей, попадает на наши устройства, влияют главным образом задержки и пропускная способность сетей под рыночными наименованиями EDGE или 3G[1]. Задержки сильнее влияют на скорость загрузки, когда устройство находится в мобильной сети, чем когда оно получает доступ в Интернет через Wi-Fi.

Скорость загрузки сильно зависит от потери пакетов и задержек. Пакеты должны сначала проделать путь от вашего устройства к ближайшей сотовой вышке. Основной причиной задержки является пространство между вашим устройством и этой вышкой. Иными словами, мобильные пользователи, пользующиеся 3/4G, уже имеют низкую пропускную способность. Задержки превращают их веб-серфинг в мучение.

Из-за ужасных задержек на мобильных устройствах сокращение времени загрузки приобретает особую важность. Наибольшее влияние на повышение производительности оказывают оптимизация сайтов для работы на мобильных устройствах, сокращение количества HTTP-запросов и DNS-поисков. Если устройство тестируется в локальной сети Starbucks Wi-Fi, больших задержек вы не почувствуете. Попробуйте протестировать свое приложение, находясь на пассажирском сиденье движущегося по живописному шоссе скоростного транспортного средства, тогда вы намного лучше поймете, что именно нужно оптимизировать.

Сокращайте количество HTTP-запросов

Существует несколько путей, позволяющих сократить количество запросов, выдаваемых вашим приложением.

Кэш браузера

Использование кэша браузера сокращает запросы на последующие загрузки обновления страницы, поскольку, когда ресурс кэширован, браузеру не нужно заново его извлекать.

Для неизменяемых статических компонентов, таких как логотип корпорации, установите значение заголовка `Expires` на весьма отдаленное будущее. Для динамических компонентов, например для JSON-ответа, воспользуйтесь соответствующим заголовком `CacheControl`, чтобы помочь браузеру разобраться с условными запросами.

---

[1] См. http://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/

Объединенный файл JavaScript

Зачастую неплохо будет объединить весь код JavaScript в одном файле. Но следует учесть, что у вашего устройства ограниченный объем памяти, поэтому наряду с объединением кода JavaScript в одном файле, что может значительно сократить количество HTTP-запросов, возможно, следует включить два файла сценариев: один для всего приложения (он используется на большинстве загружаемых страниц) и другой, с отдельными модульными сценариями, для более сложного компонента (компонентов) вашего приложения. Единственного верного ответа на данный вопрос не существует: следует осознать наличие как ограничений по памяти, так и задержек и определить, какой из вариантов имеет для ваших приложений наибольший смысл.

Неплохо было бы уменьшить файл JavaScript и обработать его утилитой gzip. Использование сети доставки содержимого может ускорить загрузку, но может и увеличить количество DNS-поисков.

Единая таблица стилей

Точно так же можно объединить весь код CSS в единый файл. Такие инструменты, как Sass, могут помочь вам в управлении модульными SCSS-файлами и объединить их в единый файл, используемый в конечном продукте. Но опять-таки следует помнить об ограниченных объемах памяти. Определите, что с точки зрения производительности имеет больший смысл при выполнении конкретно вашего приложения. По возможности объединяйте и кэшируйте как можно больше кода, а также уменьшайте его объем и пользуйтесь утилитой gzip.

Спрайты изображений

При разработке для настольного компьютера изображения тоже объединяются в спрайты. Спрайты изображений представляют собой коллекцию изображений, помещенную в одно изображение. Спрайты изображений сокращают количество HTTP-запросов, а также помогают сэкономить трафик.

Хотя спрайты предлагают весьма неплохой способ сокращения задержек, у них тоже есть недостатки. Мобильные устройства имеют ограниченную память, а эти изображения загружаются в память, даже если используется всего лишь небольшая их часть. Кроме того, в памяти большие изображения могут быть продублированы. Как правило, для устройств с ограниченной памятью рекомендуется использовать изображения размером не более 1024 пикселов в любом измерении.

Сжатие изображений

При создании изображений их нужно сжимать как можно сильнее. Хотя файлы изображений также можно обработать утилитой gzip, следует помнить, что при декодировании браузером они возвращаются к прежнему размеру, поэтому скажем еще раз: сжимайте изображения как можно сильнее.

URI данных или встроенные изображения

В случае использования небольших изображений и простых файлов изображений SVG количество HTTP-запросов для этого файла можно уменьшить до нуля, предоставляя URI данных в качестве встроенного изображения или в качестве URI данных для фонового изображения, вместо того чтобы заставлять браузер загружать отдельный двоичный файл. Встроенные изображения используют схему URI данных, чтобы вставить эти изображения в саму страницу. Это может увеличить размер вашего HTML-документа. Подумайте, стоит ли экономия HTTP-запроса увеличения размера файла таким вот методом.

Чтобы включить в свой код CSS URI-интерфейс данных, который помещает изображения в любое место, в которое они обычно помещаются, заключите его в функцию `url()`. Например:

```
a[href^="mailto"] {
  background: url(-data-uri-code-
  UhwFUUE1l)
  no-repeat right center;
  padding-right: 25px;
}
```

Если создавать код для получения фонового изображения в HTML, то он бы выглядел следующим образом:

```
<img width="16" height="16" alt="email" src="
  R0lGOYLCVDFCrKU-data-uri-code-UhwFUUE1lBavAViFID1TI0S1BCBMQiB" />
```

Здесь код URI данных был бы на самом деле существенно длиннее.

Строки URI данных могут быть очень длинными. В среднем URI данных имеют на 33 % больше байт, чем их двоичный растровый эквивалент. В обоих методах могут (и должны) использоваться приемы уменьшения размера файла с помощью утилиты gzip. Из-за способа отправки пакетов нужно проанализировать все за и против. Может быть, стоит отправлять дополнительные байты сравнительно небольших URI данных изображения, чтобы сэкономить на HTTP-запросах. Иногда неплохо было бы отправлять URI данных и для некоторых PNG-файлов высокого разрешения, поскольку фактическая загрузка самих файлов может быть несколько более продолжительной и ощущаемое время загрузки может оправдать дополнительный HTTP-запрос. Какой из вариантов наиболее выгоден с точки зрения производительности, зависит от устройства и характеристик соединения[1].

Так чем же воспользоваться — спрайтами или URI данных? Работоспособны оба варианта. Я считаю, что легче работать со спрайтами, чем экспортировать URI данных[2], но те, в свою очередь, безусловно, имеют свои преимущества. URI данных

---

[1] См. http://davidbcalhoun.com/2011/when-to-base64-encode-images-and-when-not-to.

[2] В авторской CSS-среде Compass имеется встроенный помощник для автоматизации URI данных. Существует также множество других доступных ресурсов.

удобно использовать для фоновых шаблонов[1] и в тех случаях, когда нужно слишком большое количество изображений или имеются непредсказуемые требования к изображениям, которые исключают возможность использования спрайта. Например, для рейтинговой системы, использующей звездочки, неплохо подошли бы спрайты, но аватары для Twitter с множеством возможных вариантов поместить в спрайт нельзя, и в этом случае наиболее верным решением будет использование URI данных.

Значки и наборы символов

После объявления набора символов своих файлов можно воспользоваться любыми символами из своего шрифта, включая ✉, ⌘ и ✓. Использование шрифта вместо значков изображений является более разумным решением. Можно создавать значки любого цвета, не пользуясь «Фотошопом». Можно создавать значки любых размеров без пикселизации. Использование исходного шрифта позволяет также экономить HTTP-запросы и память.

В семействах шрифтов, предварительно загруженных на устройства пользователей, скорее всего, будут найдены значки, отвечающие всем вашим потребностям. Иногда вашим дизайнерам захочется большей дизайнерской свободы. Для этого есть множество наборов шрифтов со значками. И для решения данного вопроса потребуется загрузка шрифта нужного вида, на что придется затратить один HTTP-запрос. То есть HTTP-запросов будет сделано столько же, если не меньше, чем в том случае, если бы вместо шрифтов использовались изображения.

Вы также можете создать собственный шрифт со значками. Неплохим ресурсом может послужить ресурс IcoMoon, представляющий собой набор свободно распространяемых значков и веб-приложение, предназначенное для подбора и загрузки оптимизированных шрифтов со значками. Вы просто выбираете значки, которые хотите загрузить, и сводите к минимуму их набор. Можно также добавить значки из различных наборов или SVG-файлов.

Проверка сетевых запросов

Производительность сайта или приложения можно проверить, посмотрев на водопадный график в браузерном инструментарии разработчика на настольном компьютере, используя такие средства, как weinre или Adobe Edge Inspect из мобильного устройства, отладчика BlackBerry или множества других инструментов.

Бесплатный водопадный график можно получить также из онлайн-инструментов, таких как WebPageTest.org (рис. 14.1). На графике показано количество сделанных запросов (в данном случае их было 34).

Для каждого запроса показано время, выделенное для DNS-поиска (если он был), начального соединения, время до пересылки первого байта (или задержки)

[1] В стадии разработки находится поддержка вывода на экран отдельных частей изображений, но пока эта функция не поддерживается.

