Бен Фрэйн

HTML5 и CSS3

Разработка сайтов для любых браузеров и устройств

2-е издание

[PACKT] PUBLISHING

ПИТЕР®

Ben Frain

Responsive Web Design with HTML5 and CSS3

Second Edition

[PACKT]
PUBLISHING
BIRMINGHAM - MUMBAI

Бен Фрэйн

HTML5 и CSS3

Разработка сайтов для любых браузеров и устройств

2-е издание

ПИТЕР®

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

2017

Б. Фрэйн

HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств

2-е издание

Серия «Библиотека программиста»

Перевел с английского *Н. Вильчинский*

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*О. Сивченко*
Литературный редактор	*Н. Рощина*
Художник	*С. Заматевская*
Корректоры	*Н. Гринчик, Е. Павлович*
Верстка	*А. Барцевич*

ББК 32.988.02-018
УДК 004.738.5

Фрэйн Б.

Ф86 HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. 2-е изд. — СПб.: Питер, 2017. — 272 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-02271-2

Сегодня как никогда остро стоит проблема адаптивного веб-дизайна. Все больше планшетных компьютеров, смартфонов и даже телевизоров используются для выхода в Интернет. Разработчикам веб-страниц требуется принимать во внимание огромное разнообразие размеров экранов, а также учитывать особенности соответствующего пользовательского взаимодействия. Адаптивный веб-дизайн позволяет наилучшим образом отобразить содержимое сайтов на экранах устройств, используемых для просмотра. При этом веб-страницы будут хорошо смотреться на дисплеях не только современных устройств, но и тех, что появятся в ближайшее время.

Начните разрабатывать сайты в соответствии с новой методологией адаптивного веб-дизайна, благодаря чему они будут красиво отображаться на экранах любых размеров. Читайте эту книгу, попутно создавая и улучшая адаптивные веб-дизайны с использованием HTML5 и CSS3. Вы научитесь применять на практике новые технологии и методики, призванные стать инструментами будущего для веб-разработчиков клиентских приложений.

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

ISBN 978-1784398934 англ.
ISBN 978-5-496-02271-2

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.
Подписано в печать 10.08.16. Формат 70x100/16. Бумага офсетная. Усл. п. л. 21,930. Тираж 1000. Заказ 0000.
Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru Факс: 8(496) 726-54-10, телефон: (495) 988-63-87

Краткое оглавление

Оглавление

Об авторе

Бен Фрэйн (Ben Frain) занимается веб-дизайном и разработкой веб-приложений с 1996 года. В настоящее время он работает ведущим разработчиком пользовательского интерфейса в компании Bet365.

До того как заняться веб-технологиями, работал скромным, не получившим должного признания актером на телевидении и техническим писателем. Окончил Салфордский университет, получив высшее образование в области театрального искусства.

Написал четыре так и не оцененных по достоинству (по его мнению) киносценария и до сих пор вынашивает, правда уже с угасающим энтузиазмом, планы продать хотя бы один из них. Вне работы предается весьма простым увлечениям. Играет в мини-футбол, пока его физическое состояние и жена не препятствуют этому занятию, и занимается воспитанием двух сыновей.

В настоящее время доступна еще одна книга, вышедшая из-под его пера, — *Sass and Compass for Designers*. О его делах можно узнать на сайте www.benfrain.com или в «Твиттере» — на страничке twitter.com/benfrain.

Хочу поблагодарить технических рецензентов книги за то, что они потратили свое свободное время на внесение ценного вклада в этот труд. Спасибо им за повышение качества результата.

Хочу также поблагодарить веб-сообщество в целом за постоянный обмен информацией. Без всех вас я не смог бы состояться как веб-разработчик.

А главное, хочу выразить признательность своей семье. Могу привести множество примеров жертв, приносимых ими ради предоставления мне возможности работать над книгой, включая приглушенный звук телевизора (от жены), чашку чая (от родителей) и прекращение пиратских боев на мечах (от сыновей).

О рецензентах

Эстебан С. Эбэйт (Esteban S. Abait) — разработчик архитектуры программных продуктов и бывший аспирант. Имеет опыт разработки архитектуры сложных программных продуктов и планирования работы по их созданию. Был штатным и внештатным работником в компаниях Cisco, Intuit и Southwest. Работал с различными технологиями, в частности Java, PHP, Ruby и Node.js. В последние годы проявлял особый интерес к разработке веб-приложений, приложений для мобильных устройств и к REST API-интерфейсам. Разработал несколько довольно объемных веб-приложений с использованием JavaScript. Кроме того, он создавал методические рекомендации по REST-технологиям на основе оценок клиентов. Разрабатывал сайты с высоким объемом трафика, в которых использовал такие ключевые для масштабирования технологии, как репликация, сегментирование и распределенное кэширование данных.

В настоящее время Эстебан работает главным инженером компании Globant. В этой должности он добивается сдачи проектов в срок с обеспечением наивысшего качества конечного продукта. Он также занимается разработкой обучающих курсов по созданию программных продуктов и проводит совещания с разработчиками программных средств. Кроме того, Эстебан работает выездным консультантом по вопросам веб-технологий.

Компания Globant (http://www.globant.com/) относится к новому поколению поставщиков технологических услуг, нацеленных на выработку инновационных программных решений с использованием новых технологий и тенденций. Globant сочетает в своей деятельности жесткость в вопросах проектирования и технических решений, характерную для поставщиков сервисов информационных технологий, с творческим и культурным подходом, присущим агентствам по внедрению цифровых технологий. В компании Globant выдерживаются четкие пропорции инженерных наработок, дизайна и инноваций.

Кристофер Скотт Эрнандес (Christopher Scott Hernandez), будучи по профессии дизайнером, стал разработчиком веб-технологий, начав с того, что в 1996 году создал для своего отца первый в Интернете сайт по обивке катеров. После этого он стал передавать свой опыт мелким и крупным компаниям, поработав над наиболее посещаемыми в мире сайтами, включая eBay, LinkedIn и Apple.

Был техническим редактором книги *HTML5 Multimedia Development Cookbook*, вышедшей в Packt Publishing. Является заядлым читателем и любителем книг. Когда не занимается расстановкой пикселов и написанием кода, любит проводить время с женой и дочерью, знакомясь с парками и пешеходными тропами красивого города Остина в штате Техас.

Мовис Ледфорд (Mauvis Ledford) занимается комплексным созданием и техническим руководством компаний, специализирующихся в области веб-технологий,

мобильных веб-технологий и создания приложений, масштабируемых в облачных сервисах.

Работал в компаниях Disney Mobile, Skype, Netflix, и многих других молодых компаниях в Сан-Франциско, Нью-Йорке и их окрестностях. В настоящее время он является техническим директором в недавно созданной компании Pathbrite, занимающейся разработкой образовательных технологий и специализирующейся на создании бесплатных адаптивных мультимедийных электронных портфолио и цифровых резюме для всех желающих. Вы также можете воспользоваться ее услугами, если зайдете на сайт http://www.pathbrite.com.

Мовис также был техническим редактором первого издания книги *Responsive Web Design with HTML5 and CSS3*, вышедшей в Packt Publishing, и книги *Building Hybrid Android Apps with Java and JavaScript*, вышедшей в O'Reilly Media.

Софи Уильямс (Sophie Williams) — весьма взыскательный человек, увлекающийся дизайном. Имеет ученую степень в области графического дизайна и в настоящее время работает веб-дизайнером и разработчиком пользовательского интерфейса в компании www.bet365.com. Несмотря на любовь к созданию дизайна веб-приложений, в ее сердце всегда остается особый уголок для печатных изданий. В свободное время она любит выпекать небольшие кексы, экспериментировать в области декоративно-прикладного искусства и склонна к раздаче указаний (тем, кто к ней прислушивается), когда что-нибудь в реальном мире идет не так, как ей нравится.

Сайт Софи находится по адресу www.sophiewill.com, а ее записи в «Твиттере» можно найти на страничке @sophiewill13.

Предисловие

Адаптивный веб-дизайн представляет собой такой дизайн, который имеет подобающий внешний вид на смартфоне, настольном компьютере и на всем, что относится к промежуточным устройствам. Он будет легко реагировать на размер пользовательского экрана, что позволяет пользователям получить наилучшее впечатление от ваших веб-продуктов при работе как на сегодняшних гаджетах, так и на устройствах завтрашнего дня.

Книга охватывает все важные особенности адаптивного веб-дизайна. Методология адаптивного веб-дизайна расширяется в ней за счет применения новейших, наиболее полезных технологий, предоставляемых HTML5 и CSS3, как никогда ранее повышающих компактность и облегчающих сопровождение создаваемых конструкций. В книге также даются объяснения самым востребованным и передовым методам написания и предоставления кода, изображений и файлов.

Если вы в состоянии разобраться в коде HTML и CSS, значит, вам также под силу освоить создание конструкций на основе адаптивного веб-дизайна.

О чем эта книга

Глава 1 «Основы адаптивного веб-дизайна» представляет собой краткий обзор ключевых составляющих в создании программных продуктов, отвечающих требованиям адаптивного веб-дизайна.

Глава 2 «Медиазапросы — поддержка различных окон просмотра» охватывает все, что нужно знать о медиазапросах CSS: их возможности, варианты синтаксиса и различные способы применения.

Глава 3 «Динамически изменяемые разметки и адаптивные изображения» показывает, как создавать код пропорциональных разметок и адаптивных изображений, и содержит детальное исследование разметок, создаваемых с помощью Flexbox.

Глава 4 «Использование HTML5 в целях разработки адаптивного веб-дизайна» охватывает все семантические элементы HTML5, семантику текстового уровня и вопросы обеспечения доступности веб-продуктов. В ней также рассматриваются методы вставки видео и аудио в ваши страницы с использованием средств HTML5.

Глава 5 «CSS3. Селекторы, разметка, цветовые режимы и новые возможности» позволяет разобраться с безграничными возможностями CSS: селекторами, установками цветовых решений в форматах HSLA и RGBA, веб-оформлением, относительными единицами измерений окон просмотра и многим другим.

Глава 6 «Создание эстетически привлекательных эффектов средствами CSS3» охватывает CSS-фильтры, блоковые тени, линейные и радиальные градиенты, множественные фоновые изображения и способы нацеливания фоновых изображений на устройства с высоким разрешением экрана.

Глава 7 «Использование SVG для достижения независимости от разрешения» содержит все необходимые объяснения по использованию SVG-графики внутри документов и в качестве фоновых изображений, а также дает описания способов взаимодействия с ними с помощью JavaScript.

Глава 8 «Переходы, преобразования и анимация» дает информацию, как привести все в движение с помощью CSS, и содержит исследование способов организации взаимодействия и анимации элементов с использованием CSS.

Глава 9 «Обуздание форм с помощью HTML5 и CSS3» показывает, что такой нелегкий во все времена вопрос, как работа с формами, существенно упростился с появлением самых последних свойств HTML5 и CSS3.

Глава 10 «Подходы к адаптивному веб-дизайну» содержит исследование основных вопросов, требующих разрешения, прежде чем можно будет приступать к непосредственной разработке адаптивного веб-дизайна, а также целую подборку самых актуальных мудрых мыслей, призванных вдохновить вас на увлекательнейшее занятие — создание адаптивных веб-конструкций.

Что нужно для работы с книгой

- Текстовый редактор.
- Один из самых распространенных браузеров.
- Чувство юмора.

Для кого написана эта книга

Приходилось ли вам создавать два сайта: один для мобильных, а другой — для более крупных устройств? Или, может быть, у вас уже есть первый продукт в стиле адаптивного веб-дизайна, но вы пока не можете собрать в нем все в единое целое? Если так, то второе издание этой книги даст вам все, что нужно для перевода ваших сайтов на следующий, более высокий уровень.

Чтобы следовать за авторской мыслью, нужны лишь некоторые познания в HTML и CSS, а все, что требуется знать об адаптивном веб-дизайне и создании качественных сайтов, включено в эту книгу!

Соглашения

В издании будут встречаться текстовые стили, позволяющие выделять различные виды информации. Рассмотрим примеры этих стилей и их предназначение.

Встречающиеся в тексте слова, относящиеся к программному коду и примерам пользовательского ввода, будут выделены `моноширинным шрифтом`.

Имена папок, файлов, расширения имен файлов, названия путей, URL-адреса и элементы интерфейса будут выделены узким шрифтом без засечек.

Блоки кода будут показаны в следующем виде:

```
img {
    max-width: 100%;
}
```

Новые понятия и важные слова будут выделены **полужирным шрифтом**.

ПРИМЕЧАНИЕ

В такой врезке будет содержаться важная сопутствующая информация.

СОВЕТ

А в такой врезке будут представлены советы и интересные приемы работы.

1 Основы адаптивного веб-дизайна

Всего лишь несколько лет назад сайты могли создаваться с фиксированной шириной в расчете на то, что все конечные пользователи получат удобные условия работы. Эта фиксированная ширина (обычно 960 пикселов или около того) была не слишком велика для экранов ноутбуков, а пользователи, имеющие мониторы с высоким разрешением, просто видели с обеих сторон большие поля.

Но в 2007 году на телефонах iPhone компании Apple впервые появились по-настоящему удобные условия просмотра информации и условия доступа людей к веб-данным и работы с ними изменились навсегда.

В первом издании этой книги было отмечено, что «за 12 месяцев, с июля 2010-го по июль 2011 года, процент использования мобильных браузеров во всем мире вырос с 2,86 до 7,02».

В середине 2015 года в той же статистической системе (gs.statcounter.com) утверждалось, что данный показатель увеличился до 33,47 %. Для сравнения, в компании North America Mobile этот показатель достиг 25,86 %.

Использование мобильных устройств растет по любым показателям, а на другом краю шкалы в общую практику применения входят 27- и 30-дюймовые дисплеи. Как никогда ранее растет и разрыв между наименьшими и наибольшими экранами, на которых просматривается веб-содержимое. Благо есть решение, подходящее для постоянно расширяющейся арены браузеров и устройств. Нормальная работа сайта на множестве устройств и экранов возможна благодаря так называемому адаптивному дизайну, построенному на основе применения HTML5 и CSS3. Эта технология позволяет разметке и возможностям сайта подстраиваться под имеющееся окружение (размер экрана, тип ввода и возможности устройства или браузера).

Более того, адаптивный веб-дизайн, создаваемый с помощью HTML5 и CSS3, может быть реализован без необходимости использования конечных решений на серверной стороне.

Итак, вперед к неизведанному

Я надеюсь, что глава 1 поможет достичь одной из двух целей независимо от того, в новинку ли вам такие понятия, как адаптивный дизайн, HTML5 или CSS3, или вы в них неплохо разбираетесь.

Если HTML5 и CSS3 уже используются вами для адаптивной веб-разработки, эта глава позволит быстро вспомнить их основы. А если вы новичок в этом деле,

воспринимайте ее в качестве своеобразного курса молодого бойца, охватывающего все с самого начала, так что данная глава пригодится всем без исключения.

К концу главы будет рассмотрено все необходимое для создания полностью адаптивной веб-страницы.

А зачем тогда остальные девять глав? Это также станет понятно к концу главы.

В текущей главе будут рассмотрены следующие вопросы:

- определение адаптивного веб-дизайна;
- способы настройки уровней поддержки браузеров;
- краткий обзор инструментальных средств и текстовых редакторов;
- создание первого адаптивного примера — простой страницы, созданной с использованием HTML5;
- важность метатега `viewport`;
- способ масштабирования изображений под их контейнер;
- написание медиазапросов CSS3 для создания контрольных точек дизайна;
- несовершенство нашего простого примера;
- почему мы находимся всего лишь в самом начале пути.

Определение адаптивного веб-дизайна

Понятие «адаптивный веб-дизайн» было введено Итаном Маркоттом (Ethan Marcotte) в 2010 году. В своей новаторской статье на сайте A List Apart (http://www.alistapart.com/articles/responsive-web-design/) он свел воедино три уже существовавшие на тот момент технологии (гибкий макет на основе сетки, подстраиваемые по размеру изображения и элементы мультимедиа, а также медиазапросы) в единый унифицированный подход, который он назвал адаптивным веб-дизайном.

Адаптивный веб-дизайн в кратком изложении. Адаптивный веб-дизайн является представлением веб-содержимого в наиболее удобном формате для окна просмотра и устройства, обращающегося за этим содержимым.

На ранней стадии развития наиболее характерным для адаптивного дизайна было построение, начинающееся с «рабочего стола», то есть дизайна с фиксированной шириной. Затем, чтобы этот дизайн работал на экранах меньшего размера, содержимое автоматически переформатировалось или удалялось. Но процесс не стоял на месте, и стало понятно, что все, от дизайна до содержимого и разработки, получается намного лучше, если действовать в обратном направлении, начиная с небольших экранов и работая по нарастающей.

Прежде чем вникать во все это, я хочу рассмотреть два вопроса, касающиеся поддержки браузера, а также текстовых редакторов и инструментальных средств.

Установка уровней поддержки браузеров

Благодаря популярности и возможности повсеместного использования адаптивного веб-дизайна теперь стало намного легче торговать с клиентами и ключевыми

партнерами. Некоторые представления об адаптивном веб-дизайне сложились уже у большинства людей. Понятие единого кода, способного работать практически на всех устройствах, становится весьма привлекательным.

При запуске проекта с адаптивным дизайном почти всегда возникает вопрос поддержки со стороны браузеров. При наличии столь широкого спектра браузеров и устройств вряд ли имеет смысл реализовывать полную поддержку изменений каждого отдельно взятого браузера. Вероятно, сдерживающим фактором является время, возможно — деньги. А может быть, и то и другое.

Как правило, чем старее браузер, тем больший объем работы и кода требуется для достижения желаемого результата или выравнивания эстетического восприятия с тем, которое пользователь получает при работе с современными браузерами. Поэтому рациональнее иметь менее вариативный и благодаря этому более быстродействующий код за счет распределения создаваемых впечатлений по уровням, обеспечивая получение самых совершенных визуальных эффектов и возможностей только на наиболее восприимчивых к ним браузерах.

В предыдущем издании этой книги некоторое время уделялось рассмотрению вопросов обслуживания очень старых браузеров, работающих исключительно на настольных компьютерах. В новом издании тратить время на это мы не будем.

В середине 2015 года мне уже приходилось писать о том, что времена браузеров Internet Explorer 6, 7 и 8 прошли. Даже IE 9 на всемирном рынке браузеров занимает весьма скромные 2,45 % (у IE 10 только 1,94 %, а у IE 11 уже более привлекательный показатель, равный 11,68 %). Если вам не остается ничего другого, как вести разработку под Internet Explorer 8 и более ранние версии, то я вам искренне сочувствую и заявляю, что ничего особо полезного из этой книги вы для себя не почерпнете.

Все остальные должны объяснить своему клиенту или заказчику, финансирующему проект, ошибочность разработки под скудные по своим возможностям браузеры и причины, по которым выделение времени и ресурсов преимущественно на разработку под современные браузеры и платформы во всех отношениях имеет более определенный финансовый смысл.

Но в конечном счете реальный вес имеет только ваша собственная статистика. За исключением экстремальных ситуаций, создаваемые нами сайты должны работать как минимум на каждом из самых распространенных браузеров. Кроме основных функциональных возможностей, для каждого веб-проекта имеет смысл заранее определить, на каких платформах нужно в полной мере создать наилучшие впечатления от его работы, а на каких вполне возможно будет согласиться на визуальные или функциональные отступления.

Вы также поймете, что на практике легче начинать с представления, создаваемого самым простым, базовым уровнем, и заниматься его усложнением (этот подход называется **постепенным усложнением**), чем подходить к решению проблемы с противоположной стороны — заниматься сначала созданием представления самого высокого уровня, а затем предпринимать попытки отступления для работы на платформах с более скромными возможностями (этот подход называется **постепенным упрощением**).

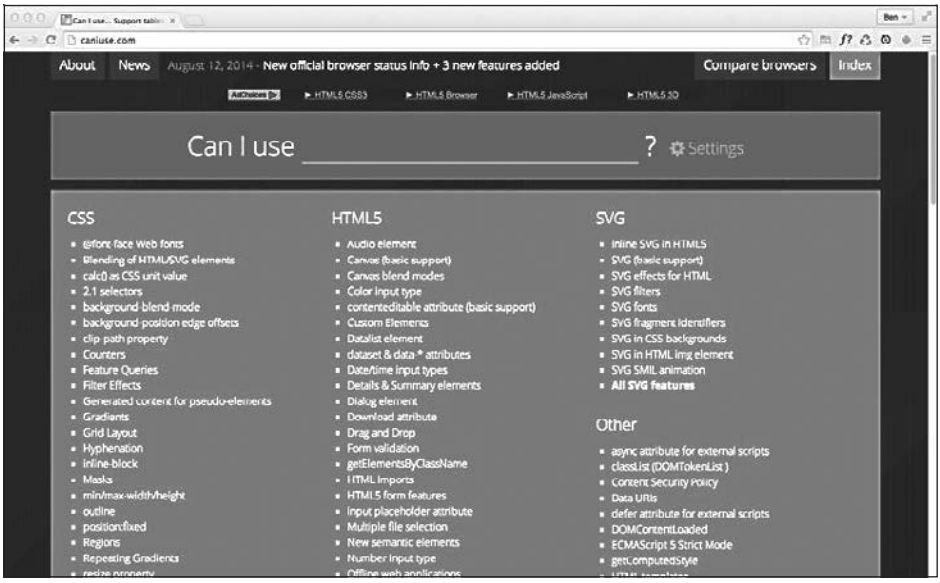
Чтобы пояснить причину, по которой это важно знать заранее, подумайте о том, что, если вам не повезет и 25 % посетителей вашего сайта будут пользоваться, к примеру, Internet Explorer 9, придется учесть, какие функции поддерживаются

этим браузером, и подстраивать свое решение под эти качества. Те же меры станут необходимы, если большой контингент ваших пользователей посещает сайт с устаревших платформ мобильных телефонов, таких как Android 2. Что именно рассматривать в качестве базового представления, будет сильно варьироваться в зависимости от проекта.

Если подходящие данные недоступны, то для определения того, нужно ли тратить время на разработку версии под конкретную платформу или браузер, я прибегаю к простому логическому приему: если стоимость разработки под браузер X и дальнейшей поддержки превышает выручку или выгоду, создаваемую пользователями браузера X, то вести разработку конкретных решений под браузер X не стоит.

Надо реже задаваться вопросом о том, возможно ли вести подгонку под устаревшую платформу или браузер, и чаще спрашивать себя о целесообразности этого.

Если при рассмотрении вопроса о том, какие функции какими платформами и версиями браузеров поддерживаются, вы еще не ознакомились с сайтом http://caniuse.com, я настоятельно советую вам сделать это. Там предоставляется весьма простой интерфейс для выявления, поддержкой какого браузера пользуются интересующие нас функции.



Краткая справка по инструментарию и текстовым редакторам. Неважно, какими именно текстовым редактором или IDE-системой вы пользуетесь для создания своих адаптивных веб-конструкций. Если самые простые текстовые редакторы позволяют вам успешно записывать код HTML, CSS и JavaScript, значит, все в порядке. Аналогично этому нет никакой особой оснастки, играющей важную роль в получении на выходе адаптивного веб-дизайна. Фактически вам

нужно лишь что-то позволяющее записывать код HTML, CSS и JavaScript. Чему вы отдадите предпочтение, Sublime Text, Vim, Coda, Visual Studio или Блокноту, не играет практически никакой роли. Работайте в той среде, которая вас больше всего устраивает.

Тем не менее следует знать, что, в отличие от прежних времен, сейчас имеется множество инструментальных средств (зачастую бесплатных), способных существенно облегчить выполнение рутинных, затратных по времени задач создания сайтов. Например, CSS-процессоры (Sass, LESS, Stylus, PostCSS) способны помочь с организацией кода, с переменными, в работе с цветовыми решениями и арифметикой. Такие средства, как PostCSS, способны также автоматизировать решение трудоемких и неприятных задач, например установку в коде CSS префиксов производителей. Кроме того, проверочные средства могут сравнить ваш код HTML, JavaScript и CSS со стандартами, по которым вы работаете, сэкономив массу времени на выявление опечаток и синтаксических ошибок.

Постоянно появляются все новые и новые инструментальные средства с более совершенными свойствами. Поэтому, какие бы актуальные и полезные средства ни упоминались в настоящее время, следует иметь в виду, что где-то рядом на выходе уже есть что-то более интересное. Следовательно, в своих примерах мы не можем полагаться на что-либо иное, кроме стандартов на основе HTML и CSS. Но вы можете воспользоваться доступными средствами для создания своей интерфейсной части кода как можно быстрее и надежнее.

Первый пример придания адаптивности

В начале главы я обещал, что к ее завершению вы узнаете обо всем необходимом для создания полностью адаптивной веб-страницы. До сих пор вокруг этого вопроса велись лишь досужие разговоры. Настало время выйти на прогулку.

ПРИМЕРЫ ПРОГРАММНОГО КОДА

Все примеры кода из этой книги можно загрузить по адресу rwd.education/download.zip или через GitHub на сайте https://github.com/benfrain/rwd. Имейте в виду, что окончательные версии отдельных примеров, приводимых в главе, имеются только в загружаемом коде. Так, если загрузить примеры кода из главы 2, они будут иметь тот вид, в котором приводятся в конце данной главы. В отличие от текста самой главы, в загружаемых примерах кода никаких промежуточных состояний не приводится.

Наш исходный файл HTML

Начнем с простой структуры HTML5. Не обращайте внимания на предназначение абсолютно каждой строки (особенно на содержимое контейнера `<head>`, которое более подробно будет рассматриваться в главе 4).

Предлагаю пока сконцентрироваться на элементах внутри тега `<body>`. Я абсолютно уверен, что вы не увидите там ничего необычного: всего лишь несколько `div`-контейнеров, символ логотипа, изображение (симпатичная булочка), один-два текстовых абзаца и список ингредиентов.

Вам будет представлена сокращенная версия кода. Для краткости я удалил из показанного далее кода текстовые абзацы, поскольку нас интересует лишь структура. Но вы должны знать, что это рецепт и описание способа изготовления булочек, относящихся к типично британской выпечке.

Если есть желание просмотреть весь файл HTML, его можно загрузить с сайта rwd.education:

```
<!doctype html>
<html class="no-js" lang="en">
  <head>
    <meta charset="utf-8">
    <title>Our first responsive web page with HTML5 and CSS3</title>
    <meta name="description" content="A basic responsive web page
      - an example from Chapter 1">
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <div class="Header">
     <a href="/" class="LogoWrapper"><img src="img/SOC-Logo.png" alt="Scone O'Clock
logo" /></a>
      <p class="Strap">Scones: the most resplendent of snacks</p>
    </div>
    <div class="IntroWrapper">
      <p class="IntroText">Occasionally maligned and
         misunderstood; the scone is a quintessentially British classic.</p>
      <div class="MoneyShot">
        <img class="MoneyShotImg" src="img/scones.jpg" alt="Incredible scones" />
        <p class="ImageCaption">Incredible scones, picture from Wikipedia</p>
      </div>
    </div>
    <p>Recipe and serving suggestions follow.</p>
    <div class="Ingredients">
      <h3 class="SubHeader">Ingredients</h3>
      <ul>

      </ul>
    </div>
    <div class="HowToMake">
      <h3 class="SubHeader">Method</h3>
      <ol class="MethodWrapper">

      </ol>
    </div>
  </body>
</html>
```

Изначально веб-страницы обладают гибкостью. Если открыть страницу примера даже в теперешнем ее состоянии (без медиазапросов) и изменить размер окна браузера, станет видно, что текст будет подвергнут необходимой перекомпоновке.

А как она себя поведет на других устройствах? При полном отсутствии CSS на iPhone получится следующее изображение.

Как видите, на iPhone она отобразилась точно так же, как и обычная веб-страница. Дело в том, что изначально iOS отображает веб-страницы шириной 980 пикселов и ужимает их в области просмотра.

Область просмотра браузера в технической терминологии известна как окно просмотра (viewport). Иногда это окно соответствует размеру экрана устройства, особенно в тех случаях, когда у пользователя есть возможность изменить размеры окна браузера. Поэтому впредь при обозначении пространства, доступного для нашей веб-страницы, как правило, будет использоваться эта, более точная терминология.

Эта заранее предполагаемая проблема легко решается путем добавления в `<head>`-контейнер следующего фрагмента кода:

```
<meta name="viewport" content="width=device-width">
```

Фактически этот метатег с именем `viewport` не считается стандартным способом указания браузеру способа отображения страницы (хотя и является стандартом

де-факто). В данном случае наш метатег `viewport` представляет собой четкое предписание «отобразить содержимое во всю ширину экрана устройства». Легче, наверное, просто показать вам действие этой строки кода на воспринимающих ее устройствах.

Отлично! Теперь тест отобразился и разлился до более естественного размера. Пойдем дальше.

Различные настройки и сочетания метатега (и стандарты, послужившие основой для версии подобных функциональных возможностей) будут рассмотрены в главе 2.

Укрощение изображений

Как говорится, лучше один раз увидеть, чем сто раз услышать. Это относится и к булочкам на нашей взятой для примера странице, на которой изображение всей этой красоты пока что отсутствует. Я собираюсь поместить изображение булочки ближе к началу страницы в качестве своеобразной приманки для пользователей, чтобы им захотелось прочитать ее содержимое.

Увы! Весьма привлекательное, но довольно крупное изображение (шириной 2000 пикселов (px)) заставило страницу показать лишь часть картинки. Нужно исправить положение. Можно, конечно, задать с помощью CSS фиксированную ширину изображения, но перед нами стоит несколько иная задача: мы хотим, чтобы изображение масштабировалось под различные размеры экрана.

Например, на взятом нами для примера устройстве iPhone ширина составляет 320 пикселов, следовательно, для этого изображения можно установить ширину 320 пикселов, но что произойдет, если пользователь повернет экран? Окно просмотра теперь будет шириной не 320, а 480 пикселов. Нам повезло, поскольку получить подстраиваемые изображения, изменяющие масштаб под доступную ширину своего контейнера, можно с помощью всего лишь одной строчки кода CSS.

Теперь я собираюсь создать файл css/styles.css, ссылку на который дам в заголовке HTML-страницы.

Следующий код попадет в него в первую очередь. Обычно я устанавливаю и некоторые другие исходные настройки, речь о который пойдет в следующих главах, но для выполнения нашей задачи остановлюсь на том, что в начало файла помещу именно этот код:

```
img {
    max-width: 100%;
}
```

Теперь после обновления страницы мы увидим что-то больше соответствующее нашим ожиданиям.

Правило, на котором основано свойство `max-width`, предполагает, что максимальная ширина всех изображений должна составлять 100 % их ширины (то есть они должны расширяться не более чем до 100 % своего размера). Когда содержащий изображения элемент (такой как `body` или `div`, внутри которого они находятся) меньше действительной ширины изображения, масштаб изображений будет просто подстроен, чтобы максимально занять доступное пространство.

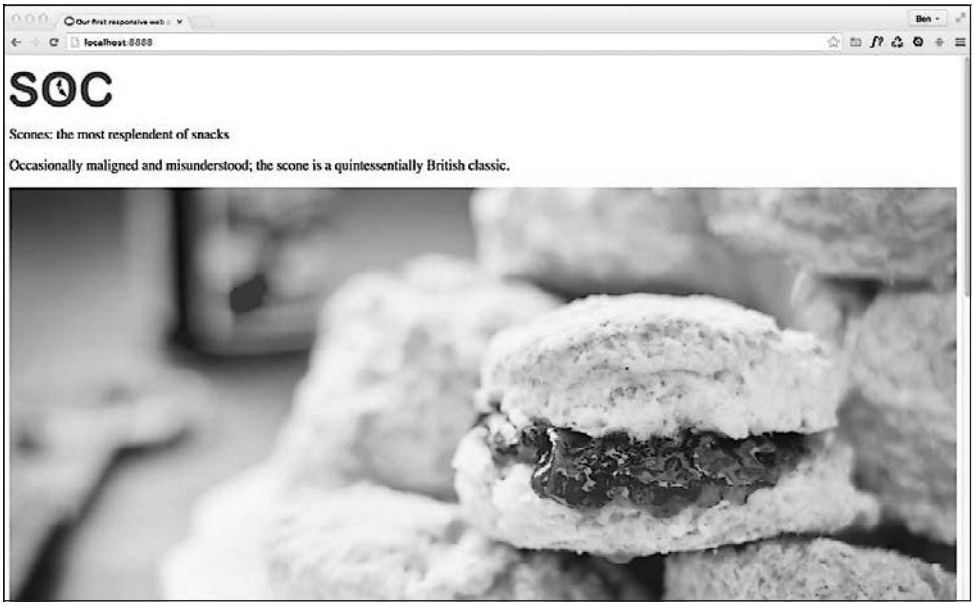
А почему бы просто не воспользоваться свойством `width: 100%`?

Чтобы превратить изображения в подстраиваемые, можно также применить более широко востребованное свойство `width`, например `width: 100%`. Но в результате будет получен совершенно другой эффект. При использовании свойства `width` изображение будет показано с заданной шириной независимо от собственной

ширины. В результате выполнения нашего примера получился бы логотип (также являющийся изображением), растянутый так, чтобы заполнить все 100 % своего контейнера. Когда контейнер намного шире изображения (как в случае с нашим логотипом), получается слишком растянутая картинка.

Замечательно. Теперь все располагается в соответствии с ожиданиями. Независимо от размера окна просмотра ничто теперь не выходит за границы страницы по горизонтали.

Но если посмотреть на страницу в более крупных окнах просмотра, основные стили как в прямом, так и в переносном смысле начинают восприниматься растянутыми. Взгляните на страницу примера при размере окна просмотра, составляющем около 1400 пикселов.



Вот так! Фактически страница начинает выглядеть растянутой уже при ширине примерно 600 пикселов. На данном этапе было бы полезно получить возможность кое-что подправить. Может быть, изменить размер изображения и расположить его рядом с одной из сторон. Может быть, изменить размеры некоторых шрифтов и фоновые цвета элементов.

И здесь нам сопутствует удача, поскольку все эти функциональные возможности мы можем получить без особого труда, привязав с помощью медиазапросов CSS требуемые настройки к нашим желаниям.

Ввод медиазапросов

Как выяснилось, когда окно просмотра выходит по ширине за 600 пикселов, текущая разметка начинает казаться растянутой. Воспользуемся медиазапросами CSS3 для коррекции разметки в зависимости от ширины экрана. Медиазапросы позво-

ляют применять конкретные CSS-правила на основе целого ряда условий (например, ширины и высоты экрана).

НЕ УСТАНАВЛИВАЙТЕ ЗНАЧЕНИЯ КОНРОЛЬНЫХ ТОЧЕК РАВНЫМИ ШИРИНЕ ПОПУЛЯРНЫХ УСТРОЙСТВ

Понятие «контрольная точка» используется для определения точки, в которой адаптивный дизайн должен претерпеть существенные изменения.

Когда люди только начинали применять медиазапросы, зачастую считалось, что контрольные точки в дизайне должны выстраиваться именно вокруг параметров наиболее популярных из имеющихся на то время устройств. Тогда эти контрольные точки обычно выстраивались вокруг параметров iPhone (320 × 480 пикселов) и iPad (768 × 1024 пикселов).

Но их выбор оказался неудачным, а в настоящее время он рассматривается как один из наихудших. Дело в том, что, поступая таким образом, мы ориентируем дизайн на конкретный размер экрана. А нам нужен адаптивный дизайн, то есть не то представление, которое неплохо смотрится только при конкретных размерах экрана, а то, которое не привязано к размеру экрана.

Поэтому позволим определять подходящие места для контрольных точек самому содержимому и дизайну. Может быть, исходная разметка начнет терять подобающий вид при ширине 500 пикселов и более, а может быть, 800 пикселов. Где именно нужно расставить контрольные точки, должно определяться дизайном вашего проекта.

Весь диапазон медиазапросов будет рассмотрен в главе 2, в названии которой и фигурирует этот термин.

Но чтобы наш простой пример не разрастался, мы сконцентрируемся на одном типе медиазапроса, касающегося минимальной ширины. Правила CSS внутри этого типа медиазапроса применяются только в том случае, если окно просмотра имеет минимальную заданную ширину. Точная минимальная ширина может указываться с применением целого набора различных единиц измерения длины, включая проценты, em, rem и px (пиксел). В CSS медиазапрос минимальной ширины записывается следующим образом:

```
@media screen and (min-width: 50em) {
    /* стили */
}
```

Директива `@media` сообщает браузеру о начале медиазапроса, компонент `screen` (применять это объявление экрана в данной ситуации технически не обязательно, но более подробно работать с ним нам придется в следующей главе) сообщает браузеру, что правила должны применяться ко всем типам экранов, и компонент `and (min-width: 50em)` сообщает браузеру, что правила должны действовать для всех окон просмотра, чья ширина превышает 50 em.

СОВЕТ

Я считаю, что первым, кто написал следующую фразу, был Брайан Ригер (Bryan Rieger) (http://www.slideshare.net/bryanrieger/rethinking-the-mobile-web-by-yiibu): «Отсутствие носителя для медиазапросов фактически и является признаком первого медиазапроса».

Он имел в виду, что первыми правилами, которые нам нужно записать вне медиазапроса, должны быть базовые правила, которые мы затем будем наращивать для более соответствующих им устройств.

А пока нужно просто иметь в виду, что этот подход всего лишь сначала интеллектуально подкрепляет наименьший экран, позволяя постепенно детализировать уровни по мере того, как этого потребует дизайн.

Корректировка примера под более крупный экран. Мы уже выяснили, что наш дизайн начинает терять подобающий вид при ширине около 600 пикселов, или 37,5 rem.

Поэтому внесем в простой пример кое-что новенькое, определив разную разметку для разных размеров окна просмотра.

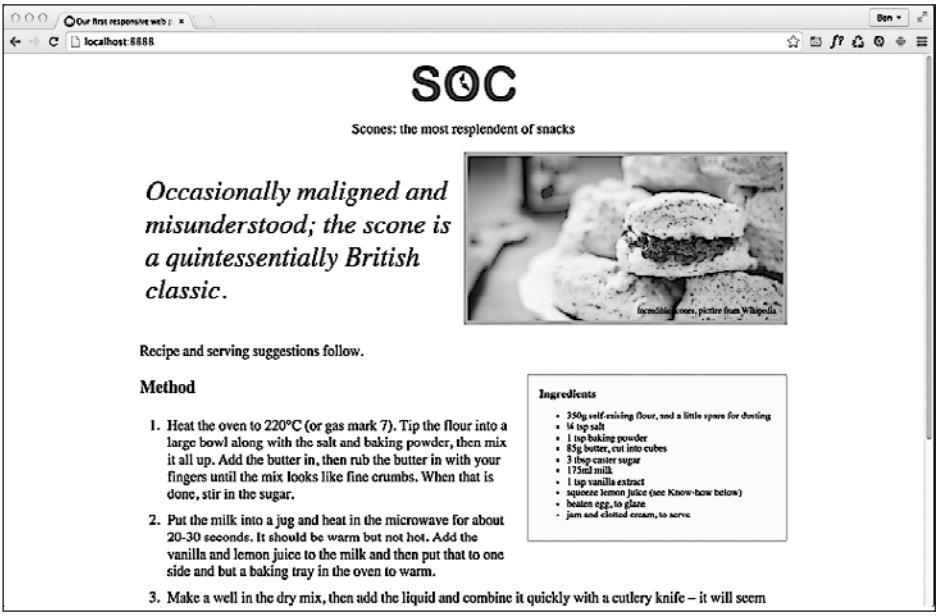
СОВЕТ

Почти у всех браузеров текст имеет исходный размер 16 пикселов, поэтому ширину легко можно преобразовать в rem, разделив значение в пикселах на 16. Зачем нам это понадобится, мы узнаем в главе 2.

Для начала остановим чрезмерное разрастание соответствующего теме сайта изображения, удерживая его в правой части экрана. Затем можно поместить вводный текст в левую часть экрана.

После этого ниже его в левой части экрана можно расположить основную часть текста — рецепт, описывающий способ выпечки булочек, а справа в небольшом разделе, заключенном в прямоугольник, перечислить ингредиенты.

Все эти изменения могут быть получены относительно легко, нужно лишь поместить конкретные стили в медиазапрос. После добавления соответствующих стилей сайт приобретет следующий вид.



На экранах меньшего размера страница будет выглядеть так же, как и раньше, но как только окно просмотра окажется больше или равно 50 rem, страница будет настраиваться под новую разметку.

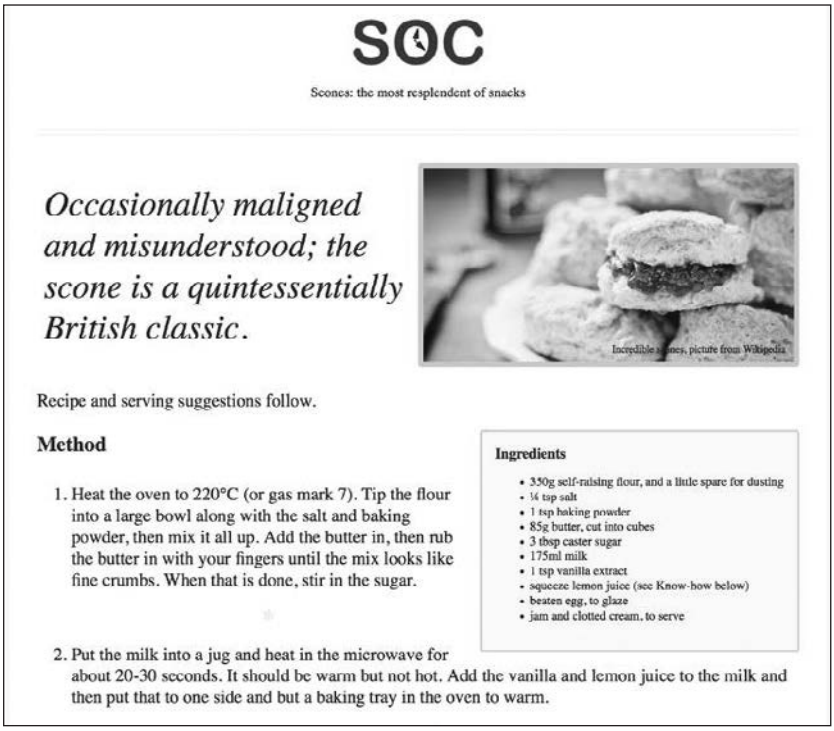
Добавленные стили разметки имеют следующий вид:

```
@media screen and (min-width: 50rem) {
    .IntroWrapper {
        display: table;
        table-layout: fixed;
        width: 100%;
    }

    .MoneyShot,
    .IntroText {
        display: table-cell;
        width: 50%;
        vertical-align: middle;
        text-align: center;
    }

    .IntroText {
        padding: .5rem;
        font-size: 2.5rem;
        text-align: left;
    }

    .Ingredients {
        font-size: .9rem;
        float: right;
        padding: 1rem;
        margin: 0 0 .5rem 1rem;
        border-radius: 3px;
        background-color: #ffffdf;
        border: 2px solid #e8cfa9;
    }

    .Ingredients h3 {
        margin: 0;
    }
}
```

Получилось совсем неплохо, правда? Используя минимум кода, мы создали страницу, реагирующую на размер окна просмотра и предлагающую по мере необходимости желательную разметку. Добавлением всего лишь нескольких дополнительных стилей мы добились более привлекательного внешнего вида страницы.