и время загрузки содержимого (трафик вместе с задержкой). Первая вертикальная черта отмечает время начала визуализации страницы, а последняя вертикальная черта (едва различимая на рисунке справа) — время до выдачи события `onload`.

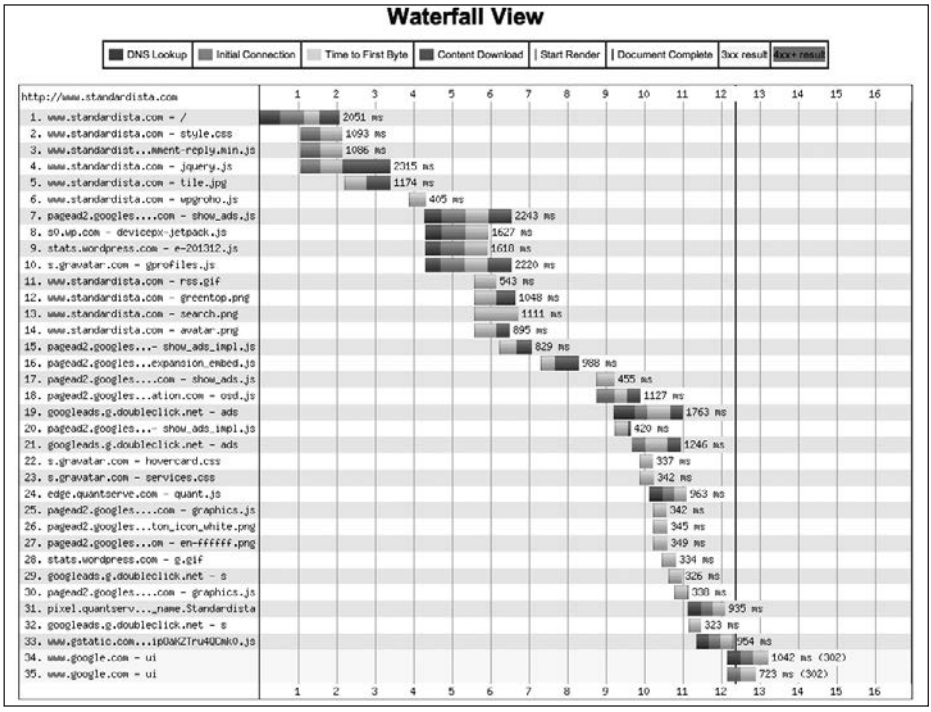


Рис. 14.1. Водопадный график с сайта WebPageTest.org

Если посмотреть на отдельно взятую строку из водопадного графика, показанную на рис. 14.2, можно заметить задержку на включение сценария из внешнего домена. В данном случае для включения внешнего файла JavaScript сначала нужно подождать 406 мс до окончания DNS-поиска и подождать 593 мс до загрузки первого байта сценария. При вызове сценария загрузка ресурсов прекращается до тех пор, пока сценарий не будет загружен, обработан парсером и выполнен. Этот запрос сценария `show_ads.js` продолжительностью 2243 мс не дает провести визуализацию страницы на протяжении более чем 2 секунд. Тем самым демонстрируется необходимость обращать внимание на задержку, порядок следования кода и производительность в целом, а конкретнее, на влияние использования сценариев сторонних разработчиков.

Рис. 14.2. Характеристика отдельного запроса

Ваша цель заключается в том, чтобы этот водопад был как можно короче и как можно у́же.

Сокращение размеров запросов

Задержки вызывают наибольшую обеспокоенность. Существенное влияние на задержки оказывает количество запросов, а не их размеры. Тем не менее чем больше размеры запросов, тем дольше они осуществляются. И зачастую память, расходуемая приложением, пропорциональна размеру файлов приложения. Среднестатистический сайт занимает более 1 Мбайт, притом что и на 24-дюймовый монитор, и на экран Android-устройства с диагональю 3 дюйма отправляются одни и те же файлы. Для сокращения задержки и позитивного влияния на расход памяти нужно сократить размеры запросов, совершаемых вашим приложением.

Чем меньше размер файла, тем меньше времени займет его получение на клиентское устройство с сервера, как только будет сделан запрос на подключение. Сократите размер CSS и JavaScript. Создайте изображения с самыми маленькими размерами файлов, обеспечивая при этом приемлемое разрешение.

Сокращайте размер ресурсов, имеющих текстовую основу

Нужно приводить к минимальному размеру все ресурсы, имеющие текстовую основу, — CSS-, JavaScript-, JSON- и SVG-файлы. Добиться наименьших размеров таких ресурсов помогут существующие службы и инструменты минимизации. Я могу еще понять ваше нежелание минимизировать такие ресурсы в ходе разработки, но перед развертыванием готового продукта минимизацию определенно следует провести.

Сжимайте двоичные файлы

Различные программы редактирования изображений предлагают разные способы сокращения размеров файлов. Вместо использования GIF-файлов сохраняйте изображения в формате PNG8. Если нужны анимированные GIF-файлы, используйте вместо них CSS-анимацию. Если ваша цветовая палитра шире той, что предлагается форматом PNG8, пропустите свои PNG-изображения через такие инструментальные средства, как ImageAlpha, или выполните их автоматическое преобразование из командной строки с помощью средства PNGCrush.

JPEG-сжатие происходит с потерей качества, но при степени сжатия от 40 до 60 % вместо 80–99 % можно добиться солидной экономии в байтах.

Используйте gzip при любой возможности

После доведения файлов до минимально возможного размера обработайте их утилитой gzip! Обработка файлов как можно большего количества типов облегчает страницу, увеличивая скорость ее загрузки. Однако при этом не сокращается влияние файла на расход памяти. Что касается памяти, размер файла на стороне клиента будет таким же, как и до его обработки утилитой gzip. Обработка экономит

трафик, но как только файлы окажутся в пункте назначения, они возвращаются к своему прежнему размеру, который был до сжатия.

Чтобы определить, какие файлы уже были или еще не были сжаты, воспользуйтесь инструментарием разработчика. Сначала через панель настройки отключите кэш инспектора. В результате будет получен замороженный кэш, или отправная точка для сравнительного анализа вашей работы. Именно в таком режиме посетители будут воспринимать ваш сайт при его первоначальной загрузке.

На вкладке водопадного графика Network (Сеть) в столбце объемов данных верхнее число будет показывать объем переданных данных, а нижнее — их реальный объем. В объеме переданных данных будет отражаться степень минимизации или сжатия утилитой gzip. В реальном объеме будет показано, сколько памяти будет выделено под данные и какова может быть экономия трафика. Если показано только одно значение, значит, это краткая строка ресурсов.

Чтобы увидеть полную строку ресурсов, нужно щелкнуть на значке расширенных строк в нижней части окна инструментария разработчика, который находится правее значка инспекции и левее значка записи (рис. 14.3).

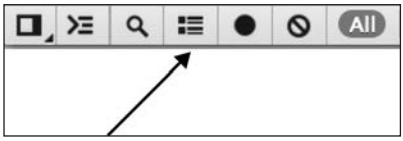


Рис. 14.3. Переключение между полным и кратким отображением ресурсов

В нижней части вкладки Network (Сеть) есть серая линия с текстом белого цвета, эта линия показывает полный размер файла и время его загрузки.

Сокращайте размеры изображений

Отправлять огромные изображения на небольшие по размерам устройства нет никакого смысла. Чтобы отправлять фоновые изображения надлежащего размера, нужно использовать медиазапросы. Для фоновых изображений следует использовать технологию Clown Car Technique, библиотеки типа Picturefill или такие инструментальные средства, как Sencha.io Src, чтобы отправлять изображения приемлемого размера. Дополнительные сведения об этих способах и ссылках можно найти в онлайн-ресурсах к главе. Более подробно размеры изображений рассматриваются далее, в подразделе «Память».

Обходитесь без использования рабочих сред

Если можно обойтись без импортирования среды JavaScript, значит, так и нужно сделать. Как уже упоминалось в подразделе «Аккумулятор» главы 13, рабочие среды изначально создавались для того, чтобы получить на всех браузерах стандартный JavaScript. На всех смартфонах имеются браузеры совершеннее IE8. jQuery добавляет 34 Кбайт, дополнительный HTTP-запрос и потребляет энергию при каждом его разборе и выполнении. Размер файла, конечно, небольшой, и если

нужна рабочая среда, ее можно использовать. Но если можно обойтись без импортирования сценариев, то так и нужно поступить.

И опять-таки не стоит изобретать колесо. Используйте библиотеку, если это необходимо, но прежде чем это сделать, хорошенько подумайте, нужны ли вам дополнительные байты, HTTP-запрос, расход памяти, затраты времени на полный разбор и выполнение кода и расход энергии батареи.

Минимизируйте количество cookie-файлов

Разумеется, в некоторых случаях, например при аутентификации, без cookie-файлов не обойтись. Браузеры и серверы отправляют cookie-файлы друг другу с каждым запросом. По возможности нужно избавиться от ненужных cookie-файлов, например от тех, которые предназначены для статического содержимого вроде изображений, для которых cookie-файлы не используются. Можно, конечно возразить, что `LocalStorage` может сократить издержки от пересылки cookie-файлов, но на доступ к `LocalStorage` требуется время, поэтому данное решение оказывается далеко не лучшим.