Теперь, после всех добавлений, наша основная адаптивная страница на iPhone выглядит следующим образом.

А так она выглядит при ширине окна просмотра свыше 50 rem.



Все эти дополнительные визуальные украшения не добавляют понимания того, что происходит в смысле адаптируемости, поэтому здесь я их опустил, но если нужно просмотреть соответствующий код, его можно загрузить по адресу http://rwd.education или https://github.com/benfrain/rwd.

При всей своей простоте этот пример включает в себя неотъемлемые части методологии построения адаптивного веб-дизайна.

Чтобы подтвердить важность всего рассмотренного, начните с базовых стилей, работающих на любом устройстве. Затем по мере роста размеров окна просмотра и/или возможностей постепенно наращивайте уровень пользовательского восприятия.

ПРИМЕЧАНИЕ

Полную спецификацию медиазапросов CSS Media Queries (Level 3) можно найти по адресу http://www.w3.org/TR/css3-mediaqueries/.

А рабочий вариант CSS Media Queries (Level 4) можно найти здесь: http://dev.w3.org/csswg/mediaqueries-4/.

Несовершенство нашего примера

В этой главе были рассмотрены все важные составные части простой адаптивной веб-страницы, использующей свойства HTML5 и CSS3.

Но и вам и мне понятно, что задачи по созданию сайтов вряд ли сведутся к такому простому примеру адаптивного дизайна. И этим примером возможности создания такого дизайна совсем не ограничиваются.

К примеру, что делать, если захочется, чтобы страница реагировала на различные условия освещенности? А как насчет изменения размеров ссылок, когда люди используют различные указывающие устройства (к примеру, палец, а не указатель мыши)? А насчет возможностей простого перемещения визуальных элементов и добавления к ним эффектов анимации с использованием исключительно CSS?

Кроме этого, есть еще вопросы разметки. Как перейти к разметке страниц, имеющих большее количество смысловых элементов, таких как статья, раздел, меню и т. п., или как создать формы со встроенной проверкой, не требующие использования JavaScript? Что если захочется изменить визуальный порядок следования элементов для различных окон просмотра?

И не нужно забывать об изображениях. В этом примере у нас были подстраиваемые изображения, но что, если люди зайдут на страницу с мобильного телефона и им потребуется загрузка большого объема графики (шириной не менее 2000 пикселов), которая на их телефоне будет показана лишь частично? Страница станет загружаться намного медленнее, чем от нее требуется. Ведь есть же более подходящий способ?

А как насчет логотипов и значков? В данном примере использовались изображения формата PNG, но мы ведь запросто можем применить масштабируемую векторную графику (Scalable Vector Graphics (SVG)), чтобы получить качественное изображение, которое не зависит от разрешения экрана просмотра.

Надеюсь, у вас есть время на учебу, поскольку ответы на все эти вопросы будут даны в следующих главах.

Резюме

Мы неплохо поработали, и теперь вам известны и понятны самые важные моменты, необходимые для создания полностью адаптивной веб-страницы. Но как мы только что выяснили, совершенству нет предела.

Все идет своим чередом. Мы хотим не просто получить возможность создания надлежащих веб-проектов с адаптивным веб-дизайном, а создавать лучшие в своем роде представления. Так добьемся же этого.

Для начала мы усвоим все, что могут предложить нам медиазапросы уровней CSS3 и CSS4 (Level 3 и Level 4 CSS Media Queries). Мы уже видели, как веб-страница может подстраиваться под ширину окна просмотра, но уже сейчас способны на более серьезные дела, и вскоре ваши браузеры станут демонстрировать гораздо более интересные возможности. Давайте двигаться дальше и смотреть на все это своими глазами.

2 Медиазапросы — поддержка различных окон просмотра

В предыдущей главе был дан краткий обзор основных компонентов адаптивной веб-страницы: подстраиваемой разметки, подстраиваемых изображений и медиазапросов.

В текущей главе будут подробно рассмотрены медиазапросы. Надеемся, что вам будет предоставлено все необходимое для полного понимания их возможностей, синтаксиса и последующего развития.

В этой главе нам предстоит:

- узнать, зачем медиазапросы нужны в адаптивном веб-дизайне;
- разобраться с синтаксисом медиазапросов;
- научиться использовать медиазапросы в тегах `link` с CSS-инструкциями `@import` и внутри самих файлов CSS;
- разобраться с тем, какие именно свойства устройств поддаются тестированию;
- воспользоваться медиазапросами для обеспечения визуальных изменений в зависимости от доступного пространства экрана;
- решить вопрос о том, нужно ли группировать медиазапросы или записывать их по мере необходимости там, где это потребуется;
- разобраться в том, что представляет собой метатег `viewport`, позволяющий медиазапросам работать надлежащим образом на устройствах под управлением iOS и Android;
- рассмотреть возможности, которые будут предложены в будущих спецификациях медиазапросов.

Спецификация CSS3 состоит из нескольких модулей. Одним из таких модулей является Media Queries (Level 3). Медиазапросы позволяют нам определить целевое назначение конкретных CSS-стилей в зависимости от возможностей устройства. Например, с помощью всего лишь нескольких строк кода CSS мы можем изменить способ отображения содержимого в зависимости от таких параметров, как ширина окна просмотра, соотношение сторон экрана, ориентация экрана (альбомная или портретная) и т. д.

Медиазапросы получили довольно широкую реализацию. Их поддерживают практически все браузеры, кроме самых ранних версий Internet Explorer (8 и ниже).

Короче говоря, не существует абсолютно никаких причин для того, чтобы не пользоваться ими!

СОВЕТ

Спецификации в W3C проходят через процесс ратификации. Если выдастся свободный денек, не поленитесь ознакомиться с официальным объяснением этого процесса по адресу http://www.w3.org/2005/10/Process-20051014/tr. Простейшая версия состоит в том, что спецификация начинается с рабочего проекта (Working Draft (WD)), проходит стадии кандидата в рекомендации (Candidate Recommendation (CR)), предложения в рекомендации (Proposed Recommendation (PR)) и, наконец, спустя много лет добирается до рекомендации (W3C Recommendation (REC)). Как правило, безопаснее использовать модули, находящиеся на более высоком уровне становления. Например, CSS Transforms Module Level 3 (http://www.w3.org/TR/css3-3d-transforms/) пребывал в статусе WD с марта 2009 года, а его поддержка со стороны браузеров была гораздо скуднее, чем поддержка модулей, пребывающих в статусе CR, например, медиазапросов.

Зачем в адаптивном веб-дизайне нужны медиазапросы

Медиазапросы CSS3 позволяют нацеливать конкретные CSS-стили на определенные возможности устройств или возникающие ситуации. Если углубиться в W3C-спецификацию, относящуюся к модулю медиазапросов CSS3 (http://www.w3.org/TR/css3-mediaqueries/), можно увидеть их следующее официальное представление:

> «Медиазапрос состоит из типа среды и выражений в количестве от нуля и более, которые ведут проверку условий конкретных медиасвойств. К медиасвойствам, используемым в медиазапросах, относятся ширина —'width', высота — 'height' и цвет — 'color'. За счет использования медиазапросов представления без изменения своего содержимого могут быть привязаны к конкретному диапазону устройств вывода информации».

Без медиазапросов мы не сможем внести существенные изменения в визуальное представление сайта с помощью одних только CSS-таблиц. Они облегчают нам написание стабильных CSS-правил, упреждающих такие случаи, как портретная ориентация экрана, небольшие или большие размеры окна просмотра и многое другое.

Хотя подстраиваемая разметка может справляться с выдерживанием определенного дизайна в довольно широком диапазоне изменений с учетом всего диапазона размеров экрана, который мы надеемся охватить, бывают случаи, когда требуется более глубокий пересмотр разметки. Возможности такого пересмотра дают медиазапросы. Их нужно рассматривать в качестве основной условной логики для CSS.

Основная условная логика в CSS. Во всех настоящих языках программирования имеются средства, благодаря применению которых обслуживается одна или несколько возможных ситуаций. Обычно такие средства существуют в виде условной логики, примером которой может послужить инструкция `if - else`.

Если выражения, характерные для программирования, непривычны вашему глазу, не волнуйтесь: данная концепция довольно проста. Наверняка вы исполь-

зуете условную логику, когда в кафе просите друга заказать себе что-нибудь: «Если у них есть тройные шоколадные кексы, то я бы взял один, а если нет, то я бы не отказался от кусочка морковного пирога». Это простая условная инструкция с двумя возможными результатами, которых в данном случае вполне достаточно.

Когда я работал над этой книгой, в CSS еще не было возможности использования настоящей условной логики или свойств, присущих программированию. Циклы, функции, итерации и сложные математические вычисления все еще не вышли за пределы прерогативы CSS-процессоров (не помню, упоминал ли я о прекрасной книге на тему препроцессора Sass, которая называется Sass and Compass for Designers?). И тем не менее медиазапросы являются одним из механизмов CSS, позволяющих нам создавать основную условную логику. В том случае, когда при использовании медиазапроса складываются конкретно оговоренные в нем условия, в область видимости попадают именно те стили, которые в нем объявляются.

ПУТИ СОЗДАНИЯ ВОЗМОЖНОСТЕЙ ПРОГРАММИРОВАНИЯ

Популярность препроцессоров CSS заставила призадуматься тех людей, которые работают над спецификациями CSS. Сейчас уже есть WD-спецификация, касающаяся CSS-переменных: http://www.w3.org/TR/css-variables/.

Но поддержка со стороны браузеров ограничивается Firefox, поэтому в настоящее время об их повсеместном использовании не может быть и речи.

Синтаксис медиазапроса

Так как же выглядят медиазапросы и, что более важно, как они работают?

Введите в конце любого файла CSS следующий код и посмотрите на связанную с ним веб-страницу. Как вариант, можете открыть пример из файла каталога example_02-01:

```
body {
    background-color: grey;
}
@media screen and (min-width: 320px) {
    body {
        background-color: green;
    }
}
@media screen and (min-width: 550px) {
    body {
        background-color: yellow;
    }
}
@media screen and (min-width: 768px) {
    body {
        background-color: orange;
    }
}
```

```
@media screen and (min-width: 960px) {
    body {
        background-color: red;
    }
}
```

Теперь посмотрите на файл в браузере и измените размер окна. Фоновый цвет страницы будет меняться в зависимости от текущего размера окна просмотра. Вскоре работа кода с этим синтаксисом будет рассмотрена более подробно. Но в первую очередь важно узнать, как и где могут использоваться медиазапросы.

Медиазапросы в тегах link. Все, кому приходилось работать с CSS, начиная со второй версии (Version 2), знают о возможности указывать тип устройства (например, `screen` или `print`) применительно к таблице стилей с атрибутом `media` тега `<link>`. Рассмотрим следующий пример, который следует помещать в теги `<head>` своей разметки:

```
<link rel="style sheet" type="text/css" media="screen" href="screen-
styles.css">
```

Медиазапросы добавляют к возможности простого указания типа устройства возможность указания целевого назначения стилей на основе возможностей или характерных особенностей устройства. Их нужно рассматривать в качестве вопросов к браузеру. Если браузер даст положительный ответ, применяются стили, заключенные в медиазапрос. Если ответ будет отрицательным, эти стили применяться не будут. Вместо того чтобы просто спросить: «Это экран?» — все, что по максимуму можно сделать, используя только CSS2, — медиазапросы задают более сложные вопросы. С их помощью можно спросить: «Экран ли это и просматривается ли он в портретной ориентации?» Рассмотрим соответствующий пример:

```
<link rel="stylesheet" media="screen and (orientation: portrait)"
href="portrait-screen.css" />
```

Сначала с помощью выражения медиазапроса задается вопрос о типе (экран ли это?), а затем о свойстве (находится ли экран в портретной ориентации?). Таблица стилей `portrait-screen.css` будет применяться для любого экранного устройства с портретной ориентацией и игнорироваться для любых других вариантов устройств и свойств. Логику любого выражения медиазапроса можно поменять на противоположную, добавляя вначале отрицание `not`. Например, результат нашего предыдущего примера будет прямо противоположным, и файл с таблицей стилей будет применяться для всего, что не является экраном с портретной ориентацией:

```
<link rel="stylesheet" media="not screen and (orientation: portrait)"
href="portrait-screen.css" />
```

Объединение медиазапросов

В одной строке можно также выстроить сразу несколько выражений. Давайте, к примеру, расширим один из предыдущих фрагментов кода и вдобавок ко всему прочему ограничим применение файла устройствами, ширина окна просмотра которых составляет не менее 800 пикселов:

```
<link rel="stylesheet" media="screen and (orientation: portrait) and
(min-width: 800px)" href="800wide-portrait-screen.css" />
```

К тому же у нас может быть список медиазапросов. Файл будет применен, если будет получен положительный ответ на любой из перечисленных запросов. Если положительных ответов не окажется, файл применен не будет. Рассмотрим следующий пример:

```
<link rel="stylesheet" media="screen and (orientation: portrait) and
(min-width: 800px), projection" href="800wide-portrait-screen.css" />
```

В нем есть две интересные особенности. Во-первых, для разделения медиазапросов используются запятые. Во-вторых, следует заметить, что в скобках после ключевого слова `projection` отсутствует следующая пара «свойство — значение» и/или сочетание свойств и значений. Дело в том, что при отсутствии таких значений медиазапрос применяется ко всем устройствам, относящимся к данному типу медиаустройств. В нашем примере стили будут применяться ко всем проекторам.

СОВЕТ

Следует иметь в виду, что для указания медиазапросов можно использовать любую единицу измерения длины, применяемую в CSS. Наиболее часто используются пикселы (px), но подойдут также em и rem. Чтобы дать более подробную информацию о качествах каждой единицы измерения, я привел расширенное толкование этой темы на ресурсе http://benfrain.com/just-use-pixels.

Таким образом, если нужна контрольная точка на 800 пикселах, но указанная в em, нужно просто разделить количество пикселов на 16. Например, 800 пикселов можно также указать как 50 em (800 / 16 = 50).

Медиазапросы с использованием @import

Для условной загрузки таблиц стилей в существующую таблицу стилей можно использовать также CSS-конструкцию `@import`. Например, следующий код импортирует таблицу стилей под названием `phone.css` при условии, что устройством является экран с максимальной шириной окна просмотра 360 пикселов:

```
@import url("phone.css") screen and (max-width:360px);
```

Следует помнить, что при использовании CSS-конструкции `@import` код добавляется к HTTP-запросам, что влияет на скорость загрузки, поэтому данный метод нужно использовать осмотрительно.

Медиазапросы в CSS

До сих пор медиазапросы включались нами в виде ссылок на CSS-файлы, а сами ссылки помещались в раздел `<head></head>` кода HTML и в виде инструкций `@import`. Но, скорее всего, нам захочется использовать медиазапросы внутри самих таблиц стилей. Например, если в таблицу стилей добавить следующий код, все элементы `h1` приобретут зеленый цвет при условии, что ширина экрана устройства — не более 400 пикселов:

```
@media screen and (max-device-width: 400px) {
    h1 { color: green }
}
```

Сначала в виде правила `@media` указывается, что нам нужен медиазапрос, затем задается желаемый тип, с которым требуется совпадение. В предыдущем фрагменте кода требуется применить правила, относящиеся только к экрану, но, к примеру, не к принтеру. Затем внутри скобок вводятся особенности запроса. После этого, как и в любом CSS-правиле, открываются фигурные скобки, внутри которых записываются требуемые стили.

Здесь, видимо, следует заметить, что в большинстве ситуаций указывать экран (`screen`) не нужно. Основное положение спецификации звучит следующим образом:

> «Сокращенный синтаксис предлагается для тех медиазапросов, которые применяются ко всем типам медиаустройств; ключевое слово all может быть опущено (наряду со следующим за ним and). То есть если тип медиаустройства не задан конкретным образом, подразумевается настройка all».

Следовательно, пока не возникнет потребность нацелить стили на конкретные типы медиаустройств, `screen` и `and` можно опустить. Впредь все медиазапросы в примерах будут записываться с учетом этой возможности.

Что можно тестировать с помощью медиазапросов

При построении адаптивного дизайна чаще всего используются медиазапросы, относящиеся к ширине окна просмотра (`width`). Исходя из личного опыта, могу сказать: другие возможности мне особо не пригодились (за редким исключением использования разрешения и высоты окна просмотра). Но на случай возникновения таких потребностей представляю вам перечень всех возможностей Media Queries Level 3, которыми можно будет воспользоваться. Может быть, некоторые из них вас смогут заинтересовать:

- `width:` — ширина окна просмотра;
- `height:` — высота окна просмотра;
- `device-width:` — ширина поверхности отображения (для нас это, как правило, ширина экрана устройства);
- `device-height:` — высота поверхности отображения (для нас это, как правило, высота экрана устройства);
- `orientation:` — возможность проверки портретной или альбомной ориентации устройства;
- `aspect-ratio:` — соотношение ширины к высоте на основе ширины и высоты окна просмотра. Дисплей с соотношением сторон 16:9 может быть описан как `aspect-ratio: 16/9`;
- `device-aspect-ratio:` — эта возможность аналогична предыдущей, но основывается на ширине и высоте не окна просмотра, а поверхности отображения устройства;

- `color:` — количество битов, приходящееся на каждую составляющую цвета. Например, `min-color: 16` задаст проверку того, обладает ли устройство цветом с глубиной 16 бит;
- `color-index:` — количество записей в таблице поиска цветов устройства. Значения должны быть числовыми и не могут быть отрицательными;
- `monochrome:` — возможность проверки количества битов на пиксел в буфере монохромного кадра. Значение должно быть целым числом, например `monochrome: 2`, и не может быть отрицательным;
- `resolution:` — эта возможность может использоваться для проверки разрешения экрана или принтера, например `min-resolution: 300dpi`. Может также приниматься единица измерения в точках на сантиметр, например `min-resolution: 118dpcm`;
- `scan:` — это свойство может отображать значение развертки (прогрессивной или чересстрочной), которое имеет отношение в основном к телевизионным устройствам. Например, нацеливание на устройство с параметрами 720p HD TV (буква «p» в 720p означает progressive — «прогрессивная») может быть обозначено с помощью выражения `scan: progressive`, а на устройство 1080i HD TV (буква «i» в 1080i означает interlaced — «чересстрочная») — с помощью выражения `scan: interlace`;
- `grid:` — эта возможность показывает, на какой основе построено устройство, сеточной или растровой.

Все перечисленные возможности, за исключением `scan` и `grid`, для создания диапазонов могут использоваться с префиксом `min` или `max`. Рассмотрим, к примеру, следующий фрагмент кода:

```
@import url("tiny.css") screen and (min-width:200px) and (max-width:360px);
```

Здесь минимум (`min`) или максимум (`max`) применены для задания диапазона ширины — `width`. Файл `tiny.css` может быть импортирован для экранных устройств с минимальной шириной окна просмотра 200 пикселов и максимальной шириной окна просмотра 360 пикселов.

СВОЙСТВА, НЕ РЕКОМЕНДУЕМЫЕ В МЕДИАЗАПРОСАХ СПЕЦИФИКАЦИИ CSS MEDIA QUERIES LEVEL 4

Следует знать, что в предварительной спецификации медиазапросов Media Queries Level 4 не рекомендуется использование ряда возможностей (http://dev.w3.org/csswg/mediaqueries-4/#mf-deprecated), в частности, device-height, device-width и device-aspect-ratio. Поддержка этих запросов останется в браузерах, но от написания новых таблиц стилей с их использованием рекомендуется воздержаться.

Использование медиазапросов для изменения дизайна

Согласно заложенной в них характерной особенности стили, находящиеся ниже в каскадной таблице стилей (для меня и вас — в CSS-файле), доминируют над такими же стилями, находящимися выше (если только расположенные выше стили не имеют

более конкретной нацеленности). Поэтому в начале таблицы стилей мы можем настроить основные стили, применимые ко всем версиям нашего дизайна (или как минимум обеспечивающие базовый внешний вид), а затем переопределить соответствующие разделы с помощью медиазапросов, следующих далее в документе. Например, можно задаться целью настроить в ограниченных окнах просмотра ссылки переходов в виде простого текста (или, может быть, в виде текста, имеющего более мелкий шрифт), а затем переопределить эти стили с помощью медиазапроса на предоставление в более просторных окнах просмотра не только текста, но и значков.

Посмотрим на то, как это может выглядеть на практике (example_02-02). Сначала разметка:

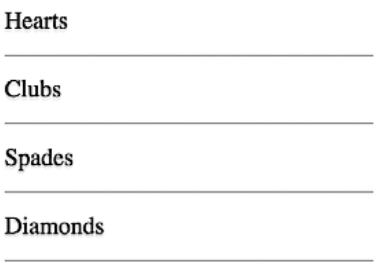
```
<a href="#" class="CardLink CardLink_Hearts">Hearts</a>
<a href="#" class="CardLink CardLink_Clubs">Clubs</a>
<a href="#" class="CardLink CardLink_Spades">Spades</a>
<a href="#" class="CardLink CardLink_Diamonds">Diamonds</a>
```

А теперь CSS:

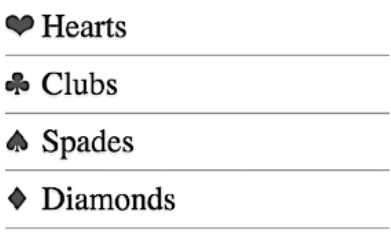
```
.CardLink {
    display: block;
    color: #666;
    text-shadow: 02px 0 #efefef;
    text-decoration: none;
    height: 2.75rem;
    line-height: 2.75rem;
    border-bottom: 1px solid #bbb;
    position: relative;
}
@media (min-width: 300px) {
    .CardLink {
        padding-left: 1.8rem;
        font-size: 1.6rem;
    }
}

.CardLink:before {
    display: none;
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    left: 0;
}

.CardLink_Hearts:before {
    content: "♥";
}

.CardLink_Clubs:before {
    content: "♣";
}
```

```
.CardLink_Spades:before {
    content: "♠";
}

.CardLink_Diamonds:before {
    content: "♦";
}

@media (min-width: 300px) {
    .CardLink:before {
        display: block;
    }
}
```

Вот как выглядит фрагмент экрана при небольшом окне просмотра.



А вот так — при более просторном окне просмотра.



В медиазапрос может быть заключен любой код CSS

Следует запомнить, что все подлежащее описанию в коде CSS также может быть заключено в медиазапрос. По существу, с помощью медиазапросов можно полностью изменить разметку и внешний вид сайта в различных ситуациях (обычно для разных размеров окна просмотра).

Медиазапросы для HiDPI-устройств

Еще одним широко распространенным способом использования медиазапросов является изменение стилей, когда сайт просматривается на устройстве с высоким разрешением. Рассмотрим следующий код:

```
@media (min-resolution: 2dppx) {
    /* стили */
}
```

Здесь медиазапрос определяет, что заключенные в него стили мы желаем применить при разрешении экрана 2 точки на пиксел (2 dppx). Они будут применены к таким устройствам, как iPhone 4 (HiDPI-устройствам компании с условным наименованием Retina), и к целому комплексу Android-устройств. Этот медиазапрос можно изменить с целью применения к более широкому кругу устройств путем уменьшения значения `dppx`.

СОВЕТ

Для более широкой поддержки при написании медиазапросов min-resolution убедитесь в наличии запущенных инструментальных средств для работы с префиксами, чтобы предоставить соответствующие префиксы производителей. Если вы пока не в курсе того, что такое префиксы производителей, волноваться не стоит, поскольку более подробно эта тема будет рассмотрена в следующей главе.

Рассмотрение аспектов организации и разработки медиазапросов

Теперь слегка коснемся различных подходов, доступных разработчикам при написании и организации их медиазапросов. Каждый подход предлагает те или иные преимущества и компромиссы, поэтому будет как минимум полезно знать об этих факторах, даже если вы решили, что они, по большому счету, не имеют отношения к вашим потребностям.

Привязка различных CSS-файлов с помощью медиазапросов

С точки зрения браузера CSS рассматривается как ресурс, приостанавливающий отображение. Браузеру нужно извлечь и проанализировать привязанные CSS-файлы до полного вывода страницы на экран.

Но современные браузеры достаточно разумны для того, чтобы определять, какие таблицы стилей (связанные в `head`-разделе страницы с медиазапросами) должны быть проанализированы немедленно, а какие могут быть отложены и проанализированы уже после вывода на экран начальной страницы.

Для таких браузеров CSS-файлы, привязанные с помощью неприменимых медиазапросов (например, если экран слишком мал для применения медиазапроса),

могут откладываться до момента после загрузки начальной страницы, предоставляя некоторые преимущества в производительности.

Подробности по данной теме можно найти на страницах разработчиков компании Google: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css.

Но я хочу обратить ваше внимание, в частности, на следующий фрагмент:

> «...Учтите, что приостановка отображения относится только к тому, должен ли браузер задержать начальное отображение страницы на этом ресурсе. В любом случае CSS-ресурс все равно загружается браузером, пусть даже с меньшим уровнем приоритета для неблокирующих ресурсов».

Следует пояснить, что все привязанные файлы все равно будут загружены, браузер просто не будет сдерживать отображение страницы на экране, если они не подлежат немедленному применению.

Следовательно, современный браузер загружает адаптивную веб-страницу (посмотрите файл каталога example_02-03) с четырьмя разными таблицами стилей, привязанными с помощью различных медиазапросов (для применения разных стилей для различных диапазонов окон просмотра) с загрузкой всех четырех CSS-файлов, но, возможно, с начальным анализом до вывода страницы на экран только одного из них, применимого в сложившихся условиях.

Практические аспекты разделения медиазапросов

Несмотря на то что мы только что узнали о потенциальных преимуществах разделения медиазапросов, ощутимость преимуществ разделения различных стилей в медиазапросах с помещением их в отдельные файлы не всегда достаточно высока (если не брать в расчет персональные предпочтения и/или обособление кода).

В конце концов, использование отдельных файлов повышает количество HTTP-запросов, необходимых для отображения страницы, что, в свою очередь, может сделать страницу медленнее в некоторых других ситуациях. Ничто в Интернете легко не дается! Поэтому возникает вопрос вычисления общей производительности вашего сайта и тестирования каждого сценария на различных устройствах.

Моя исходная позиция по данному вопросу заключается в том, что если проект располагает достаточным временем для оптимизации производительности, то искать возможности прироста производительности именно в этом месте я стану в последнюю очередь. Сначала я захочу убедиться в том, что:

- сжаты все изображения;
- объединены и минимизированы все сценарии;
- средством gzip обработаны все ресурсы;
- все статическое содержимое кэшировано посредством CDN-сетей;
- удалены все избыточные CSS-правила.

Возможно, после этого я приступлю к решению вопроса о разбиении медиазапросов на отдельные файлы с целью повышения производительности.

СОВЕТ

gzip является файловым форматом сжатия и восстановления. Обработку файлов CSS с существенным уменьшением размера файла при его передаче от сервера к устройству, где он восстанавливается до своего исходного формата, позволяет производить любой высококачественный сервер. Справку по gzip можно найти в «Википедии» по адресу http://en.wikipedia.org/wiki/Gzip.

Вложение медиазапросов путем их встраивания

При любых обстоятельствах, кроме самых крайних, я рекомендую добавлять медиазапросы в существующие таблицы стилей наряду с обычными правилами.

Если вам удается придерживаться этой рекомендации, возникает следующий вопрос: должны ли медиазапросы объявляться ниже связанного с ними селектора? Или же они должны разбиваться в конце на отдельные блоки кода для всех одинаковых медиазапросов? Если у вас возник такой вопрос, я этому весьма рад.

Как поступать — объединять медиазапросы или же записывать их там, где они пригодятся?

Я сторонник записи медиазапросов ниже исходных обычных определений. Если, к примеру, мне нужно изменить ширину двух элементов в разных местах таблицы стилей в зависимости от ширины окна просмотра, я делаю следующее:

```
.thing {
    width: 50%;
}

@media screen and (min-width: 30rem) {
    .thing {
        width: 75%;
    }
}

/* Здесь между рассматриваемыми фрагментами кода помещаются другие стили */

.thing2 {
    width: 65%;
}

@media screen and (min-width: 30rem) {
    .thing2 {
        width: 75%;
    }
}
```

Поначалу все это выглядит весьма странно. У нас есть два медиазапроса, и оба они имеют отношение к экранам с минимальной шириной 30 rem. Неужели повторение `@media`-объявления не считается многословием и расточительством? Наверное, мне следовало бы сгруппировать все идентичные медиазапросы в единый блок:

```
.thing {
    width: 50%;
}

.thing2 {
    width: 65%;
}

@media screen and (min-width: 30rem) {
    .thing {
        width: 75%;
    }
    .thing2 {
        width: 75%;
    }
}
```

Безусловно, это один из способов такой группировки. Но с точки зрения сопровождения кода он представляется мне более сложным. Правильного способа не существует, но я предпочитаю сначала определить одно правило для отдельного селектора и располагать определения любых изменений этого правила (например, изменения внутри медиазапросов) непосредственно после него. При этом мне не придется искать отдельные блоки кода, чтобы найти объявление, относящееся к конкретному селектору.

ПРИМЕЧАНИЕ

Еще более удобным может стать использование препроцессоров и постпроцессоров CSS, поскольку вариант правила, заключенный в медиазапрос, может быть вложен непосредственно в набор правил. В моей книге Sass and Compass for Designers этой теме посвящен целый раздел.

Казалось бы, вполне резонно выступить против вышеупомянутой технологии по причине ее многословия. Неужели один лишь размер файла может стать достаточной причиной для того, чтобы не записывать медиазапросы таким образом? В конце концов, никому не хочется иметь раздутый CSS-файл для обслуживания своих потребностей. Но нужно считаться с тем простым фактом, что gzip-сжатие, которому должны подвергаться все возможные ресурсы на вашем сервере, сокращает разницу до совершенно ничтожных величин. Ранее я провел соответствующие тесты, и если у вас появилось желание получить дополнительные сведения, можете обратиться по адресу http://benfrain.com/inline-or-combined-media-queries-in-sass-fight/. Суть в том, что я не верю, что вас будет тревожить размер файла, если вы предпочтете записывать медиазапросы непосредственно после стандартных стилей.

СОВЕТ

Если вам хочется разрабатывать свои медиазапросы непосредственно после исходных правил, но при этом иметь все идентичные определения медиазапросов объединенными в один запрос, то для этого существует несколько рабочих инструментов (на момент написания данной книги соответствующие дополнительные модули были как у Grunt, так и у Gulp).

Метатег viewport

Для получения максимальной отдачи от медиазапросов хотелось бы, чтобы устройства с меньшими размерами экрана отображали веб-страницы в их исходном размере и не выводили их в окне шириной 980 пикселов, для которого вам пришлось бы масштабировать их в ту или иную сторону.

Когда в 2007 году компания Apple выпустила iPhone, она ввела собственный метатег под названием `viewport`, который теперь поддерживается Android и постоянно растущим количеством других платформ. Предназначением метатега `viewport` является предоставление веб-страницам способа связи с браузерами мобильных устройств для сообщения им предпочтительного способа вывода страницы на экран.

В обозримом будущем те веб-страницы, которые хотелось бы сделать адаптивными и успешно выводимыми на устройствах с малыми размерами экрана, будут вынуждены использовать этот метатег.

ТЕСТИРОВАНИЕ АДАПТИВНОГО ДИЗАЙНА НА ЭМУЛЯТОРАХ И СИМУЛЯТОРАХ

Хотя замены для тестирования результатов разработки на реальных устройствах нет, существует ряд эмуляторов для Android и симуляторов для iOS. Для особо дотошных поясняю, что симулятор просто симулирует нужное устройство, а эмулятор фактически пытается интерпретировать исходный код устройства.

Android-эмулятор для Windows, Linux и Mac находится в свободном доступе для загрузки и установки по адресу http://developer.android.com/sdk/ и применяется в составе программного инструментария разработчика Android Software Development Kit (SDK).

Симулятор для iOS доступен только пользователям Mac OS X и поставляется как часть пакета Xcode, который можно свободно получить в Mac App Store.

У самих браузеров в их средствах разработки также имеются постоянно совершенствуемые инструменты для эмуляции мобильных устройств. Конкретные настройки на эмуляцию различных мобильных устройств и окон просмотра имеются как у Firefox, так и у Chrome.

Метатег `viewport` добавляется в `<head>`-теги кода HTML. Он может настраиваться на определенную ширину (которую, к примеру, можно выразить в пикселах) или содержать указание на масштаб, например `2.0` (удваивание текущего размера). Рассмотрим пример использования метатега `viewport`, настраивающего вывод в браузере в удвоенном по сравнению с исходным размере (200 %):

```
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
```

Разобьем предыдущий метатег на части, чтобы понять, что происходит. Атрибут `name="viewport"` разъяснений не требует. Затем в разделе `content="initial-scale=2.0`

предписывается увеличить размер содержимого вдвое (значение 0.5 уполовинило бы размер, 3.0 — утроило его и т. д.), а раздел width=device-width говорит браузеру о том, что ширина страницы должна быть равна значению device-width.

Метатег может использоваться также для управления пределами возможностей пользователя при масштабировании страницы в обе стороны. Следующий пример позволяет пользователям увеличивать масштаб до трехкратной ширины экрана устройства и уменьшать его до половинной ширины экрана устройства:

```
<meta name="viewport" content="width=device-width, maximum-scale=3,
minimum-scale=0.5" />
```

Можно также вообще не давать пользователям возможности масштабирования, но, поскольку масштабирование является важным средством, повышающим удобство просмотра содержимого, применять эту настройку на практике вряд ли придется:

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

Ключевой здесь является часть user-scalable=no.

Да, мы изменим масштаб на 1.0, что означает вывод страницы браузером мобильного устройства на 100 % окна просмотра. Настройка его на ширину устройства означает, что наша страница должна выводиться на 100 % ширины всех поддерживающих это свойство браузеров мобильных устройств. В большинстве случаев подойдет следующий метатег:

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

СОВЕТ

Заметив рост популярности использования метатега viewport, консорциум W3C предпринял попытки внедрения такой же возможности в CSS. Зайдите на страницу http://dev.w3.org/csswg/css-device-adapt/ и прочитайте все о новом объявлении @viewport. Идея заключается в том, что вместо написания в <head>-разделе вашей разметки метатега вы можете написать в CSS @viewport { width: 320px; }. В результате ширина браузера будет настроена на 320 пикселов. Но поддержка этого объявления со стороны браузеров оставляет желать лучшего, поэтому для охвата всех возможных вариантов и максимального соответствия требованиям завтрашнего дня можно воспользоваться комбинацией метатега и объявления @viewport.

На данный момент у вас уже должно было сложиться четкое представление о медиазапросах и порядке их работы. Но перед тем, как полностью перейти к совершенно другой теме, было бы неплохо рассмотреть возможности ближайшего будущего при использовании следующей версии медиазапросов. Давайте заглянем в будущее!

Спецификация Media Queries Level 4

На момент написания книги, когда CSS Media Queries Level 4 пребывали в стадии проекта спецификации (http://dev.w3.org/csswg/mediaqueries-4/), компоненты этого проекта не могли похвастаться реализацией в множестве браузеров. Это означает, что, несмотря на краткий обзор особенностей этой спецификации, она еще не приобрела достаточной стабильности. Перед использованием любого из ее компонентов

нужно убедиться в его поддержке браузером и дважды проверить, не изменился ли у него синтаксис.

А пока, хотя в спецификации четвертого уровня есть и другие компоненты, мы коснемся только возможностей использования сценариев, реагирования на прохождение указателя мыши над элементами страницы и использования освещенности.

Медиасвойство использования сценариев

Согласно устоявшейся практике тегу `HTML` присваивается класс, показывающий исходное отсутствие кода JavaScript с последующей заменой этого класса на другой при запуске JavaScript. Тем самым предоставляется легко реализуемая возможность разветвления кода, включая CSS, на основе этого нового класса, присваиваемого тегу `HTML`. В частности, используя этот прием, можно написать правила, применяемые к конкретным пользователям, у которых включен JavaScript.

Чтобы не запутаться, рассмотрим пример кода. В самом начале это будет тег, созданный в HTML:

```
<html class="no-js">
```

Когда на странице запустится JavaScript, одной из первых задач станет замена класса `no-js`:

```
<html class="js">
```

После того как это будет сделано, можно написать конкретные CSS-правила, применимые исключительно при наличии JavaScript, например `.js .header { display: block; }`.

А вот медиасвойство использования сценариев, имеющееся в спецификации CSS Media Queries Level 4, предназначено для обеспечения более соответствующего общему стандарту способа достижения аналогичного результата непосредственно в коде CSS:

```
@media (scripting: none) {
    /* стили, предназначенные для применения в отсутствие JavaScript */
}
```

И при наличии JavaScript:

```
@media (scripting: enabled) {
    /* стили, предназначенные для применения при наличии JavaScript */
}
```

И наконец, это свойство предназначено также для выявления факта применения JavaScript только при начальном отображении страницы. В спецификации W3C приводится пример задания отображаемой страницы, которая может быть размечена при начальном выводе на экран, после чего JavaScript на ней использоваться не будет:

```
@media (scripting: initial-only) {
    /* стили для страницы, применяющей JavaScript только при ее начальном выводе */
}
```

О текущем редакторском проекте этого свойства можно прочитать на странице http://dev.w3.org/csswg/mediaqueries-4/#mf-scripting.

Медиасвойства, связанные с взаимодействием со страницей

В W3C медиасвойство указателя `pointer`, представлено следующим образом:

> «Медиасвойство pointer используется для запроса наличия и точности указывающего устройства, например мыши. Если у устройства имеется несколько механизмов ввода, медиасвойство pointer должно отражать характеристики первичного механизма ввода в соответствии с определением пользовательского агента (user agent)».

Существует три возможных состояния указателя `pointer`: `none` (отсутствует), `coarse` (грубый) и `fine` (тонкий).

В качестве устройства указателя с параметром `coarse` может оказаться палец на устройстве с сенсорным экраном. Но это может быть и курсор игровой консоли, не обладающий такой же высокой точностью, как указатель мыши:

```
@media (pointer: coarse) {
    /* стили на случай присутствия указателя, имеющего состояние coarse */
}
```

Устройством со свойством `pointer`, имеющим значение `fine`, может быть мышь, а также стилус-перо или любой другой указательный механизм высокой точности:

```
@media (pointer: fine) {
    /* стили на случай присутствия указателя с высокой точностью */
}
```

Я считаю, что чем скорее в браузерах будет реализовано это свойство указателя, тем лучше. А пока еще определить, что именно используется, мышь, сенсорный ввод или и то, и другое, довольно трудно. Так же трудно определить, чем именно пользуются в данный момент.

СОВЕТ

Разумнее всегда предполагать, что пользователи работают с устройством сенсорного ввода, и задавать размеры элементов пользовательского интерфейса соответствующим образом. Тогда, даже если они используют мышь, с интерфейсом у них не возникнет никаких затруднений. Если же предположить, что устройством ввода является мышь, возникнут осложнения из-за невозможности надежного определения касания при работе с элементами интерфейса.

Более подробный обзор проблем разработки для устройств с сенсорным вводом и устройств, использующих указатель типа «мышь», представлен в наборе слайдов под названием Getting touchy, созданном Патриком Лауке (Patrick H. Lauke) и опубликованном на сайте https://patrickhlauke.github.io/getting-touchy-presentation/.

О редакторском проекте (editor's draft) этого свойства можно прочитать на сайте http://dev.w3.org/csswg/mediaqueries-4/#mf-interaction.

Медиасвойство hover

Как вы, наверное, уже догадались, медиасвойство `hover` тестирует пользовательскую возможность проведения указателя над элементами экрана. Если пользователь располагает несколькими устройствами ввода (например, сенсором и мышью), используются характеристики основного устройства ввода. Далее перечислены возможные значения и показаны примеры кода.

Для пользователей, не имеющих возможности провести указатель над элементами экрана, можно нацелить стили для значения этого свойства, равного `none`:

```
@media (hover: none) {
    /* стили для тех случаев, когда провести указатель над элементами невозможно */
}
```

Для тех пользователей, которые имеют такую возможность, но должны для ее инициирования предпринять определенные действия, можно воспользоваться значением `on-demand`:

```
@media (hover: on-demand) {
    /* стили для тех случаев, когда пользователь
    имеет возможность провести указатель над
    элементами, но от него требуются определенные
    действия */
}
```

Для пользователей, имеющих возможность провести указатель над элементами, может быть использовано объявление `hover` без указания какого-либо значения:

```
@media (hover) {
    /* стили для тех случаев, когда пользователь
    имеет возможность провести указатель над элементами */
}
```

Существуют также медиасвойства `any-pointer` и `any-hover`. Они похожи на рассмотренные ранее `hover` и `pointer`, но проверяют возможности любого из возможных устройств ввода.

Медиасвойства среды

Правда, неплохо было бы иметь возможность изменять наш дизайн на основе свойств окружающей среды, например уровня освещенности? Тогда при переходе пользователя в хуже освещенное помещение можно будет переключиться на использование более яркого цветового оформления. Или же, наоборот, повысить контрастность при переходе в место с ярким солнечным светом. Медиасвойства среды предназначены для решения разнообразных задач подобного рода. Рассмотрим следующие примеры:

```
@media (light-level: normal) {
    /* стили для стандартной освещенности */
}
```

```
@media (light-level: dim) {
    /* стили для приглушенной освещенности */
}
@media (light-level: washed) {
    /* стили для яркой освещенности */
}
```

Следует помнить, что в разрабатываемой спецификации Level 4 Media Queries предусматривается несколько реализаций таких свойств. Возможно также, что до возникновения надежных реализаций эти спецификации претерпят изменения. Но все же будет полезно иметь некоторое представление о том, какие новые возможности ждут нас в ближайшие дни.

О редакторском проекте этих свойств можно узнать на сайте http://dev.w3.org/csswg/mediaqueries-4/#mf-environment.

Резюме

Из данной главы мы узнали, что такое медиазапросы, как они включаются в наши файлы CSS и как могут посодействовать стремлению создавать адаптивный веб-дизайн. Мы также научились использовать метатег, чтобы заставить современные браузеры мобильных устройств выводить на экран страницы в соответствии с нашими предпочтениями.

Но мы также узнали, что сами по себе медиазапросы могут предоставить только такой адаптивный веб-дизайн, в котором осуществляются резкие переходы от одной разметки к другой. Это нельзя считать по-настоящему адаптивным веб-дизайном с плавными переходами от разметки к разметке. Для достижения конечной цели нам нужно будет воспользоваться подстраиваемыми разметками. Они должны позволить нашим конструкциям проявлять гибкость. В следующей главе мы займемся созданием подстраиваемых разметок для придания плавности переходам между контрольными точками наших медиазапросов.

3 Динамически изменяемые разметки и адаптивные изображения

В давние времена (в конце 1990-х годов) ширину создаваемых сайтов обычно задавали в процентах. Указание ширины в процентах приводило к подстраиванию страниц к экрану просмотра и приобрело известность как резиновые разметки (fluid layouts).