Применяйте отсрочку использования сценариев сторонних производителей

Не позволяйте внешнему сценарию расходовать энергию батареи или становиться компонентом, отказ которого приводит к отказу всего приложения, — single point of failure (SPOF). При вызове сценария до окончания его загрузки, разбора и выполнения приостанавливается загрузка всех остальных объектов. На рис. 14.2 показано, что на выполнение запроса относительно файла `show_ads.js` затрачивается 2243 мс, что задерживает визуализацию более чем на 2 секунды! Если при вызове сценария стороннего производителя произойдет сбой загрузки, ваше приложение не сможет загрузиться, пока не истечет максимальное время для загрузки сценария, если вообще сможет. Отложите вызов сценариев сторонних производителей или вообще откажитесь от их включения, если это возможно, чтобы быть уверенными, что ничей чужой сценарий не убьет ваш сайт.

Нерекомендуемые схемы, снижающие производительность

Из-за проблем с задержками сокращение количества DNS-поисков и HTTP-запросов приобретает в мире мобильных устройств жизненно важное значение. В некоторых сценариях может иметь смысл использование встраиваемых стилевых таблиц и сценариев. Я считаю, что с точки зрения оптимизации веб-производительности эту схему применять не рекомендуется, но вы можете со мной не согласиться.

Опыт ускорения работы сайта подсказывает, что нужно использовать внешние файлы JavaScript и CSS, а также сеть доставки содержимого — content delivery network (CDN). Но использование внешних файлов означает использование большего количества HTTP-запросов, а использование CDN-сетей для статического содержимого добавляет к этому как дополнительные DNS-поиски, так

и дополнительные HTTP-запросы. Хотя встроенный в ваш HTML код CSS и JavaScript противоречит пропагандируемому мной правилу, основанному на практическом опыте, если все сделать корректно, встраивание сценариев при первой загрузке может способствовать повышению производительности. Отличным примером этого может послужить мобильная версия сайта поисковой системы Bing.

В настоящий момент (как говорилось в главе 6, в подразделе, посвященном роли `LocalStorage` в повышении производительности мобильных устройств) при первом обращении по адресу m.bing.com с мобильного устройства весь сайт загружается одним файлом. Код CSS и JavaScript является встроенным. Изображения включены в виде URI данных. В мобильной версии Bing все объекты помещены в один файл, требующий только одного HTTP-запроса. Но этот один файл имеет размер 200 Кбайт. Это очень много. Но такой большой файл возвращается только после первого посещения Bing. Благодаря использованию `LocalStorage` и cookie-файлов в результате каждого последующего запроса к m.bing.com возвращается только один файл вполне приемлемого размера — всего около 15 Кбайт.

Все нужные файлы в Bing встроены в один HTML-файл. С помощью кода JavaScript, выполняемого на стороне клиента, Bing извлекает код CSS, JavaScript и изображения из исходной загрузки и сохраняет URI данных CSS, JavaScript и изображений в локальном хранилище. Имена сохраненных файлов Bing сохраняет в cookie-файле. При каждом последующем запросе страницы cookie-файл информирует сервер о том, какие файлы уже сохранены на локальном устройстве, позволяя серверу определить, какие ресурсы, если таковые имеются, нужно включить в ответ. Таким образом, последующие ответы наряду с кодом HTML включают в себя только те сценарии, стили и изображения, которые не были сохранены в локальном хранилище, если таковые имеются.

Сокращение негативных последствий задержек при загрузке мобильной версии сайта путем создания веб-приложения с одним HTTP-запросом для всего кода HTML, CSS, JavaScript и изображений включает в себя следующие шаги.

1. Встраивание кода CSS и JavaScript для первой загрузки страницы.
2. Извлечение и помещение ранее встроенных файлов в `LocalStorage`.
3. Настройка cookie-файлов с именами извлеченных встроенных файлов.
4. Проверка при последующих запросах cookie-файлов на стороне сервера.
5. Включение на основе значений cookie-файла только новых и недостающих сценариев.
6. Загрузка файлов из `LocalStorage` в ходе загрузки ответа от сервера.

ПРИМЕЧАНИЕ

Если причина большей эффективности этого метода по сравнению с простой загрузкой и кэшированием файлов вызывает у вас удивление, то следует заметить, что он не только повышает производительность путем исключения задержек при множественных DNS-поисках и HTTP-запросах, но и учитывает более ограниченный объем кэша на мобильных устройствах и отсутствие длительное время используемой памяти у операционной системы iOS.

Производительность повышается за счет помещения данных в `LocalStorage`. Но когда дело касается мобильных устройств, на первый план при повышении производительности все же выходит уменьшение задержек, особенно тех, которые связаны с ограниченной пропускной способностью.

Память

Большинство рекомендаций по повышению производительности сосредоточены на повышении скорости ввода-вывода данных. Но концентрации лишь на том, сколько времени тратится на завершение ответов в мобильном пространстве, недостаточно. Когда речь заходит о мобильных устройствах и ограниченных объемах памяти на большинстве мобильных устройств, нам нужно также управлять и тем, что внутри самого устройства. Как разработчики мы зачастую работаем на настольных компьютерах, имеющих практически неограниченную память. Но мобильные пользователи запускают наши сайты на устройствах с весьма ограниченной памятью.

За последние два десятилетия объем памяти на персональных компьютерах возрос почти экспоненциально. В 1997 году для запуска всего программного обеспечения на компьютере с процессором Pentium II вполне могло хватить 256 Мбайт. Но в 2013 году базовые (то есть небыстродействующие) модели компьютеров поставлялись как минимум с 4 Гбайт оперативной памяти. У iPhone 3G имеется 128 Мбайт памяти, у исходной модели iPad — 256 Мбайт, у более быстродействующего устройства HTC Inspire — 768 Мбайт. Нормой для новых смартфонов высокого класса является объем оперативной памяти от 512 Мбайт до 1 Гбайт и наличие процессоров с тактовой частотой минимум 1 ГГц. На мобильных устройствах работает программное обеспечение, написанное в 2013 году, но запущенное на устройствах, имеющих такой же объем памяти, который был у настольных компьютеров в 1999 году.

Хотя объем памяти 512 Мбайт может показаться вполне достаточным для запуска любого веб-приложения, при работе с памятью важно помнить, что браузер (и веб-приложение) не является единственным процессом, потребляющим ограниченную по объему оперативную память. Операционная система, фоновые процессы и другие открытые приложения (инициализированные как операционной системой, так и пользователями) используют общую память. Как правило, на мобильных устройствах запущено множество стандартных приложений, а также приложений, установленных как с ведома, так и без ведома пользователя. Запущенных приложений довольно много. К ним относятся такие инициированные пользователем приложения, как Twitter, GPS, Facebook, приложения, поставляемые с устройством, которые могли быть запущены и без ведома пользователя, например календарь и медиапроигрыватель, а также приложения, загруженные пользователем, например Angry Birds. Стандартные приложения операционной системы и все приложения с включенным оповещением в адрес пользователя продолжают работать в фоновом режиме. Устройство с объемом оперативной памяти 512 Мбайт, скорее всего, будет иметь менее 200 Мбайт доступной памяти. При работе с памятью следует помнить, что наиболее активные пользователи вашего веб-приложения наверняка будут пользоваться и другими мобильными приложениями. При тестировании нужно

применять реальные устройства. Запускайте на всех тестируемых устройствах такие приложения, как Twitter, Facebook и Mail.

Чем больше приложений запущено на устройстве, тем меньше будет доступной памяти для веб-приложения. И даже если ни одно из этих приложений не будет поглощать большие объемы памяти, значительное количество приложений, запущенных в фоновом режиме, создадут условия для потребления больших объемов памяти, что замедлит работу пользовательского интерфейса и приведет к острому дефициту памяти при работе браузера со всеми вытекающими последствиями. С целью освобождения памяти мобильный браузер будет либо закрыт, либо переведен в аварийный режим. Нужно управлять потребностями в памяти вашего веб-приложения, чтобы обеспечить ее умеренное потребление и предотвратить замедление работы или сбой мобильного браузера.

Оптимизация изображений

За исключением избегания CSS-выражений (YSlow) и оптимизации изображений (PageSpeed), рекомендации по оптимизации производительности связаны с вводом-выводом байтов, а не с тем, что происходит, когда сайт уже находится на устройстве.

Хотя обработка файлов утилитой gzip помогает повысить скорость загрузки, она не помогает решить проблемы управления памятью. Когда информационный ресурс попадает на устройство, он больше не находится в сжатом состоянии. На изображения расходуется память. Изображения, размер одной из сторон которых превышает 1024 пиксела, создают на некоторых устройствах большие проблемы с расходом памяти. Вам следует сократить потребление памяти файлами изображений, помещая изображения именно тех размеров, с которыми они будут отображаться на экране, и сжимая изображения до этих размеров.

В вашем распоряжении имеется несколько инструментов. ImageAlpha и ImageOptim могут помочь конвертировать файлы больших размеров с прозрачными изображениями в PNG-формате в восьмиразрядные PNG-файлы с полной прозрачностью. Прокси-сервер Sencha.io определяет, какой размер изображения требуется пользовательскому устройству, а затем сжимает (но не увеличивает) изображения перед их отправкой клиенту.