Вскоре, во второй половине 2000-х годов, появилась промежуточная фиксация с использованием конструкций с фиксированной шириной (я виню в этом дотошных полиграфистов с их приверженностью к абсолютной точности в пикселах). Теперь же, поскольку мы создаем конструкции с адаптивным веб-дизайном, нам нужно обратиться в прошлое, к динамически изменяемым разметкам, и вспомнить обо всех получаемых благодаря им преимуществах.

В главе 2 мы в конечном итоге признали, что наряду с предоставляемой медиазапросами возможностью создания дизайна, адаптируемого к изменяющимся размерам окна просмотра за счет переключения от одного набора стилей к другому, нам нужна какая-либо возможность динамического изменения нашего дизайна в промежутках между контрольными точками, предоставляемыми медиазапросами. Эта возможность успешно реализуется путем программирования подстраиваемой разметки, способной легко растягиваться для заполнения разрывов между контрольными точками медиазапросов.

В 2015 году в нашем распоряжении имеются такие эффективные средства для создания сайтов с адаптивным веб-дизайном, которых прежде не было. Это новый модуль CSS-разметки под названием Flexible Box (больше известный как Flexbox), который в настоящее время приобрел довольно широкую поддержку со стороны браузеров, став полноценным средством повседневного использования.

Он способен на большее, нежели просто предоставлять механизм динамически изменяемой разметки. Хотите получить возможность легко и просто выровнять содержимое по центру, изменить порядок в источнике разметки и вообще относительно легко создавать великолепные разметки? Тогда механизм разметки Flexbox для вас. Основная часть этой главы посвящена Flexbox, она охватывает самые интересные из предлагаемых им возможностей.

Есть еще одна ключевая область адаптивного веб-дизайна, которую теперь можно рассмотреть намного предметнее, чем когда-либо раньше, — я имею в виду адаптивные изображения. Теперь уже есть точно определенные методы и синтаксис для отправки на устройства наиболее подходящей для их окон просмотра версии изображения. Последний раздел главы будет посвящен разъяснению порядка работы адаптивных изображений и изучению способов, благодаря которым мы можем заставить их работать на нас.

В этой главе будут рассмотрены следующие вопросы:

- преобразование фиксированных размеров в пикселах в пропорциональные размеры;
- существующие механизмы CSS-разметки и их недостатки;
- разбор модуля Flexible Box Layout Module и предоставляемых им преимуществ;
- изучение допустимого синтаксиса для переключения разрешения и режиссуры адаптивных изображений.

Преобразование дизайна с фиксированными размерами в пикселах в подстраиваемую пропорциональную разметку

Все графические составляющие, созданные в таких программах, как Photoshop, Illustrator, Fireworks (RIP) или Sketch, имеют фиксированные размеры в пикселах. Когда разработчик перерабатывает дизайн в подстраиваемую разметку для отображения в браузере, элементы дизайна нуждаются в преобразовании в пропорциональные величины.

Для этого преобразования есть красивая и простая формула, которую отец адаптивного веб-дизайна, Итан Маркотт (Ethan Marcotte), разместил в своей статье 2009 года Fluid Grids (http://alistapart.com/article/FLUIDGRIDS):

Цель / Среда = Результат.

Если все, что относится к математике, вас раздражает, воспринимайте эту формулу следующим образом: разделите нужные вам составные части на то, в чем они помещаются. Введем это в обиход в качестве понятия, позволяющего выполнить преобразование любой разметки с фиксированными размерами в ее адаптивные (подстраиваемые) эквиваленты.

Рассмотрим очень простую разметку страницы, предназначенную для просмотра на настольном компьютере. В идеале мы всегда будем подходить к разметке для настольного компьютера, начиная с разметки для более скромных экранов, но с целью приведения примеров пропорций рассмотрим две данные ситуации в обратном порядке.

Эта разметка имеет следующий вид.

Ее ширина — 960 пикселов. Заголовок и подвал (нижний колонтитул) выводятся на всю ширину разметки. Область слева имеет ширину 200 пикселов, а область справа — 100 пикселов. Даже при моей недостаточной математической подкованности я могу вас заверить, что средняя часть будет иметь ширину 660 пикселов. Нам нужно преобразовать среднюю и боковые части, приведя их к пропорциональным размерам.

Начнем с левой стороны. Она имеет ширину 200 единиц (в нашей формуле это цель). Разделим этот размер на 960 единиц (среду) и получим 0,2083 33333. Теперь, вычислив результат по формуле, нужно переместить десятичный разделитель на две позиции вправо. В результате получим 20,83 33333 %. Именно так 200 пикселов описываются в процентах к 960 пикселам.

А как насчет средней части? Здесь 660 единиц (цель), разделенные на 960 (среда), дают нам 0,6875. Переместим десятичный разделитель на две позиции вправо и получим 68,75 %. И наконец, правая сторона: 100 единиц (цель), разделенные на 960 (среда), дают 0,1041 66667. После перемещения десятичного разделителя получим 10,41 66667 %. Как видите, ничего сложного. Повторяйте за мной: цель, разделенная на среду, дает результат.

В качестве доказательства быстро создадим эту простую разметку в виде блоков в браузере. Код разметки можно увидеть в файле каталога example_03-01. Вот как выглядит код HTML:

```
<div class="Wrap">
    <div class="Header"></div>
    <div class="WrapMiddle">
        <div class="Left"></div>
        <div class="Middle"></div>
        <div class="Right"></div>
    </div>
    <div class="Footer"></div>
</div>
```

А вот как выглядит код CSS:

```
html,
body {
    margin: 0;
    padding: 0;
}

.Wrap {
    max-width: 1400px;
    margin: 0 auto;
}

.Header {
    width: 100%;
    height: 130px;
    background-color: #038C5A;
}

.WrapMiddle {
    width: 100%;
    font-size: 0;
}

.Left {
    height: 625px;
    width: 20.8333333%;
    background-color: #03A66A;
    display: inline-block;
}

.Middle {
    height: 625px;
    width: 68.75%;
    background-color: #bbbf90;
    display: inline-block;
}
```

```
.Right {
    height: 625px;
    width: 10.4166667%;
    background-color: #03A66A;
    display: inline-block;
}

.Footer {
    height: 200px;
    width: 100%;
    background-color: #025059;
}
```

Если пример кода открыть в браузере и изменить размер страницы, вы увидите, что размер средней части останется пропорциональным по отношению к другим частям. Можно также попробовать изменять значение свойства `max-width` для класса `.Wrap`, чтобы сделать окружающие размеры для разметки больше или меньше (в примере установлено значение 1400 пикселов).

СОВЕТ

Если при изучении разметки вас беспокоил вопрос, почему я не воспользовался такими семантическими элементами, как header, footer и aside, не стоит волноваться. Эти элементы HTML5 подробно рассматриваются в главе 4.

Теперь посмотрим, каким будет то же самое содержимое на экранах меньших размеров. Сначала оно динамически подстраивается под контрольную точку, после чего подстраивается под среду, а затем изменяется под ту разметку, которую мы уже видели. Окончательный вариант кода этой разметки можно увидеть в файле каталога example_03-02.

Суть замысла в том, что для экранов меньшего размера у нас будет одна колонка содержимого. Левосторонняя часть будет видна как область, выходящая за пределы холста, обычно эта область отводится под меню или что-то подобное, что не попадает на просматриваемое пространство экрана и выплывает при нажатии кнопки меню. Основное содержимое помещается ниже заголовка, еще ниже помещаются правосторонняя часть и, наконец, область подвала. В нашем примере левостороннюю область меню можно показать после щелчка в любом месте заголовка. Обычно при реализации схемы дизайна подобного рода для активизации бокового меню используется кнопка меню.

СОВЕТ

Для переключения класса тела документа я воспользовался небольшим сценарием на JavaScript. Но он не отвечает требованиям готовности к полноценному использованию, поскольку в JavaScript применяется обработчик события щелчка 'click', а в идеале должно быть предусмотрено что-нибудь для работы на сенсорном экране (для удаления задержки 300 мс, которая до сих пор имеется в устройствах, работающих под управлением iOS).

Как вы, наверное, и ожидали, применив только что приобретенные навыки создания медиазапросов, мы можем настроить окно просмотра, и дизайн просто

адаптируется под эту настройку, перейдя от одной разметки к другой и растягиваясь при этом в промежуточных позициях между этими двумя разметками.

Я не собираюсь показывать здесь весь код CSS, доступный в файле каталога example_03-02, и привожу лишь его пример, относящийся к левосторонней части:

```
.Left {
    height: 625px;
    background-color: #03A66A;
    display: inline-block;
    position: absolute;
    left: -200px;
    width: 200px;
    font-size: .9rem;
    transition: transform .3s;
}

@media (min-width: 40rem) {
    .Left {
        width: 20.8333333%;
        left: 0;
        position: relative;
    }
}
```

Как видите, сначала идет разметка без медиазапроса, предназначенная для небольшого экрана. Затем для более крупных экранов ширина задается в пропорциях, позиционирование задается относительным, а `left` присваивается значение `0`. Значения таких свойств, как `height`, `display` или `background-color`, переписывать не нужно, поскольку их мы не изменяем.

Дело сдвинулось с места. Мы объединили две уже рассмотренные основные технологии адаптивного веб-дизайна, преобразовав фиксированные размеры в пропорции и воспользовавшись медиазапросами для создания целевых CSS-правил, относящихся к размеру окна просмотра.

СОВЕТ

В предыдущем примере есть два примечательных момента. Во-первых, вам могла показаться странным необходимость учитывать все десятичные знаки после разделителя. Дело в том, что значения ширины будут в итоге преобразованы браузером в пикселы и их еще предстоит вычислить (например, более точно определить ширину вложенных элементов). Из этого следует, что я рекомендую никогда не трогать числа, стоящие после разделителя. Во-вторых, в реальном проекте мы предоставили бы что-нибудь на случай отключения JavaScript и сохранения необходимости просмотра содержимого меню. Такой сценарий еще будет прорабатываться в главе 8.

Зачем нам нужен Flexbox

Давайте углубимся в подробности использования модуля CSS Flexible Box Layouts, более известного как Flexbox.

Но полагаю, что перед этим было бы разумно рассмотреть недостатки существующих технологий разметки, таких как линейные блоки, плавающие элементы и таблицы.

Линейные блоки и свободное пространство

Самой большой проблемой использования линейных блоков (inline-block) является механизм разметки, выводящий пробелы между элементами HTML. Это не считается серьезном недостатком (хотя многие разработчики были бы не прочь узнать способ удаления пробела), но означает, что для удаления нежелательного пробела нужно прибегать к особым приемам, что, на мой взгляд, отнимает около 95 % времени. Существует множество способов решения этой проблемы, в предыдущем примере мы воспользовались обнулением размера шрифта — `'font-size zero'`, у которого есть собственные недостатки и ограничения. Но вместо перечисления всех возможных способов устранения свободных пространств при использовании линейных блоков я предлагаю ознакомиться со статьей неугомонного Криса Койера (Chris Coyier): http://css-tricks.com/fighting-the-space-between-inline-block-elements/.

Стоит также заметить, что простого способа центрирования содержимого внутри линейного блока по вертикали не существует. Кроме того, при использовании линейных блоков отсутствует способ получения двух одноуровневых элементов, один их которых имеет фиксированную ширину, а другой подстраивается для заполнения всего оставшегося пространства.

Плавающие элементы (floats)

Я ненавижу плавающие элементы. И этим все сказано. Их положительным свойством является довольно устойчивая работа при любых обстоятельствах. Но у них есть две весьма досадные особенности.

Во-первых, при указании ширины плавающих элементов в процентах вычисленная ширина не округляется всеми браузерами одинаково (некоторые округляют вверх, а некоторые — вниз). Это означает, что иногда разделы вопреки задуманному будут выпадать из общей картины, а иногда будут оставлять с одной из сторон портящие внешний вид пробелы.

Во-вторых, плавающие элементы приходится вычищать, поскольку в противном случае родительские блоки или элементы не схлопываются. Конечно, особого труда это не составляет, но все же служит постоянным напоминанием о том, что плавающие элементы не следует использовать в качестве надежного механизма разметки ни при каких условиях.

Table и table-cell

Не нужно путать `display: table` и `display: table-cell` с соответствующими элементами HTML. Эти CSS-свойства просто имитируют разметку своих собратьев из HTML и на структуру HTML не влияют.

Я считаю табличную CSS-разметку исключительно полезной. Как минимум она позволяет осуществлять единообразное и надежное выравнивание элементов по центру друг в друге. Кроме того, элементы, имеющие настройку отображения `display: table-cell` внутри элемента, имеющего настройку `display: table`, отлично справляются с расстановкой пустых пространств и не страдают от проблем округления, как плавающие элементы. А поддержка этих свойств уходит корнями аж к Internet Explorer 7!

Но и здесь не обошлось без ограничений. Зачастую возникает необходимость заключать элемент в дополнительную оболочку (чтобы получить четкое выравнивание по центру, ячейка таблицы (table-cell) должна находиться внутри элемента, настроенного как таблица (table)). Кроме того, элементы, имеющие настройку отображения `display: table-cell`, не могут переносить свое содержимое с одной строки на другую.

В заключение необходимо отметить, что ограничения имеются у всех существующих методов разметки. Но, к нашему глубокому удовлетворению, есть новые CSS-методы разметки, справляющиеся с данными проблемами, которые способны не только на это, но и на многое другое. Дуйте в фанфары, расстилайте красный ковер. К нам идет Flexbox.

Представляем Flexbox

Модуль Flexbox предназначен для устранения недостатков, имеющихся в каждом из упомянутых ранее механизмов отображения. Коротко перечислю его исключительные возможности.

- Он легко справляется с выравниванием содержимого по вертикали.
- Он может изменить порядок визуального отображения элементов.
- Он может автоматически заполнять пространство пробелами и выравнивать элементы внутри блока, автоматически распределяя между ними доступное пустое пространство.
- Он может заставить вас выглядеть на десять лет моложе (наверное, все же не может, но, в чем я смог лично убедиться, уменьшая количество проб и ошибок, он снижает у вас уровень стресса).

Тернистый путь к сегодняшнему Flexbox

Прежде чем Flexbox превратился в сегодняшнюю относительно стабильную версию, он прошел через несколько основных стадий. Рассмотрим, к примеру, изменения, произошедшие со времени выхода версии 2009 года (http://www.w3.org/TR/2009/WD-css3-flexbox-20090723/), версии 2011 года (http://www.w3.org/TR/2011/WD-css3-flexbox-20111129/) и версии 2014 года, на которой основаны приводимые здесь примеры (http://www.w3.org/TR/css-flexbox-1/). Можно заметить, что у этих версий имеются существенные синтаксические различия.

Наличие различных спецификаций означает, что есть три основные версии реализации. Скольким версиям следует уделить внимание, зависит от необходимого вам уровня поддержки со стороны браузеров.

Браузерная поддержка Flexbox

Сразу обозначим исключения: Internet Explorer 9, 8 и более ранних версий Flexbox не поддерживает.

Что же касается всех остальных браузеров, чью поддержку хотелось бы получить (и практически всех браузеров мобильных устройств), то у вас есть возможность

воспользоваться большинством, если не всеми свойствами Flexbox. Информацию о поддержке можно получить по адресу http://caniuse.com/.

Прежде чем углубиться в изучение Flexbox, нужно сделать небольшое, но важное отступление.

Кому-то все еще нужны префиксы

Я надеюсь, что, посмотрев на примеры Flexbox, вы отдадите должное пользе от его применения и вдохновитесь на его использование. Но вводить вручную весь код, необходимый для поддержки различных Flexbox-спецификаций, — задача не из легких. Вот пример, в котором я собираюсь задать три относящихся к Flexbox свойства и значения. Рассмотрим его код:

```
.flex {
    display: flex;
    flex: 1;
    justify-content: space-between;
}
```

Именно так свойства и значения будут выглядеть в самой последней версии синтаксиса. Но если нужно получить поддержку со стороны Android-браузеров (v4 и ниже), а также IE 10, потребуется следующий код:

```
.flex {
    display: -webkit-box;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
    -webkit-box-flex: 1;
    -webkit-flex: 1;
        -ms-flex: 1;
            flex: 1;
    -webkit-box-pack: justify;
    -webkit-justify-content: space-between;
       -ms-flex-pack: justify;
            justify-content: space-between;
}
```

Необходимость написания всего этого обусловлена тем, что в последние несколько лет во всех браузерах стали доступны экспериментальные версии новых функциональных возможностей, но с использованием так называемого префикса производителя (vendor prefix). У каждого производителя имелся собственный префикс. Например, `-ms-` для Microsoft, `-webkit-` для WebKit, `-moz-` для Mozilla и т. д. Для каждой новой функциональной возможности это означало необходимость написания нескольких версий одного и того же свойства, сначала в форме версий с префиксами, а в конце — в форме официальной версии консорциума W3C.

Результатом такого написания в веб-истории является код CSS, подобный приведенному в предыдущем примере. Это единственный способ получить функцию, работающую на широком круге устройств. В наши дни производители довольно редко добавляют префиксы, но в обозримом будущем нам придется жить, сообразуясь с реальностью существования множества браузеров, по-прежнему требу-

ющих использования префиксов для включения конкретных функций. Это возвращает нас к Flexbox, который является ярким примером необходимости указания префиксов производителей не только из-за нескольких версий от разных производителей, но и из-за различных спецификаций функций. Но понимание и запоминание всего, что необходимо для написания текущего формата и каждого предыдущего формата, не составляет полной картины сомнительных удовольствий.

Не знаю, как вы, а я предпочитаю тратить свое время на более продуктивные занятия, чем всякий раз записывать весь этот объем кода! Короче говоря, если вы собираетесь тратить свои нервы на использование Flexbox, потратьте время на настройку решения для автоматической расстановки префиксов.

Выбор решения для автоматизации расстановки префиксов

Чтобы сберечь нервы и аккуратно, без особых усилий добавить к CSS префиксы производителей, воспользуйтесь решением для автоматической расстановки префиксов. Лично я на данный момент предпочитаю пользоваться средством Autoprefixer (https://github.com/postcss/autoprefixer). Оно работает довольно быстро, легко настраивается и отличается высокой точностью работы.

Существуют версии Autoprefixer для большинства настроек, и вам не обязательно пользоваться средством создания на основе использования командной строки (например, Gulp или Grunt). К примеру, если пользоваться средством Sublime Text, то существует версия, которая будет работать непосредственно из его палитры команд: https://github.com/sindresorhus/sublime-autoprefixer. Есть также версии Autoprefixer для Atom, Brackets и Visual Studio.

С этого момента, если не понадобится конкретная иллюстрация, префиксы производителей в примерах кода использоваться не будут.

Разбираемся с возможностью динамического изменения

У Flexbox имеются четыре основные характеристики: направление (direction), выравнивание (alignment), определение порядка (ordering) и динамическое изменение (flexibility). Мы рассмотрим все эти характеристики и их взаимоотношения на нескольких примерах.

Это намеренно простые примеры, касающиеся простого перемещения блоков и их содержимого и позволяющие понять принципы работы Flexbox.

Текст, безупречно выровненный по вертикали

Этот первый пример использования Flexbox можно найти в файле каталога example_03-03.