Сокращение размера файла изображения всегда играло важную роль для веб-производительности, но, когда речь заходит о мобильных устройствах, мы не можем просто сконцентрироваться на размере файлов ввода-вывода. Поскольку объем памяти ограничен, нужно учитывать размер файла изображения в несжатом состоянии. Память расходуют все изображения. Составные изображения используют память не центрального, а графического процессора. Поэтому, хотя освобождение памяти — не такая уж простая задача, составные изображения используют почти в четыре раза больше памяти, чем несоставные, поэтому использовать их нужно с особой осмотрительностью.

Как уже отмечалось, нужно размер всех информационных ресурсов сводить к минимуму. Есть ли ответ на вопрос: «Что значит слишком большой?» Ответ на него с позиций сегодняшнего дня может не совпадать с ответом применительно

к завтрашним устройствам. Ответ применительно к моей целевой аудитории может отличаться от ответа, справедливого для вашей целевой аудитории.

Наилучший совет, который я могу дать, заключается в определении того, какие ограничения должны быть наложены на ваше приложение до начала его разработки. Перед конструированием и разработкой приложения вам следует решить, какими должны быть ограничения по размерам информационных ресурсов. В ходе разработки нужно стараться придерживаться намеченных для себя ограничений. Это поможет сфокусироваться на производительности в течение всего процесса разработки. У вас или у кого-нибудь из вашей команды может появиться желание включить функцию, выводящую за пределы выделенных объемов. Если не задумываться об ограничениях, назначенных по своему собственному усмотрению, то вряд ли придется ставить под сомнение необходимость в том или ином информационном ресурсе. А когда такие ограничения установлены, вам придется оценивать необходимость подобного компонента. Если без него не обойтись, нужно подумать о том, как можно уменьшить его размер. Если его потребности в памяти и пропускной способности будут снижены до наименьшего из возможных уровней, но попрежнему будет ощущаться необходимость в нем, нужно подумать, размеры чего еще можно сократить, чтобы вписаться в назначенные ограничения. В итоге вы можете выйти за пределы ограничений. Но к этому моменту сайт станет значительно меньше, чем в том случае, если бы вы не рассматривали возможность сокращения объема трафика, используемой памяти и количества HTTP-запросов на каждом этапе разработки.

Оценка преимуществ CSS

CSS может помочь сократить количество HTTP-запросов и их размер. Существенного сокращения количества HTTP-запросов можно добиться с помощью применения градиентов, скругленных углов границ, блочных и текстовых теней и граничных изображений.

Преимущества, предоставляемые CSS, заключаются в уменьшении числа HTTP-запросов, легкости обновлений и полной масштабируемости эффектов, а также в простых и эффективных переходах, преобразованиях и анимации.

При всех достоинствах CSS[1] прорисовка эффектов на экране требует вполне определенных расходов. Иногда изображения в форматах PNG и JPEG используют меньше памяти и выводятся на экран быстрее эффектов CSS.

Преимущества CSS нужно оценивать взвешенно. Хотя предпочтения зачастую отдают изображениям, создаваемым с помощью CSS, а не экспортируемым и загружаемым изображениям, создаваемым с помощью «Фотошопа», некоторые особенности CSS приводят к скрытым расходам памяти и замедлению визуализации.

Некоторые CSS-свойства требуют для визуализации более весомых затрат по сравнению с другими свойствами. Например, прорисовка размытых теней по сравнению с неоднородным фоновым изображением влечет за собой определение цвета конечного пиксела на основе тени первого плана в сочетании с фоновым цветом для

[1] Многие ненавидят CSS. Они ошибаются!

каждого пиксела. Даже если это внутренняя тень, невидимая из-за сплошного цвета или изображения, помещенного поверх нее в ходе перерисовки, ее пикселы все равно определяются и браузеры выполняют всю работу по формированию общего изображения при прорисовке элементов от задних планов к передним.

CSS-свойства, подвергаемые преобразованию, вычисляются, как правило, при каждом переформатировании и каждой перерисовке, используя при этом память. Изображения в форматах PNG, JPEG и GIF, в отличие от изображений, создаваемых с помощью CSS, выводятся на экран и перемещаются в виде битовых массивов, зачастую используя меньше памяти (но больше HTTP-запросов). Например, тени, особенно внутренние, заново вычисляются при каждой перерисовке, даже если тень закрыта другим элементом, фоновым изображением или эффектом. Сочетание каждого полупрозрачного пиксела с находящимся позади него цветом элемента или эффектом нужно просчитывать для каждого пиксела, каждого эффекта от заднего плана к переднему и при каждой перерисовке.

На создание CSS-градиентов может уйти меньше времени и сил, чем на создание эффекта в «Фотошопе». Созданный с помощью CSS линейный градиент, состоящий из 140 образов, не только занимает меньше байтов трафика по сравнению со своим эквивалентом в формате JPEG, но и экономит HTTP-запросы. Линейный градиент расходует небольшой объем памяти, а растровое изображение, создаваемое браузером, обычно имеет небольшой размер и повторяется.

В то же время радиальный градиент, состоящий из 140 образов и объявленный в CSS, который экономит трафик и HTTP-запросы, может вызвать сбой браузера. Браузер прорисовывает и сохраняет в памяти весь градиент, а не только ту его часть, которая отображается в области просмотра. Если создается небольшой непрозрачный круг, то, безусловно, нужно использовать стандартный радиальный градиент CSS. Но если создается круг большого радиуса, он будет нарисован и за пределами области просмотра, расходуя память. Если вспомнить, слишком большие изображения выкладываются в памяти в виде плитки. Я рекомендую использовать вместо изображений линейные градиенты и стандартные скругленные углы, а вот производительности при использовании радиальных градиентов и внутренних теней нужно давать трезвую оценку, соизмеряя ее с производительностью, достигаемой при загрузке изображения. Последний вариант может иметь более высокую производительность.

Объединение ряда CSS-свойств может вылиться в более продолжительное время прорисовки, превышающее время прорисовки отдельных свойств, которые могут применяться к отдельным DOM-узлам.

Как правило, прорисовка происходит довольно быстро. Действительно быстро. Но время прорисовки нужно учитывать при перерисовках. Каждое переформатирование требует перерисовки. Также перерисовки требует анимация. Если к одному элементу применяются 27 различных эффектов, было бы неплохо просто нарисовать его на странице один раз. Но если элемент подвергается анимации, нужно понимать, что прорисовка некоторых свойств CSS, особенно частично прозрачных компонентов, может занять более 16,67 мс, выделяемых на прорисовку каждого ключевого кадра. Здесь может помочь аппаратное ускорение анимации, но у него тоже есть свои недостатки.

Преимущества и недостатки использования графического процессора

Как упоминалось ранее, аппаратное ускорение может существенно повысить производительность, особенно при анимации. Но перевод всей визуализации в режим `translate3d` не является панацеей от всех бед! Элементы, подвергаемые аппаратному ускорению, приобретают составной характер и занимают в четыре раза больше памяти. Использование графического процессора вместо центрального повысит производительность только до определенного предела. Хотя элементы, выводимые с аппаратным ускорением, используют меньше оперативной памяти, они используют слишком много видеопамяти, поэтому объявление `transform: translateZ(0);` нужно использовать весьма осмотрительно.

Область просмотра: нахождение вне поля зрения еще не означает отсутствия в памяти

Областью просмотра мобильного устройства является просматриваемая область его экрана. В отличие от браузера настольного компьютера, где содержимое можно прокрутить, на мобильном устройстве, если высота и ширина области просмотра не установлены, а масштабирование отключено, область просмотра фиксирована, а пользователь перемещает содержимое под ней. Область просмотра (viewport) является «портом», через который пользователи просматривают содержимое. Но какое отношение эта область имеет к производительности? Большинство людей не понимают, что содержимое, которое выведено на страницу, даже если его не видно в текущей области просмотра, все равно находится в памяти.

Минимизация DOM

При каждом переформатировании пересчитывается каждый DOM-узел. Центральный процессор на настольном компьютере может фактически работать с бесконечным количеством узлов. Ситуация с мобильными устройствами несколько иная. Объем памяти у них ограничен, а механизм сбора «мусора» отличается невысокой надежностью. Для повышения производительности нужно свести количество узлов к минимуму. Вместо распределения и удаления DOM-узлов (забывая иногда их удалять) нужно составлять пул узлов, который затем повторно использовать. Например, в CubeeDoo максимальное количество карт для одной игры составляет 24. Вместо создания новых карт для каждой игры мы создали 24 карты и повторно их используем для новой игры.

CubeeDoo является простым примером повторного использования узлов. Более сложным и востребованным сценарием создания повторно используемых пулов узлов является бесконечная прокрутка. По мере прокрутки страницы информационный канал будет добавлять все новые и новые записи. Со временем браузер или устройство исчерпают всю память. Пользователь будет, как правило, прокручивать страницу не вверх, а вниз. Вместо создания новых узлов для каждой новой записи введите в приложении ограничение на количество узлов, с которым могут справиться все устройства.

По мере прокрутки страницы вниз объединяйте в пул те узлы, которые зашли за верхнюю границу экрана, и используйте их заново для записей, находящихся ниже в информационном канале. Если пользователь начинает прокручивать страницу вверх, берите узлы, которые зашли за нижнюю границу сайта, и используйте их повторно для самых новых записей.

Большинство информационных каналов, разработанных для браузеров настольных систем, не предусматривает объединений в пулы и повторного использования. Бесконечная прокрутка без объединения в пулы и повторного использования может привести (и приводит) к аварийному отказу браузеров. У меня была возможность заставить Facebook задействовать 76 Мбайт памяти путем прокрутки вниз своего информационного канала. Со временем это привело к аварийному отказу браузера настольного компьютера, хотя я не уверена в том, что она была вызвана потреблением памяти бесконечно создаваемыми узлами или возникновением каких-то других проблем. Скорее всего, такая ситуация привела бы и к аварийному отказу мобильного браузера.

Вместо добавления все новых и новых узлов для дополнительного содержимого по мере прокрутки страницы вниз ограничьте количество узлов с содержимым в своем приложении. Реализуйте повторное использование верхних узлов по мере прокрутки страницы вниз и нижних узлов — по мере ее прокрутки вверх.

Управление памятью

Инструментарий разработчика дает нам средства для анализа и исследования потребления памяти приложением.

Панель Timeline (Временная шкала) предоставляет обзор затрат времени при загрузке сайта и взаимодействия с ним или с веб-приложением. На шкалу времени наносятся все события, от загрузки ресурсов до разбора кода JavaScript, вычисления стилей и перерисовки. На шкале отображаются события, включая вычисление стилей, переформатирование и перерисовку, разбор и выполнение кода JavaScript и т. д.

Информацию, предоставляемую панелью Timeline средства Chrome Developer Tools (рис. 14.4), можно применять для управления использованием памяти. Чтобы проверить расход памяти, нужно выбрать пункт меню Memory на верхней левой панели, а затем запустить перехват памяти и событий, щелкнув на черном кружке значка записи, который находится на нижней панели инструментов. При запуске записи этот значок станет красным. Если нужно лишь посмотреть на текущее использование памяти, то просто запустите запись, а если нужно проинспектировать все события, которые становятся причиной использования памяти, щелкните на значке записи и перезагрузите страницу.

В текущей версии средства Chrome Developer Tools время, затраченное на загрузку, обозначается синим цветом, на работу со сценариями — желтым, на визуализацию — фиолетовым, а на прорисовку — зеленым. Отображение каждого из этих типов событий можно включать и выключать, используя флажки в строке состояния в нижней части окна. Вы также можете отфильтровать события продолжительностью менее 1 мс или 15 мс из меню, раскрываемого щелчком на кнопке фильтров All (Все) в строке состояния, расположенной слева от флажков событий. Я рекомендую включать фильтрацию для отстройки от шумов при выявлении причин снижения производительности.

В представлении Memory (Память) в узкой области в самом верху отображается время, а синие и красные вертикальные линии являются индикаторами инициирования событий DOMContentLoaded и loaded в ходе загрузки страницы.

Событие DOMContentLoaded выдается при загрузке разметки, CSS и блоков кода JavaScript, и с этого момента браузер приступает к визуализации страницы. Чтобы повысить производительность, вы должны минимизировать время процессов, результатом завершения которых является событие DOMContentLoaded, и, что может быть важнее, время между событиями DOMContentLoaded и onLoad.

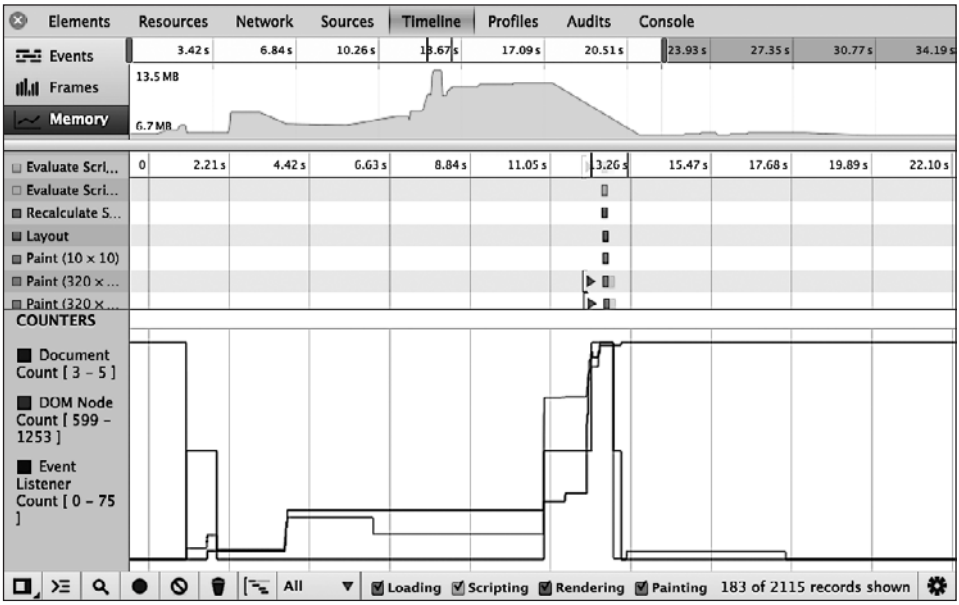


Рис. 14.4. Панель памяти средства Google Chrome Timeline

Если осуществляется «ленивая» загрузка, то после выдачи события onLoad могут произойти дополнительные загрузки. Это нормально. Ощущаемое время загрузки, которое обычно приходится на промежуток между этими двумя событиями, все-таки ближе к событию onLoad, и это важно учесть. Клиентов отпугивают недостатки, связанные именно с этим фактором. Фактическая продолжительность загрузки может оказаться немного больше, но ваши пользователи об этом не знают. Нужно, чтобы они увидели содержимое и начали работать с ним как можно быстрее. Если значки социальных сетей и колонтитул страницы будут «лениво» загружаться где-то за «изгибом» страницы, то пользователи это вряд ли заметят.

ПРИМЕЧАНИЕ

Если вы решили включить значки социальных сетей, попробуйте воспользоваться простыми ссылками на эти службы, не применяя JavaScript API-функции, предоставляемые сайтами социальных сетей. Многие виджеты социальных сетей снижают производительность.

Водопадный график можно сохранять в файле формата HAR JSON, чтобы сравнивать загрузки страницы по мере оптимизации сайта.

В остальном пространстве верхней части представлена информация по общему использованию памяти открытым в данный момент сайтом или приложением. Можно заметить, что потребление памяти обычно возрастает по мере роста количества DOM-узлов и снижается после сбора «мусора».

В средней части основной области панели перечисляются все события, включая связанные с загрузкой, сценариями, визуализацией и прорисовкой. Каждая запись представляет информацию о таких характеристиках, как продолжительность, время использования центрального процессора и строка кода, вызвавшая событие. Для каждого события показывается общее время загрузки, выполнения сценария, визуализации и прорисовки. Таким образом, можно установить, какие из событий негативно влияют на производительность.

Счетчик в нижней части основной области представляет такую статистику, как количество DOM-узлов, прослушивателей событий и документов, имеющихся в приложении в каждый момент времени.

Обращение к DOM с чтением или записью с точки зрения производительности обходится довольно дорого. Для повышения производительности нужно кэшировать поиски в DOM и сохранять их результаты в переменных. Нужно также собирать в отдельные пакеты DOM-запросы и манипуляции с DOM, сводя к минимуму работу с DOM путем обновления содержимого вне ее, перед тем как обновить объектную модель документа.

Когда дело касается управления памятью, то оптимизация изображений, визуализация CSS и количество DOM-узлов не являются основными поводами для беспокойства. При решении проблем производительности это всего лишь пункты, которые в области настольных компьютеров не входят в сферу обязательного рассмотрения.

Отзывчивость пользовательского интерфейса

Мобильные браузеры запускаются в одном потоке. Этим они похожи на браузеры настольных компьютеров. Но отзывчивость пользовательского интерфейса не сводится лишь к факту использования одного потока.

Сенсорные события

Из-за задержек браузер после выбора того или иного действия может зависать, поскольку на круговое путешествие к серверу и обратно нужно определенное время. Важно в течение 200 мс после того, как действие было предпринято, дать пользователю какую-нибудь ответную реакцию, и лучше сделать это как можно быстрее.

При показе или скрытии элемента ответную реакцию демонстрировать не имеет смысла, поскольку приложение будет способно реагировать на эти действия. Но чтобы показать, что сайт отзывается, когда пользователь вынужден ожидать

завершения кругового путешествия с целью обновления пользовательского интерфейса, какую-нибудь ответную реакцию все же нужно обеспечить. Например, чтобы проинформировать пользователя о том, что его действие было воспринято, выключите кнопку отправки, убрав ее с отправляемой формы. Если AJAX-запрос может занять более 200 мс, предоставьте пользователю какой-нибудь вращающийся индикатор, индикатор выполнения или даже анимированный подпрыгивающий мячик. Позвольте своей команде взаимодействия с пользователем определить, какая ответная реакция подойдет больше, но убедитесь в том, что на время ожидания пользователем ответа на его действия на вашем сайте эта ответная реакция ему предоставлена.

Кроме того, поскольку мобильные устройства имеют сенсор и пользователь вполне может применить двойное касание, мобильное устройство, как правило, прежде чем отозваться на прикосновение, ждет потенциально возможного двойного касания. По умолчанию это ожидание после наступления события `touchend` и до выполнения какого-либо действия на большинстве мобильных устройств занимает от 300 до 500 мс. Поэтому может появиться потребность повысить отзывчивость приложения, взяв на себя обработку таких исходных событий, как прикосновения, путем добавления прослушивателя к событию `touchend`.

ПРИМЕЧАНИЕ

При добавлении сенсорных событий не следует удалять события щелчка: ваш сайт должен работать независимо от того, что именно выберет пользователь для взаимодействия с ним — пальцы, мышь или какой-нибудь другой метод взаимодействия.

Задержка при ожидании возможного двойного касания предусмотрена не для всех сценариев развития событий[1]. В Chrome и Firefox, если масштабирование отключено, задержки не будет. Не стоит вводить в практику возможное избавление от этой задержки путем запрещения масштабирования с помощью метатега: не следует запрещать масштабирование без серьезной причины, примером которой может стать интерактивная игра.

Анимация

Поскольку веб-приложения выполняются в одном потоке, а код JavaScript имеет приоритет над CSS-анимацией, для второстепенной анимации вместо JavaScript всегда следует применять CSS.

Из-за слишком низкого уровня приоритета CSS-анимации она не начнется до тех пор, пока страница не будет загружена, поскольку поток пользовательского интерфейса занят парсингом сценариев и визуализацией. Хотя анимация не может начаться, счетчик задержки анимации `animation-delay` не ждет загрузки страницы. При наличии множества анимаций, начинающихся после различных задержек, можно заметить, что несколько анимаций могут начаться одновременно с окончанием загрузки страницы,

[1] В настоящее время предотвращение масштабирования в функционально ограниченных браузерах является одной из причин того, что браузеры, распознающие сенсорные события, не ожидают возможного второго прикосновения. Но в будущем их функционал может быть расширен.

поскольку анимации элементов с более короткими, чем время загрузки страницы, задержками начнутся в одно и то же время.

Как отмечалось ранее, самая плавная анимация на большинстве устройств выполняется со скоростью 60 кадров/с. Это предполагает выполнение всех вычислений и прорисовки за 16,67 мс. Чтобы анимация была плавной, вычисления и прорисовка узлов должны осуществляться менее чем за 16,67 мс.

Резюме

Список рассмотренных в данной главе тем, касающихся обеспечения высокой производительности пользовательского интерфейса мобильных систем, конечно, неисчерпывающий, но он должен послужить неплохой отправной точкой. При весьма высокой скорости, с которой происходит обновление браузеров мобильных и настольных устройств, поднятые здесь темы, скорее всего, уже не совсем актуальны. Но все рекомендации даны на основе практического опыта. Хотя жизнь не стоит на месте и, может быть, уже приобретен новый полезный опыт, позволивший решить и некоторые упомянутые здесь вопросы, следование данным рекомендациям будет, наверное, вполне разумным поступком и в обозримом будущем.

Следует помнить, что наиболее быстро растущий сегмент пользователей — это мобильные пользователи. И их не нужно игнорировать. Эти рекомендации нетрудно реализовать без нанесения ущерба пользователям браузеров настольных устройств. Поэтому я призываю придерживаться рекомендаций в отношении всех сайтов, даже если доля их посетителей, использующих мобильные устройства, невелика. В конечном счете вы не можете знать, из-за чего доля посетителей с мобильными устройствами так низка: из-за того, что ваша аудитория не пользуется мобильными устройствами (что маловероятно), или из-за того, что у пользователей мобильных устройств осталось плохое впечатление от вашего сайта (что более вероятно).

Будучи разработчиками, мы тестируем наши сайты, чтобы убедиться в том, что следуем всем пунктам и установкам, рекомендованным средством YSlow компании Yahoo! и средством PageSpeed компании Google. И все эти многократные тестирования проводим в браузерах настольных компьютеров. Мы считаем, что основные положения оптимизации веб-производительности повышают производительность веб-приложения во всех браузерах, независимо от того, что именно применяют наши пользователи для обращения к этому веб-приложению — ноутбук, iPad, телефон с операционной системой Android или даже свою приставку Wii. Но при этом следует помнить, что применительно к мобильным устройствам нужно пользоваться не только широко известными и повсеместно принимаемыми во внимание рекомендациями.

Продолжайте тестирование своего сайта, но теперь уже на мобильных устройствах. Эмуляторы не могут в точности повторить эти устройства. Эмуляторы не могут смоделировать ограничения памяти и устройство, имеющее 100 открытых приложений. Проводите тесты с имеющими определенные пределы памятью и пропускной способностью. Проводите тесты на реальных устройствах и по реальным сценариям: отключайте Wi-Fi и тестируйте при множестве незакрытых приложений, подвисших в фоновом режиме. Тестируйте. Тестируйте. Тестируйте.

Приложение

CSS-селекторы и уровни конкретизации

CSS-селекторы уровня 3

Образец	Значение	Уровень конкретизации и примеры
Универсальный селектор		0-0-0
С точки зрения конкретизации универсальный селектор никакого веса не имеет		
*	Соответствует любому элементу	* {}
Селектор типа		
Селектор типа или элемента имеет самый низкий уровень конкретизации		0-0-1
E	Соответствует элементам типа E	em, strong
Селектор класса		0-1-0
myClass	Соответствует всем элементам, чей список классов содержит класс myClass	.myClass
Селектор идентификатора		1-0-0
#myId	Соответствует элементу, у которого атрибут ID равен myId	#myId
Комбинаторы Комбинаторы, включая >, + и ~, не влияющие на конкретизацию		0-0-0
E F	Соответствует элементам F, являющимся потомками (непосредственными или иными дочерними элементами) элемента E	ol li tr td
E > F	Соответствует элементам F, являющимся непосредственными дочерними элементами элемента E	ol > li thead > tr
E + F	Соответствует элементу F, который следует непосредственно после элемента E, если E и F имеют общего родителя	h1 + p tr.current + tr
E ~ F	Соответствует всем элементам F, которые следуют после E и у которых тот же самый родительский элемент	li:first-child ~ li
Селекторы атрибута		0-1-0
Селекторы атрибута имеют такой же уровень конкретизации, что и у селектора класса		

Продолжение ⇩

(Продолжение)

Образец	**Значение**	**Уровень конкретизации и примеры**
E[attr]	Соответствует элементам E, имеющим атрибут attr, независимо от значения атрибута	input[type]
E[attr="val"]	Соответствует элементам E, у которых значение атрибута attr в точности равно val	input[type="checkbox"]
E[attr~="val"]	Соответствует элементам E, чьи значения атрибута attr представлены списком значений с пробелом в качестве разделителя, где одно из значений в точности соответствует значению val	img[alt~="figure"]
E[attr^="val"]	Соответствует элементам E, чье значение атрибута attr начинается в точности со строки val	a[href^="mailto:"]
E[attr$="val"]	Соответствует элементам E, чье значение атрибута attr заканчивается в точности строкой val	a[href$=".pdf"]
E[attr*="val"]	Соответствует элементам E, чье значение атрибута attr содержит подстроку val	a[href*="://"] a[href*="twitter.com"]
E[attr|="val"]	Соответствует элементам E, чье значение атрибута attr равно val или начинается с подстроки val, за которой следует дефис	html[lang|="en"]
Структурные псевдоклассы		0-1-0
У псевдоклассов такой же уровень конкретизации, как и у селектора класса		
E:first-child	Соответствует элементу E, который является первым дочерним элементом своего родительского элемента	h1:first-child
E:last-child	Соответствует элементу E, который является последним дочерним элементом своего родительского элемента	p:last-child
E:only-child	Соответствует элементу E лишь в том случае, если E является единственным дочерним элементом своего родительского элемента	li:only-child
E:first-of-type	Соответствует элементу E, который является первым элементом E своего типа и не обязательно первым дочерним элементом	li:first-of-type
E:last-of-type	Соответствует элементу E, который является последним элементом E своего типа и не обязательно последним дочерним элементом	li:last-of-type
E:only-of-type	Соответствует элементу E, если E является единственным дочерним элементом своего родительского элемента данного	h1:only-of-type

Образец	Значение	Уровень конкретизации и примеры
	типа и не обязательно единственным дочерним элементом родительского элемента	
E:nth-child(n)	Соответствует элементу (элементам) E, являющемуся *n*-м дочерним элементом своего родительского элемента, где *n* может быть целым числом, уравнением, соответствующим a*n* + b, где a является множителем, a b — смещением, или ключевым словом even или odd	tr:nth-child(odd)
E:nth-last-child(n)	Соответствует элементу (элементам) E, являющемуся *n*-м дочерним элементом своего родительского элемента, вычисляемому от последнего дочернего элемента в обратном направлении	li:nth-last-child(5)
E:nth-of-type(n)	Соответствует элементу (элементам) E, являющемуся *n*-м элементом среди элементов одного уровня (имеющих один и тот же родительский элемент) их типа	th:nth-of-type(2)
E:nth-last-of-type(n)	Соответствует элементу (элементам) E, являющемуся n-м элементом среди элементов одного уровня их типа, вычисляемому от последнего элемента E	
E:root	Соответствует элементу E, если он является корневым элементом документа, которым в наших HTML-документах всегда является элемент HTML	html:root
E:empty	Соответствует элементу E, если E является пустым элементом, не имеющим никаких дочерних элементов, кроме комментария. Если в элементе содержится хотя бы один пробел, он пустым не считается	p:empty
Псевдоклассы ссылок, действий пользователя и пользовательского интерфейса		0-1-0
Эти псевдоклассы инициируются тем или иным состоянием и имеют ту же конкретизацию, что и селектор класса		
E:link E:visited	Псевдоклассы link соответствуют тем гиперссылкам E, которые еще не были посещены (:link) или уже были посещены (:visited)	a:link a:visited
E:active E:hover E:focus	Псевдоклассы пользовательского действия соответствуют элементу (элементам) E в ходе конкретных действий пользователя, когда элемент активен, над ним находится указатель или он имеет фокус	a:active img:hover input:focus
E:enabled E:disabled	Соответствует элементу пользовательского интерфейса E, который включен (enabled) или выключен (disabled)	input:enabled select:disabled

Продолжение ⇩

(Продолжение)

Образец	Значение	Уровень конкретизации и примеры
E:checked	Соответствует элементу пользовательского интерфейса E, такому как переключатель или флажок, который был установлен	input[type="radio"] :checked
E:default	Соответствует элементу E, который является выбранным по умолчанию среди набора подобных ему элементов, например исходных настроек, выбранных при загрузке страницы	option:default
E:valid E:invalid	Соответствует элементу E, когда значение элемента приемлемо (valid) или неприемлемо (invalid), например, когда оно соответствует или не соответствует атрибуту pattern или type тега input	input:valid input:invalid
E:in-range E:out-of-range	Соответствует элементу E, если этот элемент имеет ограничения по диапазону, например диапазону вводимых числовых значений с минимальными и максимальными значениями, и введенное значение находится либо в указанном диапазоне (:in-range), либо за его пределами (:out-of-range)	input:in-range input:out-of-range
E:required E:optional	Соответствует элементу поля формы E, если это либо обязательный (:required), либо необязательный (:optional) элемент	input:required input:optional
E:read-only E:read-write	Соответствует элементу E, если его содержимое либо не подлежит изменению пользователем (:read-only), либо подлежит (:read-write), к примеру, это могут быть поля ввода текста	input:read-only input:read-write
Цель или язык		
E:target	Элемент E, служащий целью ссылающегося на него URI-идентификатора	div:target
E:lang(fr)	Элемент типа E на языке fr (на порядок определения языка указывает язык документа)	p:lang(fr)
Отрицание		?-?-? (зависит от параметра)
E:not(exclude)	Соответствует всем элементам E, которые не соответствуют селектору exclude. Отрицание :not с точки зрения конкретизации веса не имеет, а к весу добавляется содержимое аргумента	div:not([class]) .foo:not(div)
Псевдоэлементы		0-0-1
E::first-line	Соответствует первой отформатированной строке элемента E	p::first-line
E::first-letter	Соответствует первой отформатированной букве элемента E	p::first-letter

Образец	Значение	Уровень конкретизации и примеры
E::before	Генерирует содержимое перед содержимым элемента E и соответствует этому содержимому	div::before
E::after	Генерирует содержимое после содержимого элемента E и соответствует этому содержимому	div::after
E::selection	Этот пока что отсутствующий в спецификации псевдоэлемент соответствует содержимому элемента E, который в данный момент выбран или выделен пользователем	*::selection *::-moz-selection

Памятка по селекторам CSS

*	::after	:empty
E	::first-letter	:not()
.class	::first-line	:target
#id	E[attribute^=value]	:enabled
E F	E[attribute$=value]	:disabled
E > F	E[attribute*=value]	:checked
E + F	E ~ F	:indeterminate[4]
E[attribute]	:root	:default
E[attribute=value]	:last-child	:valid
E[attribute~=value]	:only-child	:invalid
E[attribute|=value]	:nth-child()	:in-range
:first-child	:nth-last-child()	:out-of-range
:link[1]	:first-of-type	:required
:visited	:last-of-type	:optional
:lang()	:only-of-type	:read-only
::before[2]	:nth-of-type()	:read-write
::selection[3]	:nth-last-of-type()	

[1] В некоторых браузерах по соображениям безопасности поддержка :link и :visited ограничена.

[2] Для поддержки в устаревших версиях IE следует использовать запись с одинарным двоеточием.

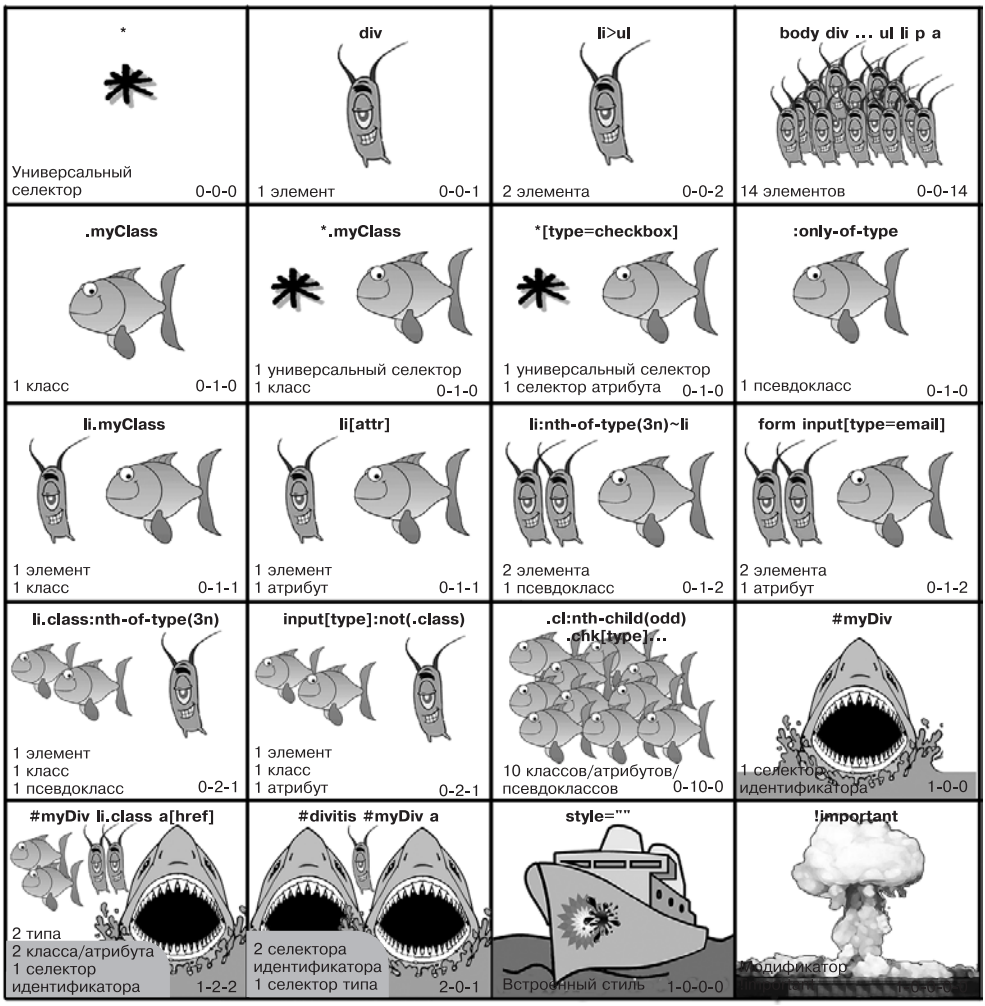
[3] Не входит в спецификацию CSS Selectors level 3, но полностью поддерживается. Для Firefox должен указываться префикс -moz-.

[4] Последние девять селекторов являются частью спецификации основного модуля пользовательского интерфейса уровня 3 — CSS Basic User Interface Module Level 3 (CSS3 UI) specification — и находятся в спецификации CSS Selectors Level 4.

Конкретизация CSS-селекторов

Конкретизация CSS на примере морского царства

с планктоном, рыбами и акулами



x-0-0: Количество селекторов идентификаторов.
0-y-0: Количество селекторов класса.
0-0-z: Количество селекторов типа и псевдоэлементов.
*,+,>,~: Универсальный селектор не имеет значения,
а комбинаторы не повышают уровень конкретизации.
:not(x): Селектор отрицания не имеет значения, но переданный ему аргумент
повышает уровень конкретизации.

CSS-селекторы уровня 4

Селектор	Определение	Уровень
Основные селекторы		
*	Универсальный селектор, соответствует всем элементам	2
E	Селектор типа (имени тега), соответствует элементам типа E	1
.someClass	Селекторы класса, соответствуют элементам, имеющим перечисленный класс, в данном случае someClass	1
#myID	Селектор идентификатора, соответствует элементу с идентификатором, равным myID	1
Комбинаторы		
E F	Комбинатор потомка, соответствует элементу F, который является потомком элемента E	1
E > F	Комбинатор дочернего элемента, соответствует элементу F, который является дочерним элементом элемента E	2
E + F	Комбинатор следующего элемента того же уровня, соответствует элементу F, которому непосредственно предшествует элемент E	2
E ~ F	Комбинатор следующих элементов того же уровня, соответствует элементам F, которым предшествует элемент E	3
E /foo/ F	Комбинатор ссылки, соответствует элементу F, на который по идентификатору ссылается атрибут foo элемента E (будет соответствовать имеющемуся в форме элементу F, на который была ссылка в атрибуте foo элемента E, являющегося тегом label)	4
E! > F	Определяет объект селектора плюс комбинатор дочернего элемента, соответствующий элементу E, который является родительским элементом элемента F	4
Селекторы атрибута		
E[foo]	Соответствует элементу E, у которого есть атрибут foo	2
E[foo="bar"]	Соответствует элементу E, чей атрибут foo имеет значение, равное bar, чувствительность к регистру символов зависит от такой же чувствительности значения атрибута	2
E[foo="bar" i]	Соответствует элементу E, чей атрибут foo имеет значение, равное bar, в любом варианте написания с точки зрения регистров символов	4
E[foo~="bar"]	Соответствует элементу E, чей атрибут foo имеет значение, состоящее из списка значений, разделенных пробелами, в котором одно из значений равно bar	2
E[foo^="bar"]	Соответствует элементу E, чей атрибут foo имеет значение, начинающееся в точности со строки bar	3
E[foo$="bar"]	Соответствует элементу E, чей атрибут foo имеет значение, заканчивающееся в точности строкой bar	3
E[foo*="bar"]	Соответствует элементу E, чей атрибут foo имеет значение, содержащее подстроку bar	3
E[foo|="en"]	Соответствует элементу E, чей атрибут foo имеет значение, состоящее из списка значений, разделенных дефисом, и начинающееся с en	2
Структурные псевдоклассы		
E:root	Соответствует элементу E, который является корневым элементом документа	3

Продолжение ⇩

(Продолжение)

Селектор	Определение	Уровень
E:empty	Соответствует элементу E, у которого нет дочерних элементов (нет даже текстовых узлов)	3
E:blank	Соответствует элементу E, у которого нет содержимого, за исключением пробела	4
E:first-child	Соответствует элементу E, который является первым дочерним элементом своего родительского элемента	2
E:last-child	Соответствует элементу E, который является последним дочерним элементом своего родительского элемента	3
E:only-child	Соответствует элементу E, который является единственным дочерним элементом своего родительского элемента	3
E:first-of-type	Соответствует элементу E, который является первым элементом этого типа среди элементов одного уровня	3
E:last-of-type	Соответствует элементу E, который является последним элементом этого типа среди элементов одного уровня	3
E:only-of-type	Соответствует элементу E, который является единственным элементом этого типа среди элементов одного уровня	3
E:nth-child(n)	Соответствует элементу E, который является *n*-м дочерним элементом своего родительского элемента	3
E:nth-last-child(n)	Соответствует элементу E, который является *n*-м дочерним элементом своего родительского элемента, вычисляемым от последнего элемента	3
E:nth-of-type(n)	Соответствует элементу E, который является *n*-м элементом этого типа среди элементов одного уровня	3
E:nth-last-of-type(n)	Соответствует элементу E, который является *n*-м элементом этого типа среди элементов одного уровня, вычисляемым от последнего элемента	3
E:nth-match(n селектора)	Соответствует элементу E, который является *n*-м элементом среди элементов одного уровня, соответствующих селектору	4
E:nth-last-match(n селектора)	Соответствует элементу E, который является *n*-м элементом среди элементов одного уровня, соответствующих селектору и вычисляемых от последнего элемента	4
Grid-структурные псевдоклассы		
F || E	Соответствует элементу E, который представляет ячейку grid-структуры или таблицы, принадлежащей столбцу, представленному элементом F	4
E:nth-column(n)	Соответствует элементу E, который представляет ячейку, принадлежащую *n*-му столбцу в grid-структуре или таблице	4
E:nth-last-column(n)	Соответствует элементу E, который представляет ячейку, принадлежащую *n*-му столбцу в grid-структуре или таблице, вычисляемому с последнего столбца	4
Псевдоклассы ссылок		
E:any-link	Соответствует элементу E, который является исходным указателем гиперссылки	4
E:link	Соответствует элементу E, который является исходным указателем гиперссылки, по которой целевой ресурс еще не был посещен	1
E:visited	Соответствует элементу E который является исходным указателем гиперссылки, по которой целевой ресурс уже был посещен	1

Селектор	Определение	Уровень
E:local-link	Соответствует элементу E, который является исходным указателем гиперссылки, по которой целевой ресурс находится в текущем документе	4
E:local-link(0)	Соответствует элементу E, который является исходным указателем гиперссылки, по которой целевой ресурс находится в текущем домене, но не обязательно в текущем документе	4
E:target	Псевдокласс target соответствует элементу E, который является той целью, на которую ссылается URL-адрес	3
Псевдоклассы пользовательского интерфейса		
E:active	Соответствует элементу E, который находится в активизированном состоянии	1
E:hover	Соответствует элементу E, который находится под указателем или является потомком элемента, находящегося под указателем	2
E:focus	Соответствует элементу E, который находится в фокусе пользовательского ввода	2
E:enabled	Соответствует элементу пользовательского интерфейса E, который находится во включенном состоянии	3
E:disabled	Соответствует элементу пользовательского интерфейса E, который находится в выключенном состоянии	3
E:read-only	Соответствует элементу пользовательского интерфейса E, который не подлежит редактированию	3/4
E:read-write	Соответствует элементу пользовательского интерфейса E, который подлежит редактированию, и элементу E, имеющему атрибут contenteditable, установленный в true	3/4
E:placeholder-shown	Соответствует элементу управления вводом E, который в данный момент показывает текст «заполнителя»	3/4
E:default	Соответствует элементу пользовательского интерфейса E, который был настройкой, выбираемой по умолчанию	3/4
E:checked	Соответствует элементу пользовательского интерфейса E, на котором был щелчок или который был выбран, например установленному флажку или выбранному элементу переключателя	3
E:indeterminate	Соответствует элементу пользовательского интерфейса E, который находится в неопределенном состоянии (ни в выбранном, ни в невыбранном)	4
E:valid	Соответствует элементу пользовательского ввода E, который считается приемлемым из-за отсутствия ограничений приемлемости (всегда приемлемым) или из-за того, что его содержимое соответствует ограничениям приемлемости	3/4
E:invalid	Соответствует элементу пользовательского ввода E, который считается неприемлемым, когда его содержимое не соответствует ограничениям приемлемости, выраженным в атрибутах	3/4
E:in-range	Соответствует элементу пользовательского ввода E, чье значение вписывается в диапазон, например в границы минимального и максимального значений	3/4
E:out-of-range	Соответствует элементу пользовательского ввода E, чье значение не вписывается в диапазон, например выходит за границы минимального и максимального значений	3/4
E:required	Соответствует элементу пользовательского ввода E с обязательным для ввода значением (не может быть оставлен пустым)	3/4

Продолжение ⇩

(Продолжение)

Селектор	Определение	Уровень
E:optional	Соответствует элементу пользовательского ввода E с необязательным для ввода значением (может быть оставлен пустым)	3/4
Псевдоклассы перетаскивания (drag-and-drop)		
E:active-drop	Соответствует элементу E, который получит перетаскиваемый элемент	
E:valid-drop	Соответствует элементу E, который может получить текущий перетаскиваемый элемент	
E:invalid-drop	Соответствует элементу E, который не может получить текущий перетаскиваемый элемент, но может получить какой-нибудь другой элемент	
Псевдоклассы соответствия, отрицания и области видимости		
E:not(s1, s2)	Соответствует элементам E, которые не соответствуют ни составному селектору s1, ни составному селектору s2. В CSS Level 3 может быть передан только один простой селектор	3/4
E:matches(s1, s2)	Соответствует элементам E, которые соответствуют составному селектору s1 и/или составному селектору s2	4
E:scope	Псевдокласс scope соответствует элементу E, который является намеченным элементом ссылки	4
Псевдоклассы языка и направления		
E:dir(ltr) E:dir(rtl)	Соответствует элементам E с направлением чтения слева направо или справа налево на основании языка документа	4
E:lang(zh, *-hant)	Соответствует элементам E, нацеливание на которые происходит потому, что они на китайском (на любом диалекте или в любой системе записи) или же написаны традиционными китайскими символами. В CSS-селекторах уровня 2 (CSS Selectors Level 2) псевдокласс :lang() воспринимает только один параметр	2/4
Псевдоклассы на шкале времени (Time-dimensional pseudoclasses)		
E:current	Соответствует элементу E, который в данный момент присутствует на шкале времени	4
E:current(s)	Соответствует элементу E, который на шкале времени является самым отдаленным элементом :current, соответствующим селектору s	4
E:past	Соответствует элементу E, который на шкале времени находится в прошлом	4
E:future	Соответствует элементу E, который на шкале времени находится в будущем	4