Разметка выглядит следующим образом:

```
<div class="CenterMe">
    Hello, I'm centered with Flexbox!
</div>
```

А вот полное CSS-правило, придающее стиль этой разметке:

```
.CenterMe {
    background-color: indigo;
    color: #ebebeb;
    font-family: 'Oswald', sans-serif;
    font-size: 2rem;
    text-transform: uppercase;
    height: 200px;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

Большинство пар «свойство — значение» в этом правиле просто устанавливают цветовые решения и размер шрифта. Нам интересны лишь три свойства:

```
.CenterMe {
    /* другие свойства */
    display: flex;
    align-items: center;
    justify-content: center;
}
```

Если вы еще не пользовались Flexbox или какими-либо свойствами из родственной спецификации Box Alignment Specification (http://www.w3.org/TR/css3-align/), эти свойства, возможно, покажутся вам несколько странными. Рассмотрим, что делает каждое из них.

- `display: flex` — это хлеб с маслом Flexbox. А именно, простая настройка элемента на его принадлежность к Flexbox (в противоположность блоку-контейнеру, линейному блоку и т. д.).
- `align-items` — это свойство приводит к выравниванию элементов внутри Flexbox по поперечной оси (в нашем примере текст выравнивается по вертикали).
- `justify-content` — это свойство задает выравнивание содержимого по центру главной оси. Что касается строки Flexbox, то об этом свойстве можно думать как о кнопке в текстовом процессоре, выравнивающей текст по левому или правому краю или по центру (хотя существуют и другие значения `justify-content`, которые также вскоре будут рассмотрены).

Итак, перед тем, как продолжить изучение свойств Flexbox, рассмотрим еще несколько примеров.

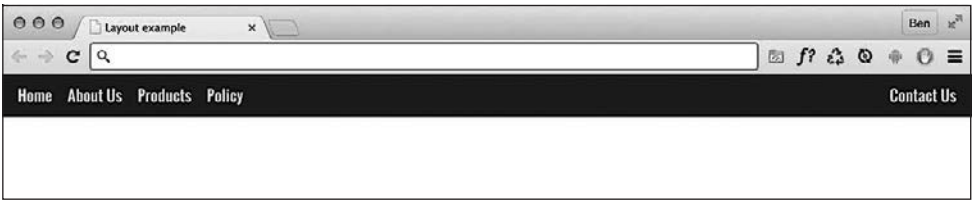
СОВЕТ

В некоторых из этих примеров использовался имеющийся у Google шрифт Oswald (с откатом к шрифту sans-serif). В главе 5 мы рассмотрим способы использования правила @font-face, предназначенного для привязки к заказным файлам шрифтов.

Смещение элементов

А как вам понравится простой список элементов перехода, в котором один из этих элементов смещен в другую сторону?

Вот как это выглядит.



Вот разметка:

```
<div class="MenuWrap">
    <a href="#" class="ListItem">Home</a>
    <a href="#" class="ListItem">About Us</a>
    <a href="#" class="ListItem">Products</a>
    <a href="#" class="ListItem">Policy</a>
    <a href="#" class="LastItem">Contact Us</a>
</div>
```

А вот код CSS:

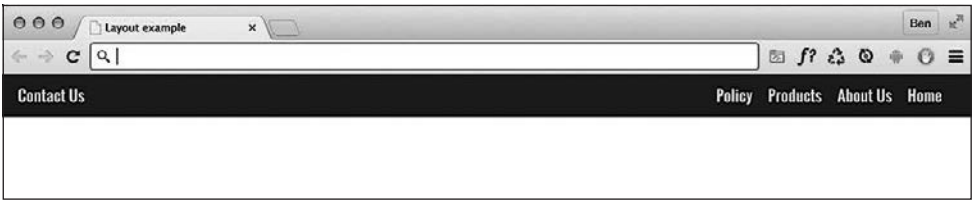
```
.MenuWrap {
    background-color: indigo;
    font-family: 'Oswald', sans-serif;
    font-size: 1rem;
    min-height: 2.75rem;
    display: flex;
    align-items: center;
    padding: 0 1rem;
}

.ListItem,
.LastItem {
    color: #ebebeb;
    text-decoration: none;
}
```

```
.ListItem {
    margin-right: 1rem;
}

.LastItem {
    margin-left: auto;
}
```

Как видите, не нужны никакие отдельно взятые плавающие элементы, линейные блоки или ячейки таблицы (table-cell)! При настройке `display: flex;`, примененной в отношении элемента-контейнера, его дочерние элементы становятся подстраиваемыми, выводимыми затем с использованием модели динамически изменяемой разметки. Все волшебство здесь заключается в свойстве `margin-left: auto`, которое заставляет этот элемент использовать все незаполненное место, доступное в данной стороне.

Изменение порядка следования элементов

Хотите изменить порядок следования элементов на обратный?



Можно просто добавить к элементу-контейнеру свойство `flex-direction: row-reverse;`, а для смещенного элемента вместо `margin-left: auto` написать `margin-right: auto`:

```
.MenuWrap {
    background-color: indigo;
    font-family: 'Oswald', sans-serif;
    font-size: 1rem;
    min-height: 2.75rem;
    display: flex;
    flex-direction: row-reverse;
    align-items: center;
    padding: 01rem;
}

.ListItem,
.LastItem {
    color: #ebebeb;
    text-decoration: none;
}

.ListItem {
    margin-right: 1rem;
}
```

```
.LastItem {
    margin-right: auto;
}
```

А что если нам захочется расположить элементы не горизонтально, а вертикально?

Да запросто. Измените свойство элемента-контейнера на `flex-direction: column;` и удалите автоматическую установку полей:

```
.MenuWrap {
    background-color: indigo;
    font-family: 'Oswald', sans-serif;
    font-size: 1rem;
    min-height: 2.75rem;
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 01rem;
}

.ListItem,
.LastItem {
    color: #ebebeb;
    text-decoration: none;
}
```

Изменение порядка следования элементов в столбце на обратный

Хотите расположить элементы друг под другом в обратном порядке? Измените настройку на `flex-direction: column-reverse;`, и все будет сделано как надо.

ПРИМЕЧАНИЕ

Следует отметить, что для установки в одной настройке значений сразу двух свойств, и flex-direction, и flex-wrap, можно воспользоваться сокращением в виде свойства flex-flow. Например, настройка flex-flow: row wrap; установит строчное направление и включит возможность размещения блоков в нескольких строках. Но я считаю, что по крайней мере на первых порах проще будет указать эти два свойства по отдельности. Свойство flex-wrap также отсутствует в более ранних реализациях, поэтому в определенных браузерах его объявление может игнорироваться.

Различные разметки Flexbox внутри разных медиазапросов

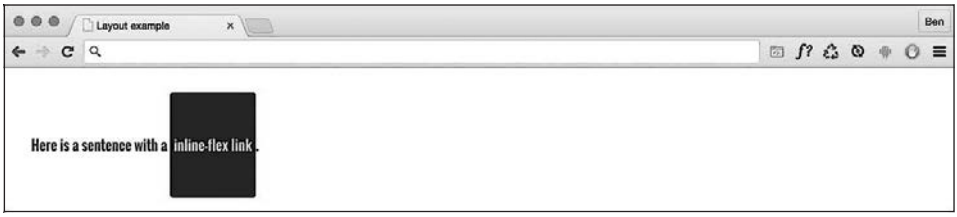
Что касается поведения списка элементов в столбце на экранах меньшего размера и стиля разметки строки, Flexbox в соответствии со своим названием изначально способен динамически изменять разметку. При использовании Flexbox эта задача существенно облегчается:

```
.MenuWrap {
    background-color: indigo;
    font-family: 'Oswald', sans-serif;
    font-size: 1rem;
    min-height: 2.75rem;
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 0 1rem;
}

@media (min-width: 31.25em) {
    .MenuWrap {
        flex-direction: row;
    }
}

.ListItem,
.LastItem {
    color: #ebebeb;
    text-decoration: none;
}

@media (min-width: 31.25em) {
    .ListItem {
        margin-right: 1rem;
    }
    .LastItem {
        margin-left: auto;
    }
}
```

Все это можно посмотреть, запустив в браузере файл каталога example_03-05. Чтобы увидеть различные разметки, нужно изменить размеры окна браузера.

inline-flex

У Flexbox имеется линейный вариант для дополнения линейных блоков (inline-block) и линейных таблиц (inline-table). Нетрудно догадаться, что для этого имеется объявление `display: inline-flex;`. Благодаря его превосходным возможностям по выравниванию по центру вы можете легко добиться весьма впечатляющих результатов.



Разметка выглядит следующим образом:

```
<p>Here is a sentence with a <a href="http://www.w3.org/TR/css-
flexbox-1/#flex-containers" class="InlineFlex">inline-flex link</a>.</p>
```

А вот для нее и код CSS:

```
.InlineFlex {
    display: inline-flex;
    align-items: center;
    height: 120px;
    padding: 04px;
    background-color: indigo;
    text-decoration: none;
    border-radius: 3px;
    color: #ddd;
}
```

Когда свойство `inline-flex` придается элементам анонимно (например, когда их родительский элемент не имеет настройки `display: flex;`), они сохраняют пробелы между элементами точно так же, как это происходит при использовании линейных блоков или линейных таблиц. Но если такие элементы находятся внутри flex-контейнера, пробелы удаляются, что уже похоже на элементы, являющиеся ячейками таблицы внутри самой таблицы.

Разумеется, никто вас не заставляет всегда центрировать элементы внутри Flexbox. Существует несколько различных вариантов. Их мы сейчас и рассмотрим.

Свойства выравнивания, имеющиеся у Flexbox

Если вы хотите попрактиковаться с этим примером, его можно найти в файле каталога example_03-07. Не забудьте, что загружаемый код примера будет находиться в том состоянии, к которому мы придем к концу раздела, поэтому, если есть желание проработать все самостоятельно, лучше удалить CSS из файла примера и приступить к его созданию с нуля.

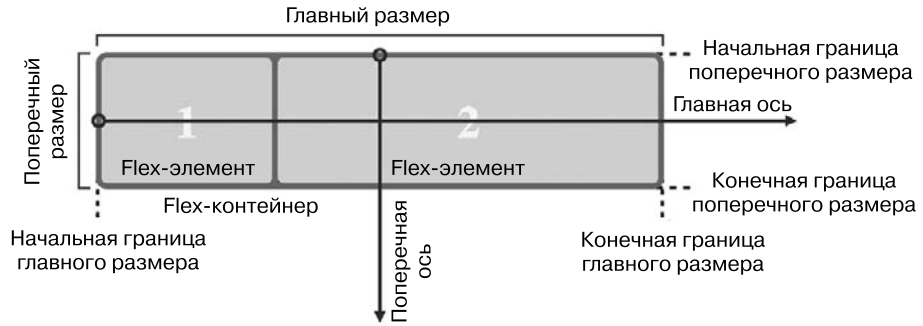
При изучении выравнивания, осуществляемого средствами Flexbox, важно разобраться с концепцией осей. В ней рассматриваются две оси: главная (main axis) и поперечная (cross axis). Что представляет собой каждая из них, зависит от направления, по которому распространяется Flexbox-контейнер. Например, если направление Flexbox-контейнера настроено на строку, главная ось будет горизонтальной, а поперечная ось — вертикальной.

И наоборот, если направление распространения вашего Flexbox-контейнера настроено на столбец, главная ось будет вертикальной, а поперечная ось — горизонтальной.

В помощь авторам в спецификации (http://www.w3.org/TR/css-flexbox-1/#justify-content-property) приводится следующая иллюстрация.

Основная разметка нашего примера имеет такой вид:

```
<div class="FlexWrapper">
    <div class="FlexInner">I am content in the inner Flexbox.</div>
</div>
```



Зададим основные стили, относящиеся к Flexbox:

```
.FlexWrapper {
    background-color: indigo;
    display: flex;
    height: 200px;
    width: 400px;
}

.FlexInner {
    background-color: #34005B;
    display: flex;
    height: 100px;
    width: 200px;
}
```

В браузере будет получена следующая картинка.

Хорошо, а теперь проведем тестирование ряда свойств.

Свойство align-items

Свойство `align-items` позиционирует элементы относительно поперечной оси. Если применить это свойство к нашему элементу-контейнеру следующим образом:

```
.FlexWrapper {
    background-color: indigo;
    display: flex;
    height: 200px;
    width: 400px;
    align-items: center;
}
```

то нетрудно представить, что элемент внутри контейнера получит вертикальное выравнивание по центру.

Точно такой же эффект будет применен к любому количеству находящихся внутри контейнера дочерних элементов.

Свойство align-self

Иногда нужно применить другую настройку выравнивания всего лишь к одному из элементов. Для самостоятельного выравнивания отдельных flex-элементов может использоваться свойство `align-self`. Сейчас я удалю прежние свойства выравнивания, добавлю в разметку еще два элемента, которые получат HTML-класс `.FlexInner`, а к среднему элементу добавлю еще один HTML-класс (`.AlignSelf`) и воспользуюсь им для добавления свойства `align-self`. Возможно, более наглядное представление об этом вы получите при просмотре следующего кода CSS:

```
.FlexWrapper {
    background-color: indigo;
    display: flex;
    height: 200px;
    width: 400px;
}

.FlexInner {
    background-color: #34005B;
    display: flex;
    height: 100px;
    width: 200px;
}

.AlignSelf {
    align-self: flex-end;
}
```

А вот такой эффект будет получен на экране браузера.

Великолепно! Flexbox действительно с легкостью справился с этими изменениями. В данном примере свойству `align-self` было присвоено значение `flex-end`. Рассмотрим возможные значения, которыми можно будет воспользоваться относительно поперечной оси, а потом рассмотрим возможности выравнивания относительно главной оси.

Возможные значения выравнивания

Для выравнивания относительно поперечной оси в модуле Flexbox имеются следующие значения:

- `flex-start` — настройка элемента на `flex-start` заставит его начинаться с начальной границы своего flex-контейнера;
- `flex-end` — настройка элемента на `flex-end` заставит его выровняться по конечной границе своего flex-контейнера;
- `center` — заставит элемент расположиться посредине flex-контейнера;
- `baseline` — заставит все flex-элементы в контейнере выровняться по нижним строкам;
- `stretch` — заставит элементы растянуться по размеру их flex-контейнера (по поперечной оси).

Есть подробное описание, касающееся использования этих свойств, поэтому, если что-то не получается, всегда обращайтесь к спецификации за любым самым актуальным вариантом применения: http://www.w3.org/TR/css-flexbox-1/.

Свойство justify-content

Выравнивание по главной оси управляется свойством `justify-content` (для элементов, не находящихся в Flexbox-контейнере, предлагается также свойство `justify-self`, см. http://www.w3.org/TR/css3-align/). У свойства `justify-content` могут быть следующие значения:

- flex-start;
- flex-end;
- center;
- space-between;
- space-around.

Действие первых трех из них для вас вполне ожидаемо. А вот что получается при использовании space-between и space-around, мы сейчас увидим. Рассмотрим следующую разметку:

```
<div class="FlexWrapper">
    <div class="FlexInner">I am content in the inner Flexbox 1.</div>
    <div class="FlexInner">I am content in the inner Flexbox 2.</div>
    <div class="FlexInner">I am content in the inner Flexbox 3.</div>
</div>
```

А затем рассмотрим следующий код CSS. Мы настроили каждый из трех flex-элементов (FlexInner) на ширину 25 %, заключив их во flex-контейнер (FlexWrapper), настроенный на ширину 100 %.

```
.FlexWrapper {
    background-color: indigo;
    display: flex;
    justify-content: space-between;
    height: 200px;
    width: 100%;
}
.FlexInner {
    background-color: #34005B;
    display: flex;
    height: 100px;
    width: 25%;
}
```

Все три элемента займут только 75 % доступного пространства, а свойство justify-content объясняет, что именно мы хотим получить от браузера относительно оставшегося пространства. Значение space-between заставит распределить свободное пространство поровну между элементами, а значение space-around заставит разместить его в равных долях вокруг элементов. Возможно, разобраться со всем этим помогут скриншоты — это работа значения space-between.

А это результат использования вместо него значения space-around.

Полагаю, что сомневаться в практической пользе этих двух значений у вас нет никаких оснований.

СОВЕТ

Имеющиеся в модуле Flexbox различные свойства выравнивания теперь приводятся и в описании спецификации CSS Box Alignment Module Level 3. Тем самым закладывается общая основа с эффектами выравнивания других свойств отображения, такими как display: block; и display: table;. Спецификация все еще остается в силе, поэтому проверяйте ее состояние, обращаясь по адресу http://www.w3.org/TR/css3-align/.

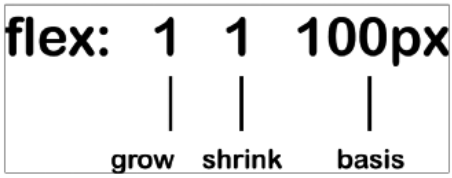
Свойство flex

Мы уже использовали в отношении flex-элементов свойство width, но свойство flex позволяет определить по вашему желанию либо ширину, либо адаптивность. Чтобы проиллюстрировать его возможности, рассмотрим следующий пример, в котором для элементов используется та же разметка, но уже с исправленным кодом CSS:

```
.FlexItems {
    border: 1px solid #ebebeb;
    background-color: #34005B;
    display: flex;
    height: 100px;
    flex: 1;
}
```

Фактически свойство flex является сокращенным определением трех отдельных свойств: flex-grow, flex-shrink и flex-basis. Более подробно спецификацию, в которой дается развернутое описание всех этих отдельных свойств, можно найти по адресу http://www.w3.org/TR/css-flexbox-1/. Но в этой спецификации рекомендуется использовать сокращенную форму в виде свойства flex, поэтому мы здесь его и рассмотрим.

Если для flex-элементов имеется свойство flex и браузер его поддерживает, то оно используется для указания размера элемента, а не значений ширины или высоты, если таковые имеются. Даже если после свойства flex указано значение ширины или высоты, оно все равно не будет действовать. Посмотрим, на что влияет каждое из этих значений.



- `flex-grow` (первое значение, которое может быть передано свойству `flex`) является величиной относительно других flex-элементов, до которой может увеличиваться flex-элемент при наличии свободного пространства.
- `flex-shrink` является величиной, до которой может уменьшаться flex-элемент относительно других flex-элементов, когда имеется дефицит пространства.
- `flex-basis` (последнее значение, которое может быть передано свойству `flex`) является базовым размером, к которому приводится размер flex-элемента.

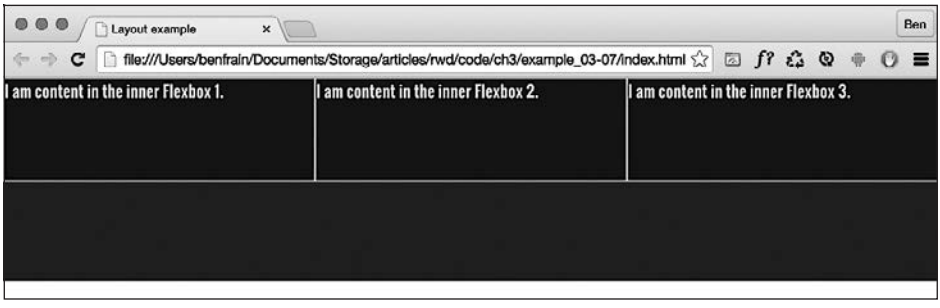
Хотя можно просто написать `flex: 1`, я рекомендую записывать в свойство `flex` все три значения. Полагаю, вы понимаете, что должно произойти. Например, `flex: 1 1 auto` означает, что элемент будет увеличиваться на одну часть доступного пространства, он также станет уменьшаться до одной части, когда будет существовать дефицит пространства, и базовым размером для динамического изменения будет служить внутренняя ширина содержимого (размер, который был бы у содержимого, если бы динамическое изменение не требовалось).

Рассмотрим еще один вариант: настройка `flex: 0 0 50px` означает, что элемент не будет ни увеличиваться, ни уменьшаться и его базовая величина составит `50px` (то есть он будет иметь размер 50 пикселов независимо от величины свободного пространства). А теперь разберем вариант `flex: 2 0 50%` — элемент будет стремиться занять два «лота» доступного пространства, он не станет уменьшаться и его базовый размер будет определяться значением `50%`. Надеюсь, эти небольшие примеры немного развеяли мистику вокруг свойства `flex`.

СОВЕТ

Если установить для свойства flex-shrink нулевое значение, то свойство flex-basis поведет себя как задание минимальной ширины.

Свойство `flex` можно представлять в виде способа задания отношений. Когда для каждого flex-элемента этому свойству присвоено значение `1`, то все они занимают одинаковое пространство.



Хорошо, но чтобы проверить теорию на практике, внесем изменения в HTML-классы нашей разметки:

```
<div class="FlexWrapper">
    <div class="FlexItems FlexOne">I am content in the inner Flexbox 1.</div>
    <div class="FlexItems FlexTwo">I am content in the inner Flexbox 2.</div>
    <div class="FlexItems FlexThree">I am content in the inner Flexbox 3.</div>
</div>
```

А так будет выглядеть измененный код CSS:

```
.FlexItems {
    border: 1px solid #ebebeb;
    background-color: #34005B;
    display: flex;
    height: 100px;
}

.FlexOne {
    flex: 1.50 auto;
}

.FlexTwo,
.FlexThree {
    flex: 10 auto;
}
```

В этом примере элемент с классом `FlexOne` занимает 1,5 того пространства, которое занято элементами с классами `FlexTwo` и `FlexThree`.

Этот сокращенный синтаксис проявляет свою реальную практичность для быстрой установки соизмеримости элементов. Например, если выдвигается требование «это должно быть в 1,8 раза шире всего остального», то его можно будет легко выполнить, используя свойство `flex`.

Надеюсь, вы уже начинаете понимать всю необычную эффективность свойства `flex`?

О Flexbox можно написать не одну главу! Примеров, достойных рассмотрения, великое множество. Но перед тем, как перейти к другой основной теме этой главы — адаптивным изображениям, мне хочется поделиться с вами еще двумя особенностями.

Простой зафиксированный подвал

Предположим, нужно, чтобы подвал находился в самом низу окна просмотра, даже когда основного содержимого недостаточно, для того чтобы он там оказался. Добиться нужного результата нелегко, но только не с Flexbox. Рассмотрим следующую разметку, которую можно увидеть в файле каталога example_03-08:

```
<body>
    <div class="MainContent">
        Here is a bunch of text up at the top. But there isn't enough
        content to push the footer to the bottom of the page.
```

```
    </div>
    <div class="Footer">
        However, thanks to flexbox, I've been put in my place.
    </div>
</body>
```

А вот так выглядит код CSS:

```
html,
body {
    margin: 0;
    padding: 0;
}

html {
    height: 100%;
}

body {
    font-family: 'Oswald', sans-serif;
    color: #ebebeb;
    display: flex;
    flex-direction: column;
    min-height: 100%;
}

.MainContent {
    flex: 1;
    color: #333;
    padding: .5rem;
}

.Footer {
    background-color: violet;
    padding: .5rem;
}
```

Посмотрите на окно браузера и попробуйте добавить содержимое в `.MainContentdiv`. Станет видно, что при недостаточном объеме содержимого подвал фиксируется в нижней части окна просмотра. Когда содержимого достаточно для заполнения окна, подвал располагается ниже содержимого.

Такая работа обусловливается тем, что наше свойство `flex` настроено на увеличение при наличии доступного пространства. Поскольку тело является flex-контейнером со 100 % минимальной высоты, основное содержимое может увеличиваться на все доступное пространство. Превосходно.

Изменение порядка следования исходных элементов

В CSS с момента появления этой технологии был только один способ переключения визуального порядка следования HTML-элементов на веб-странице. Такое

переключение достигалось за счет помещения элементов внутрь какого-нибудь контейнера с настройкой `display: table` с последующим переключением свойства `display` в отношении находящихся внутри элементов между `display: table-caption` (элемент помещался в верхнюю часть контейнера), `display: table-footer-group` (элемент помещался в нижнюю часть контейнера) и `display: table-header-group` (элемент помещался непосредственно под элементом с настройкой `display: table-caption`). Но, несмотря на всю грубость этой технологии, ее воспринимали как весьма удачное обстоятельство, не принимая во внимание истинное назначение таких настроек.

А вот в модуле Flexbox имеется встроенная функция изменения порядка следования отображаемых элементов. Посмотрим, как она работает.

Рассмотрим следующую разметку:

```
<div class="FlexWrapper">
    <div class="FlexItems FlexHeader">I am content in the Header.</div>
    <div class="FlexItems FlexSideOne">I am content in the SideOne.</div>
    <div class="FlexItems FlexContent">I am content in the Content.</div>
    <div class="FlexItems FlexSideTwo">I am content in the SideTwo.</div>
    <div class="FlexItems FlexFooter">I am content in the Footer.</div>
</div>
```

Обратите внимание на то, что третий элемент внутри общего контейнера имеет HTML-класс `FlexContent`, и представьте себе, что в этом `div`-контейнере предполагается размещать основное содержимое страницы.

Давайте не будем все усложнять. Добавим несколько простых цветовых настроек, чтобы было проще различать разделы, и поместим эти разделы в стопку в том самом порядке, в котором они появляются в разметке:

```
.FlexWrapper {
    background-color: indigo;
    display: flex;
    flex-direction: column;
}

.FlexItems {
    display: flex;
    align-items: center;
    min-height: 6.25rem;
    padding: 1rem;
}

.FlexHeader {
    background-color: #105B63;
}

.FlexContent {
    background-color: #FFFAD5;
}

.FlexSideOne {
    background-color: #FFD34E;
}
```

```
.FlexSideTwo {
    background-color: #DB9E36;
}

.FlexFooter {
    background-color: #BD4932;
}
```

Предположим, что нам нужно изменить порядок `.FlexContent`, сделав его первым разделом, не касаясь при этом разметки. С Flexbox эта задача решается простым добавлением всего лишь одной пары «свойство — значение»:

```
.FlexContent {
    background-color: #FFFAD5;
    order: -1;
}
```

Свойство `order` позволяет легко и просто пересмотреть порядок следования элементов внутри Flexbox. В данном примере значение `-1` говорит о том, что нам нужно поместить элемент перед всеми остальными элементами.

СОВЕТ

Если есть намерение часто менять элементы местами, я бы порекомендовал расширить описание и присвоить каждому элементу порядковый номер. Это упростит понимание задачи в сочетании с медиазапросами.

Объединим наши новые возможности по изменению порядка следования с медиазапросами, чтобы получилась не просто различная разметка для разных размеров, а разный порядок следования элементов.

ПРИМЕЧАНИЕ

Код примера в окончательном виде можно найти в файле каталога example_03-09.

Пересмотрим разметку и поместим основное содержимое в начале документа, поскольку такой порядок обычно считается более разумным:

```
<div class="FlexWrapper">
    <div class="FlexItems FlexContent">I am content in the Content.</div>
    <div class="FlexItems FlexSideOne">I am content in the SideOne.</div>
    <div class="FlexItems FlexSideTwo">I am content in the SideTwo.</div>
    <div class="FlexItems FlexHeader">I am content in the Header.</div>
    <div class="FlexItems FlexFooter">I am content in the Footer.</div>
</div>
```

Сначала идет содержимое страницы, затем две боковые области, после этого заголовок и, наконец, подвал. Поскольку используется Flexbox, мы можем структурировать HTML в том порядке, который имеет смысл для документа независимо от того, как все должно располагаться при отображении на экране.

Для самых мелких экранов (вне каких-либо медиазапросов) порядок будет такой:

```
.FlexHeader {
    background-color: #105B63;
    order: 1;
}

.FlexContent {
    background-color: #FFFAD5;
    order: 2;
}

.FlexSideOne {
    background-color: #FFD34E;
    order: 3;
}
```

```
.FlexSideTwo {
    background-color: #DB9E36;
    order: 4;
}

.FlexFooter {
    background-color: #BD4932;
    order: 5;
}
```

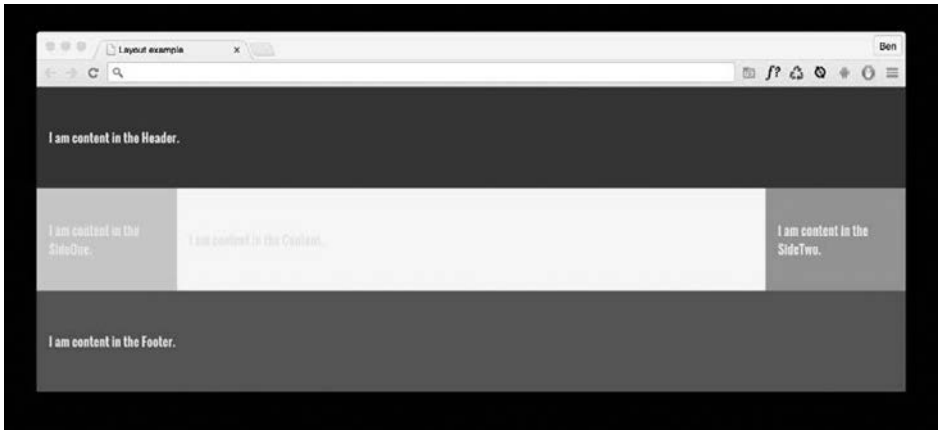
Соответственно, в браузере будет наблюдаться такая картина.

А затем в контрольной точке произойдет переход к следующим настройкам:

```
@media (min-width: 30rem) {
    .FlexWrapper {
        flex-flow: row wrap;
    }
```

```
    .FlexHeader {
        width: 100%;
    }
    .FlexContent {
        flex: 1;
        order: 3;
    }
    .FlexSideOne {
        width: 150px;
        order: 2;
    }
    .FlexSideTwo {
        width: 150px;
        order: 4;
    }
    .FlexFooter {
        width: 100%;
    }
}
```

Что в браузере будет выглядеть таким образом.



ПРИМЕЧАНИЕ

В данном примере использовалось сокращение flex-flow: row wrap. Это позволило flex-элементам располагаться на нескольких строках. Данное свойство относится к числу наименее поддерживаемых, поэтому в зависимости от того, насколько глубоко в прошлое должна простираться поддержка вашего продукта, может понадобиться заключить содержимое и две боковые панели в другой элемент.

Краткое заключение по Flexbox

Благодаря присущей Flexbox возможности внесения динамических изменений (flexiness) при использовании этой системы разметки мы располагаем практиче-

ски неисчерпаемыми возможностями, что великолепно подходит для создания адаптивного дизайна. Если раньше вам еще не приходилось создавать что-либо с использованием Flexbox, все новые свойства и значения покажутся немного странными, а та легкость, с какой получаются разметки, в которые прежде приходилось вкладывать намного больше труда, может вызвать небольшое замешательство. Чтобы еще раз проверить особенности реализации функциональных возможностей на соответствие самой последней версии спецификации, зайдите на сайт http://www.w3.org/TR/css-flexbox-1/.

Я думаю, что созидательная работа с использованием Flexbox вам понравится.

ПРИМЕЧАНИЕ

На пятки Flexible Box Layout Module уже наступает Grid Layout Module Level 1: http://www.w3.org/TR/css3-grid-layout/.

Этот модуль еще сыроват по сравнению с Flexbox (как и в ранней истории Flexbox, grid-разметка уже претерпела ряд существенных изменений), и поэтому мы не станем проявлять к нему пристальный интерес. Тем не менее его не нужно сбрасывать со счетов, поскольку этот модуль обещает дать еще более эффективные средства разметки веб-страниц.

Адаптивные изображения

Предоставление пользователям изображений, наиболее подходящих для конкретных характеристик применяемого устройства и среды, всегда было задачей не из легких. Особое звучание она приобрела с появлением адаптивного веб-дизайна, сама природа которого приспособлена для использования единой кодовой базы абсолютно для всех устройств.

Проблема, присущая адаптивным изображениям

Разработчик не может заранее знать или включать в свой план любое возможное устройство, с которого могут заходить на сайт сейчас или в будущем. Особенности устройства на момент обслуживания и вывода на экран содержимого (к примеру, размеры экрана и возможности устройства) известны только браузеру, который используется на этом устройстве.

И наоборот, только разработчик (вы и я) знает, какие версии изображения имеются в его распоряжении. Например, могут существовать три версии одного и того же изображения: малая, средняя и большая — и у каждой размер будет больше, чем у предыдущей, чтобы охватить множество вариантов размеров и плотностей экрана. Браузеру об этом ничего не известно. Наша задача заключается в доведении до него данной информации.

Подводя черту под этими осложнениями, можно констатировать, что у нас есть половина решения, то есть мы знаем, какими изображениями располагаем, а второй половиной решения располагает браузер, знающий, с какого устройства зашли на сайт и какие размер и разрешение изображения будут наиболее подходящими.

Как сообщить браузеру об имеющихся в нашем распоряжении изображениях, чтобы он мог выбрать самое подходящее для пользователя?

В первые несколько лет после появления адаптивного веб-дизайна какого-либо способа сделать это просто не было. Но, к общему удовольствию, теперь есть спецификация встроенного содержимого — Embedded Content: https://html.spec.whatwg.org/multipage/embedded-content.html.

В этой спецификации описываются способы работы с простым переключением разрешения изображений (чтобы упростить пользователю получение на экран с более высоким разрешением соответствующей улучшенной версии изображения) и варианты «режиссуры», при которых разработчики в зависимости от ряда характеристик устройства (возникают мысли о медиазапросах) хотят, чтобы пользователи видели совершенно другое изображение.

Продемонстрировать примеры адаптивных изображений довольно трудно. На одном и том же экране невозможно дать оценку различным изображениям, которые могут быть загружены с применением конкретных синтаксиса или технологии. Поэтому следующие примеры будут в основном в виде кода, и вам придется поверить мне на слово, что в поддерживающих эту технологию браузерах он будет выдавать нужный вам результат.

Рассмотрим два наиболее распространенных сценария, для которых, вероятнее всего, понадобятся адаптивные изображения. Они будут переключать изображение, когда потребуется другое разрешение, и полностью менять изображение в зависимости от доступного пространства окна просмотра.

Простое переключение разрешения с помощью srcset

Предположим, что имеются три версии изображения. Они выглядят одинаково, за исключением того, что одно из изображений имеет наименьшие размер или разрешение и предназначено для самых мелких окон просмотра, другое предназначено для окон просмотра среднего размера, и, наконец, самое большое изображение предназначено для всех остальных окон просмотра. О наличии у нас этих трех версий браузер оповещается следующим образом:

```
<img src="scones_small.jpg" srcset="scones_medium.jpg 1.5x, scones_
   large.jpg 2x" alt="Scones taste amazing">
```

Это почти так же просто, как и все то, что касается адаптивных изображений, поэтому давайте вникнем в смысл данного синтаксиса.

Сначала идет уже знакомый вам атрибут `src`, который здесь играет двойную роль. Он указывает на самую простую версию изображения, а также служит указателем на изображение отката, если браузер не поддерживает атрибут `srcset`. Именно поэтому он используется для указания самого простого изображения. Благодаря этому старые браузеры, игнорирующие информацию в `srcset`, получат самое простое и, возможно, самое динамичное изображение.

Браузерам, понимающим атрибут `srcset`, предоставляется список изображений, разделенных запятыми, из которых браузер может сделать выбор. После имени изображения (например, scones_medium.jpg) выдается простое указание на разре-

шение. В данном примере использованы указания 1.5x и 2x, но допускается применение любого целого числа. Например, 3x или 4x тоже будут работать — при условии, что отыщется подходящий экран высокого разрешения.

Но здесь есть одна проблема: устройство с шириной экрана 1440 пикселов и разрешением 1x получит то же самое изображение, что и устройство с шириной экрана 480 пикселов и разрешением 3x. Такой исход может оказаться нежелательным.

Более совершенный вариант переключения с использованием атрибутов srcset и sizes

Рассмотрим еще одну ситуацию. В адаптивном веб-дизайне довольно часто случается, что при небольших окнах просмотра изображение занимает все их пространство, а при окнах просмотра большего размера — всего лишь его половину. Это может быть проиллюстрировано основным примером из главы 1. Довести намерения до браузера можно следующим образом:

```
<img srcset="scones-small.jpg 450w, scones-medium.jpg 900w"
sizes="(min-width: 17em) 100vw, (min-width: 40em) 50vw"
src="scones-small.jpg" alt="Scones">
```

Внутри тега изображения снова используется srcset. Но на этот раз после указания изображений добавляется значение с суффиксом w. С его помощью браузеру сообщается о ширине изображения. В нашем примере имеются изображения шириной 450 пикселов (scones-small.jpg) и 900 пикселов (scones-medium.jpg). Важно отметить, что значение с суффиксом w не является настоящим размером. Это всего лишь указание браузеру, приблизительно соответствующее ширине в так называемых пикселах CSS.

СОВЕТ

Я задался вопросом: как именно определяется пиксел CSS? Затем нашел объяснение по адресу http://www.w3.org/TR/css3-values/ и понял, что лучше бы я этим вопросом не задавался.

Значение с суффиксом w обретает больший смысл, когда учитывается значение атрибута sizes. Этот атрибут позволяет сообщить браузеру о намерениях относительно изображений. В предыдущем примере первое значение равноценно следующей установке: «Для устройств, имеющих ширину как минимум 17 em, я хочу, чтобы показывалось изображение с параметрами, близкими к 100 vw».

ПРИМЕЧАНИЕ

Если вы не можете разобраться в некоторых используемых единицах измерения, например в vh (где 1 vh является эквивалентом 1 % высоты окна просмотра) и vw (где 1 vw является эквивалентом 1 % ширины окна просмотра), обратитесь к материалам главы 5.

Вторая часть фактически указывает: «Браузер, я хочу, чтобы для устройств шириной как минимум 40 em показывалось изображение шириной 50 vw». Пока учет ведется в dpi (или в dpr, что означает device pixel ratio), это может показаться

излишним. Например, на устройстве шириной 320 пикселов с разрешением 2x (при выводе на полную ширину для него требуется изображение шириной 640 пикселов) браузер может решить, что изображение шириной 900 пикселов подойдет больше, поскольку первый из имеющихся у него вариантов касается изображения, которое будет достаточно большим для заполнения требуемого размера.

Вы что, сказали, что браузер может предпочесть одно изображение другому?

Важно запомнить, что атрибуты `sizes` всего лишь дают браузеру подсказку. Это не гарантирует повиновения со стороны браузера. И это, уж поверьте, совсем неплохо и означает, что в будущем, если появится надежный способ для браузеров выяснить возможности сети, он сможет предпочесть одно изображение другому, поскольку к тому моменту будут известны обстоятельства, неизвестные нам как разработчикам программы. Возможно, пользователь настроит свое устройство на загрузку только изображений с разрешением 1x или только изображений с разрешением 2x, при таких сценариях браузер сможет выбрать вызов самого подходящего кода.

Альтернативой решению браузера является использование элемента `picture`. Использование этого элемента гарантирует использование браузером конкретно запрошенного вами изображения. Посмотрим на его работу.

Режиссура, применяемая в отношении элемента picture

И последний сценарий, с которым вы можете столкнуться, предполагает наличие разных изображений, применимых при различных размерах окон просмотра. Рассмотрим, к примеру, нашу булочку, взятую опять же из примера главы 1. Возможно, для самых маленьких экранов мы предпочтем булочку, изображенную крупным планом, щедро политую сверху вареньем и сливками. Для более крупных экранов, наверное, предпочтем воспользоваться более широким изображением. Возможно, это будет общий план стола с разнообразной выпечкой. И наконец, в еще более крупных окнах просмотра мы, наверное, предпочтем увидеть кондитерскую на деревенской улице, людей, сидящих возле нее за столиками, лакомящихся булочками и запивающих их чаем (понимаю, похоже на некую идиллию). Нам нужны три разных изображения, наиболее подходящих к различным диапазонам окон просмотра. Решить задачу с помощью элемента `picture` можно следующим образом:

```
<picture>
    <source media="(min-width: 30em)" srcset="cake-table.jpg">
    <source media="(min-width: 60em)" srcset="cake-shop.jpg">
    <img src="scones.jpg" alt="One way or another, you WILL get cake.">
</picture>
```

Во-первых, следует иметь в виду, что элемент `picture` — это просто контейнер, помогающий другим изображениям попасть в тег `img`. Если нужно придать изображениям то или иное стилевое оформление, переключите свое внимание на тег `img`.

Во-вторых, атрибут srcset работает здесь точно так же, как и в предыдущем примере.

В-третьих, тег img предоставляет альтернативное изображение, а также изображение, которое будет показано, если браузер распознает изображение, но не найдет соответствие ни одному из медиаопределений. Нужно четко уяснить, что не следует убирать тег img из контейнера picture, иначе все плохо кончится.

Основной особенностью контейнера picture является наличие в нем тега source. В этом теге можно применять выражения в стиле медиазапросов, конкретно сообщая браузеру, какой из ресурсов применять в той или иной ситуации. Например, в первом теге из предыдущего примера браузеру предписывается: «Если экран шириной не менее 30 em, нужно загрузить изображение cake-table.jpg». При совпадении условий браузер будет подчиняться предписаниям.

Продвижение новомодных форматов изображений. В качестве бонуса элемент picture также помогает вам в предоставлении альтернативных форматов изображений. WebP (информацию о котором можно получить на сайте https://developers.google.com/speed/webp/) является новейшим форматом, не получившим еще достаточного уровня поддержки во многих браузерах (http://caniuse.com/). Тем браузерам, которые обеспечивают его поддержку, можно предложить файл в этом формате, а тем, которые с этим не справляются, — файл в обычном формате:

```
<picture>
    <source type="image/webp" srcset="scones-baby-yeah.webp">
    <img src="scones-baby-yeah.jpg" alt="Again, you WILL eat cake.">
</picture>
```

Надеюсь, что теперь ситуация прояснилась. Вместо атрибута media используется атрибут type (с ним мы еще встретимся в главе 4), который, хотя и используется чаще всего для указания на видеоисточники (возможные типы видеоисточников можно найти на сайте https://html.spec.whatwg.org/multipage/embedded-content.html), позволяет нам в данном случае определить в качестве наиболее предпочтительного формат WebP. Если браузер в состоянии его отобразить, он это сделает, а если нет, возьмет исходное изображение, указанное в теге img.

СОВЕТ

Есть множество устаревших браузеров, которые никогда не смогут воспользоваться официально заявленными консорциумом W3C адаптивными изображениями. Мой совет: пока на то не будет конкретных причин, не отказывайтесь от возможностей использования встроенных резервных вариантов. Чтобы не портить пользователям таких браузеров впечатление от вашего сайта, воспользуйтесь резервными изображениями подходящих размеров, а усовершенствованные возможности оставьте для более способных на их реализацию устройств.

Резюме

В данной главе было рассмотрено множество важных вопросов. Много времени было потрачено на ознакомление с Flexbox, самой последней, эффективной и теперь уже широко поддерживаемой технологией. Были также рассмотрены способы

обслуживания для наших пользователей любого количества альтернативных изображений в зависимости от требующих решения задач. Благодаря применению `srcset`, `sizes` и `picture` пользователи всегда будут получать самые подходящие изображения, отвечающие их потребностям как сейчас, так и в будущем.

К этому моменту нами рассмотрены довольно большой объем кода CSS и некоторые возникающие в этой технологии перспективы и возможности, но более современной разметки мы касались только в отношении адаптивных изображений. Далее уделим внимание именно этому вопросу.

Следующая глава будет целиком посвящена HTML5. Мы разберемся с тем, что предлагается в этой версии языка, что изменилось по сравнению с его предыдущей версией, и в значительной мере с тем, как наилучшим образом воспользоваться новыми семантическими элементами для создания более понятных и осмысленных HTML-документов.

4 Использование HTML5 в целях разработки адаптивного веб-дизайна

Если вы предполагаете увидеть здесь руководство по использованию **интерфейсов прикладного программирования** (API) HTML5, то я хочу перефразировать высказывание из одного общеизвестного вестерна и сказать: «Я не ваш Хаклберри».

Я собираюсь рассмотреть с вами «словарную» часть HTML5, то есть семантику этого языка. Если конкретнее, то способ использования новых элементов HTML5 для описания содержимого, помещаемого в разметку. Основная часть содержимого главы не имеет прямого отношения к адаптивному веб-дизайну. Но HTML является основой, на которой строятся все веб-проекты и приложения. А кому же не хочется возводить свое здание на самом крепком из фундаментов?

К тому же вас может заинтересовать, что представляет собой HTML5. В таком случае я скажу, что HTML5 — это всего лишь название самой последней версии HTML, языка тегов, используемого для создания веб-страниц. Сам HTML является постоянно развивающимся стандартом, предыдущая основная версия которого имела номер 4.01.

О предшествующих версиях и путях развития HTML можно узнать в «Википедии» по адресу http://en.wikipedia.org/wiki/HTML#HTML_versions_timeline.

СОВЕТ

В настоящее время HTML5 является рекомендацией консорциума W3C. Соответствующую спецификацию можно найти по адресу http://www.w3.org/TR/html5/.

В этой главе будут рассмотрены следующие вопросы:

- насколько широка поддержка HTML5;
- как правильно написать начало страницы на HTML5;
- покладистость HTML5;
- новые семантические элементы;
- семантика на уровне текста;
- устаревшие функции;

- внедрение новых элементов;
- соответствие уровня доступности требованиям *Руководства по обеспечению доступности веб-контента* (Web Content Accessibility Guidelines (WCAG)) и *Стандарта предоставления возможности полноценного использования Интернета людьми с физическими ограничениями* (Web Accessibility Initiative-Accessible Rich Internet Applications (WAI-ARIA)) для повышения общей доступности веб-приложений;
- встраиваемое медиасодержимое;
- адаптивное видео и iFrames;
- замечание по поводу приоритета автономности.

ПРИМЕЧАНИЕ

В HTML5 также предоставляются конкретные инструменты для работы с формами и пользовательским вводом. Набор соответствующих функций существенно разгружает более ресурсоемкие технологии вроде JavaScript в вопросах проверки правильности заполнения форм. Хотя формы HTML5 будут рассмотрены отдельно в главе 9.

Разметку на HTML5 понимают все современные браузеры

Сегодня основная масса просматриваемых мною сайтов (и все сайты, созданные моими руками) написаны с использованием HTML5, без применения старого стандарта HTML 4.01.

Новые семантические элементы HTML5 (новые элементы структуры, видео- и аудиотеги) понятны всем современным браузерам, и даже старые версии Internet Explorer (предшествующие Internet Explorer 9) могут пользоваться небольшими дополнениями (полифиллами), позволяющими выводить эти новые элементы.

ЧТО ТАКОЕ ПОЛИФИЛЛЫ?

Термин «полифилл» придуман Реми Шарпом (Remy Sharp) как намек на заполнение трещин в старых браузерах с помощью Polyfilla (известной в США шпатлевки). То есть полифилл является своеобразной прокладкой, написанной на JavaScript и предназначенной для эффективного воспроизводства новейших функций в устаревших браузерах. Но при этом важно понимать, что полифиллы утяжеляют код. Поэтому даже возможность добавления 15 полифиллных сценариев, способных заставить Internet Explorer 6 выводить сайт на экран не хуже любых других браузеров, не означает необходимости подобного добавления.

Если нужно включить в работу структурные элементы HTML5, то я бы посоветовал обратить внимание на исходный сценарий Реми Шарпа (http://remysharp.com/2009/01/07/html5-enabling-script/) или создание заказной сборки Modernizr (http://modernizr.com). Если инструментальное средство Modernizr вам еще не попадалось или не использовалось вами, то в следующей главе ему посвящен целый раздел.

А теперь, имея в виду все вышеизложенное, рассмотрим начало страницы на HTML5 и разберемся во всех открывающих тегах и их предназначении.

Как правильно написать начало страницы на HTML5

Начнем с самых первых строчек документа на HTML5. Стоит здесь ошибиться, и вы будете долго удивляться, почему ваша страница не ведет себя нужным образом. Первые несколько строк должны иметь примерно следующий вид:

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

Разберем эти теги по отдельности. Вообще-то они не будут меняться при каждом создании веб-страницы, но, вы уж мне поверьте, узнать об их предназначении будет весьма полезно.

doctype

Объявление `doctype` предназначено для того, чтобы сообщить браузеру о типе имеющегося документа. Иначе он не получит необходимые ему сведения о том, как нужно распорядиться содержимым этого документа.

Наш документ открывается HTML5-объявлением `doctype`:

```
<!DOCTYPE html>
```

Если вам нравится использовать нижний регистр, можете записать объявление в виде `<!doctype html>`. Разницы никакой.

По сравнению с HTML 4.01, где в начале страницы использовалось что-то вроде

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

это изменение было весьма долгожданным. Прежнее объявление вызывало стойкое неприятие, и я обычно переносил его целиком с предыдущих страниц.

А вот в HTML5 объявление `doctype` радует своей краткостью, ограничиваясь лишь формой `<!DOCTYPE html>`. Интересный факт (во всяком случае, для меня): все закончилось тем, что был избран самый короткий способ сообщения браузеру о необходимости вывода страницы в стандартном режиме.

СОВЕТ

Если вас интересует, что такое режим совместимости и стандартный режим, зайдите на страницу «Википедии» по адресу http://en.wikipedia.org/wiki/Quirks_mode.

Тег HTML и атрибут lang

После объявления `doctype` открывается тег `html`, являющийся корневым тегом документа. В нем также используется атрибут `lang`, указывающий на язык документа, а после него открывается раздел `<head>`:

```
<html lang="en">
<head>
```

Указание альтернативных языков

В соответствии со спецификациями W3C (http://www.w3.org/TR/html5/dom.html#the-lang-and-xml:lang-attributes) атрибут `lang` указывает основной язык содержимого элементов и любого из атрибутов элемента, содержащего текст. Если вы не пишете страницы на английском, то лучше указать правильный код языка. Например, для японского HTML-тег должен выглядеть так: `<html lang="ja">`. Полный перечень языков можно найти на сайте http://www.iana.org/assignments/language-subtag-registry.

Кодировка символов

В заключение указывается кодировка символов. Поскольку это пустой элемент (в котором ничего не может содержаться), закрывающий тег ему не нужен:

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

Для `charset` без веской причины для чего-нибудь другого почти всегда указывается значение `utf-8`. Любознательные могут найти дополнительные сведения по этому вопросу на сайте http://www.w3.org/International/questions/qa-html-encoding-declarations#html5charset.

Покладистость HTML5

Помню, в школьные времена, когда порой наш учитель математики, по прозвищу Супермин (образовано от его фамилии Mean, то есть «средний», хотя на самом деле он был весьма неплохим учителем), не выходил на работу, все вздыхали с облегчением. В отличие от мистера Мина (фамилия изменена, дабы не нанести вреда) заменявший его учитель был покладистым и дружелюбным человеком. Он сидел тихо и обходился без крика и постоянных назиданий. Он не требовал тишины во время занятий и особо не зацикливался на нашем поведении, если мы вникали в способы решения предлагаемых им задач. Для него имели смысл наши ответы и объяснение того, как мы к ним пришли. Если бы HTML5 был учителем математики, он был бы именно таким вот покладистым по характеру. И я сейчас поясню эту странную аналогию.

Если уделяется особое внимание способу написания кода, то обычно применяются символы в нижнем регистре, значения атрибутов заключаются в кавычки, а для сценариев и таблиц стилей используется объявление типа (`type`). Например, вполне возможно, что для ссылки на таблицу стилей используется следующая запись:

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5 не требует подобной строгости, ему вполне достаточно и такой записи:

```
<link href=CSS/main.css rel=stylesheet >
```

Заметили? В теге отсутствует закрывающий слеш, нет кавычек вокруг значений атрибутов и нет объявления типа. Но покладистому HTML5 этого и не надо. Второй пример так же работоспособен, как и первый.

Такой нестрогий синтаксис применяется по всему документу, а не только в ссылках на ресурсы. Например, если хотите, можете указать на div-контейнер следующим образом:

```
<div id=wrapper>
```

И это будет вполне приемлемый код HTML5. То же самое касается вставки изображения:

```
<img SRC=frontCarousel.png aLt=frontCarousel>
```

Это также вполне приемлемый код HTML5. Без закрывающего тег слеша, без кавычек и с использованием вперемешку символов в нижнем и верхнем регистре. Можно даже опустить такие элементы, как открывающий тег <head>, и страница все равно будет полноценной. Что бы сказали об этом приверженцы XHTML?

СОВЕТ

Хотите сократить код до норм, допустимых в HTML5? Присмотритесь к набору HTML5 Boilerplate (http://html5boilerplate.com/). Это готовый файл HTML5, выдержанный в лучших традициях этого языка, включающий свойственные ему стили, полифиллы и такие дополнительные инструментальные средства, как Modernizr. Просматривая код, можно почерпнуть множество полезных подсказок, кроме того, можно получить заказную сборку шаблона, соответствующую именно вашим потребностям. Настоятельно рекомендую!

Разумный подход к разметке на HTML5

Лично мне нравится записывать разметку в стиле языка XHTML. Это означает использование закрывающих тегов, заключение значений атрибутов в кавычки и приверженность к постоянному использованию одного и того же регистра символов. Можно утверждать, что избавление от подобной практики позволит сэкономить несколько байтов данных, но для экономии существует ряд инструментальных средств (при необходимости с их помощью могут быть убраны все ненужные символы и данные). Я же хочу, чтобы моя разметка была как можно разборчивее, и призываю всех поступать точно так же. Я считаю, что нельзя стремиться к краткости кода в ущерб его ясности.

Поэтому я полагаю, что при написании документов HTML5 вы можете создавать понятный и удобочитаемый код и в то же время пользоваться экономией, предлагаемой HTML5. К примеру, для ссылки на CSS я бы воспользовался следующим кодом:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

Я сохранил слеш, закрывающий тег и кавычки, но обошелся без атрибута type. В данном случае нужно выдерживать ту меру, которая для вас наиболее комфортна. HTML5 не станет на вас кричать, размахивать вашей разметкой перед всем классом и ставить вас в угол, как нерадивого ученика, не справившегося с заданием (неужели все это было только в моей школе?). Но вам хочется написать свою разметку на отлично.

Думаете, я шучу? Хочу, чтобы вы именно сейчас узнали, что если станете записывать код без кавычек вокруг значений атрибутов и не закрывать теги, то заслужите мое молчаливое осуждение.

СОВЕТ

Несмотря на свободный синтаксис HTML5, никогда не помешает проверить допустимость разметки. Правильная разметка обладает большей доступностью. Для этих целей был создан механизм проверки на соответствие стандартам W3C, который можно найти на сайте http://validator.w3.org/.

Ну хватит, наверное, мне уже осуждать новаторов разметки. Посмотрим на те преимущества, которые дает нам HTML5.

Приветствую могучий тег <a>

В HTML5 теперь можно сильно сэкономить, заключив в тег <a> сразу несколько элементов (и давно пора было именно так и сделать). Раньше, чтобы соблюсти правила разметки, необходимо было заключать каждый элемент в собственный тег <a>. Рассмотрим, к примеру, следующий код HTML 4.01:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</
a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" /></a>
```

В HTML5 можно избавиться от всех индивидуальных тегов <a> и заключить всю группу в один такой тег:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

Но следует помнить об одном вполне понятном ограничении: нельзя заключать один тег <a> в другой такой же тег (просто потому, что это глупо) или в другой интерактивный элемент, например в кнопку button (поскольку это глупо вдвойне!), нельзя также заключать в тег <a> форму (почему — догадайтесь сами).

Новые семантические элементы в HTML5

Если посмотреть, как определяется слово «семантика» в словаре OS X, мы увидим следующее:

> «Раздел лингвистики и логики, занимающийся определением смысла».

Для нас семантика — это придание смысла разметке. Почему так важен этот процесс? Буду рад, если именно это вы и собирались спросить.

Большинство сайтов придерживаются довольно стандартных структурных соглашений, при которых типичными областями являются заголовок (header), подвал (footer), боковая панель (sidebar), панель навигации (navigation bar) и т. д. Мы, как

разработчики веб-приложений, зачастую называем используемые `div`-контейнеры так, чтобы конкретнее обозначить эти области (например, `class="Header"`). Но что касается использования самого кода, любой пользовательский агент (браузер, программа для чтения содержимого экрана, поисковая машина и т. д.), просматривающий этот код, не может с уверенностью определить предназначение каждого из этих `div`-элементов. Пользователям вспомогательных технологий также было бы трудно отличить один `div`-контейнер от другого. В HTML5 для решения этой задачи имеются новые семантические элементы.

ПРИМЕЧАНИЕ

Полный перечень HTML5-элементов можно получить абсолютно свободно, если указать в браузере адрес http://www.w3.org/TR/html5/semantics.html#semantics.

Я не стану рассматривать здесь абсолютно все новые элементы, ограничусь только теми из них, которые, на мой взгляд, наиболее полезны или интересны в повседневной работе с использованием адаптивного веб-дизайна. Перейдем к их изучению.

Элемент <main>

В HTML5 долго не было элемента для выделения основного содержимого страницы. Внутри тела веб-страницы им будет обозначаться элемент, в котором находится основной блок содержимого.

Вначале утверждалось, что содержимое, находящееся за пределами одного из прочих новых семантических элементов HTML5, будет по принципу обратной логики принадлежать основному содержимому. К счастью, спецификация изменилась и теперь у нас есть более описательный способ группировки основного содержимого, называемый тегом `<main>`.

Что бы ни заключалось в этот элемент, основное содержимое страницы или основной раздел веб-приложения, он послужит для группировки всего основного. Вот как звучит весьма полезное положение из спецификации:

> «Область основного содержимого документа включает уникальную для этого документа информацию и исключает содержимое, повторяющееся в наборе документов, такое как имеющиеся на сайте навигационные ссылки, информация об авторском праве, логотипы и баннеры сайта и формы для поиска данных (если только основной функцией документа или приложения не является именно поиск данных)».

Стоит также отметить, что на странице не может быть более одной основной области (в конечном счете у вас же не может быть двух частей основного содержимого) и она не может быть использована в качестве потомка некоторых других семантических элементов HTML5, таких как `article`, `aside`, `header`, `footer`, `nav` или `header`. А вот они могут находиться внутри элемента `main`.

ПРИМЕЧАНИЕ

Официальные положения, касающиеся элемента main, можно найти на сайте http://www.w3.org/TR/html5/grouping-content.html#the-main-element.

Элемент <section>

Элемент <section> используется для определения универсального раздела документа или приложения. Например, можно создавать разделы, включающие содержимое: один раздел для контактной информации, еще один раздел для канала новостей и т. д. Важно понимать, что этот элемент не предназначен для придания стилевого оформления. Если нужно поместить элемент в контейнер просто для его стилевой настройки, можно, как и прежде, использовать div-элемент.

При работе с веб-приложениями я предпочитаю использовать section в качестве элемента-контейнера для визуальных компонентов, поскольку он предоставляет простой способ обозначения начала и конца компонентов в разметке.

Для себя можно также решить, будет ли элемент section использоваться там, где разделяемое содержимое имеет внутри себя естественный заголовок (например, h1). Если такого заголовка нет, то лучше, наверное, будет воспользоваться div-контейнером.

ПРИМЕЧАНИЕ

Чтобы посмотреть, что об элементе <section> говорится в спецификации W3C HTML5, обратитесь по адресу http://www.w3.org/TR/html5/sections.html#the-section-element.

Элемент <nav>

Элемент <nav> используется в качестве контейнера основных навигационных ссылок на другие страницы или фрагменты внутри той же самой страницы. Он не предназначен исключительно для использования в подвалах (хотя может применяться и там) и им подобных структурах, где обычно располагаются ссылки на другие страницы.

Если вы обычно размечаете свои элементы навигации в виде неупорядоченного списка (<ul>) и набора списочных тегов (li), то вместо этого вам могут больше подойти элемент nav и несколько вложенных тегов <a>.

ПРИМЕЧАНИЕ

Чтобы посмотреть, что об элементе <nav> говорится в спецификации W3C HTML5, перейдите в браузере по адресу http://www.w3.org/TR/html5/sections.html#the-nav-element.

Элемент <article>

Элемент <article> вполне можно перепутать с элементом <section>. Прежде чем понять разницу между ними, мне пришлось несколько раз перечитать спецификацию. Переложение спецификации в моем исполнении звучит следующим образом: элемент <article> используется в качестве контейнера для законченных фрагментов содержимого. При структурировании страницы нужно задаться вопросом: можно ли содержимое, которое вы собираетесь использовать внутри элемента <article>, переместить на другой сайт в виде цельного фрагмента, без потери завершенности? Можно ли

также представить содержимое, рассматриваемое для помещения внутрь элемента <article>, как отдельную статью для RSS-канала? Вполне очевидными примерами содержимого, пригодного для помещения в элемент <article>, могли бы стать публикации в блогах или новостные сообщения. Нужно понимать, что вложенные элементы <article> прежде всего имеют отношение к внешним элементам <article>.

ПРИМЕЧАНИЕ

Чтобы узнать, что говорится в спецификации W3C HTML5 об элементе <article>, нужно зайти на сайт http://www.w3.org/TR/html5/sections.html#the-article-element.

Элемент <aside>

Элемент <aside> используется для содержимого, которое косвенно касается окружающего его контента. На практике я часто пользуюсь им для боковых панелей (когда в них содержится подходящий контент). Этот элемент также считается подходящим для цитат, рекламных вставок и групп элементов навигации. В основном в качестве содержимого элемента aside вполне подойдет все, что не имеет прямого отношения к основному содержимому. Полагаю, что при использовании сайта электронной торговли первейшими кандидатами для <aside> вполне могут стать области типа «клиенты, купившие этот товар, приобрели также...».

ПРИМЕЧАНИЕ

Дополнительные сведения из спецификации W3C HTML5 относительно элемента <aside> можно найти по адресу http://www.w3.org/TR/html5/sections.html#the-aside-element.

Элементы <figure> и <figcaption>

В спецификации, относящейся к элементу <figure>, говорится следующее:

> «...таким образом, он может использоваться для аннотации иллюстраций, диаграмм, фотографий листингов кода и т. д.».

С его помощью корректуру части разметки из первой главы можно выполнить следующим образом:

```
<figure class="MoneyShot">
  <img class="MoneyShotImg" src="img/scones.jpg" alt="Incredible scones" />
  <figcaption class="ImageCaption">Incredible scones, picture from
     Wikipedia</figcaption>
</figure>
```

Как видите, элемент <figure> используется в качестве контейнера этого небольшого законченного блока. Внутри него, чтобы предоставить подпись для родительского элемента <figure>, используется элемент <figcaption>.

Этот элемент пригодится тогда, когда изображениям или коду нужны небольшие сопроводительные подписи, которые не будут соответствовать основному тексту содержимого.

ПРИМЕЧАНИЕ

Спецификацию, касающуюся элемента <figure>, можно найти по адресу http://www.w3.org/TR/html5/grouping-content.html#the-figure-element.

А спецификация на <figcaption> находится по адресу http://www.w3.org/TR/html5/grouping-content.html#the-figcaption-element.

Элементы <details> и <summary>

Вам, наверное, не раз хотелось создать на странице простой открывающийся и закрывающийся виджет, то есть фрагмент краткого изложения, по щелчку на котором открывается панель с дополнительной информацией. HTML5 облегчает создание такой модели с помощью элементов `<details>` и `<summary>`.

Рассмотрим следующую разметку (для ее практической проверки можно открыть пример, находящийся в файле example3.html, входящем в каталог кода для данной главы):

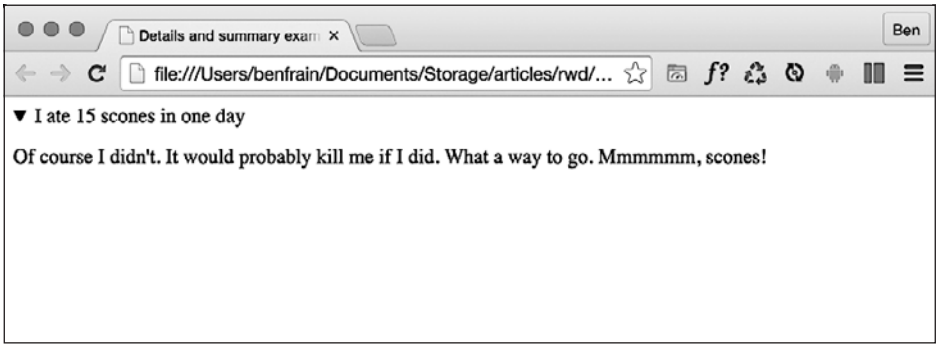
```
<details>
    <summary>I ate 15 scones in one day</summary>
    <p>Of course I didn't. It would probably kill me if I did. What a
        way to go. Mmmmmm, scones!</p>
</details>
```

Этот файл, открытый в браузере Chrome без добавления какого-либо стиля, покажет изначально только текст элемента `summary`.

При щелчке на любом месте текста элемента `<summary>` открывается панель. При повторном щелчке эта панель закрывается. Если нужно чтобы изначально панель была открыта, к элементу `<details>` можно добавить атрибут `open`:

```
<details open>
    <summary>I ate 15 scones in one day</summary>
    <p>Of course I didn't. It would probably kill me if I did. What a
```

```
        way to go. Mmmmmm, scones!</p>
</details>
```



Поддерживающие данные элементы браузеры обычно добавляют некоторое исходное стилевое оформление, чтобы указать на возможность открытия панели. В данном случае в браузере Chrome (а также Safari) будет выведен темный треугольник открытия. Чтобы выключить это оформление, нужно воспользоваться фирменным для WebKit псевдоселектором:

```
summary::-webkit-details-marker {
    display: none;
}
```

Разумеется, этот же селектор можно использовать для другого стилевого оформления маркера.

В данный момент способ анимации открытия и закрытия отсутствует. Не существует также (без использования кода JavaScript) способа закрытия других панелей подробностей (отображаемых на том же уровне) при открытии какой-нибудь другой панели. Я не уверен в том, что какое-либо из этих пожеланий когда-нибудь будет (или должно быть) воплощено в жизнь. Это следует воспринимать в основном как способ подсказать, что можно было бы сделать с помощью свойства `display: none;`, переключаемого с помощью кода JavaScript.

К сожалению, на момент работы над этим текстом (в середине 2015 года) поддержка этих элементов в Firefox или Internet Explorer отсутствовала (они просто выводили оба элемента в качестве линейных). Есть, правда, полифиллы (https://mathias\bynens.be/notes/html5-details-jquery), но я надеюсь, что вскоре появится и полноценная поддержка.

Элемент <header>

На практике элемент `<header>` может использоваться для области титульных данных заголовка сайта. Он также может использоваться в качестве введения в остальное содержимое, например в раздел, заключенный в элемент `<article>`. На одной странице его можно использовать столько раз, сколько нужно (к примеру, на вашей странице элемент `<header>` может быть внутри каждого элемента `<section>`).

ПРИМЕЧАНИЕ

Узнать, что говорится об элементе <header> в спецификации W3C HTML5, можно по адресу http://www.w3.org/TR/html5/sections.html#the-header-element.

Элемент <footer>

Элемент <footer> нужно использовать для содержания информации о разделе, в котором он находится. В нем могут содержаться ссылки на другие документы или, к примеру, информация об авторском праве. Если нужно, то он, как и элемент <header>, может использоваться на одной странице многократно. Например, его можно использовать для подвала блога, а также как подвальную часть в отдельной публикации блога. Но в спецификации объясняется, что контактная информация автора публикации в блоге должна вместо него помещаться в элемент <address>.

ПРИМЕЧАНИЕ

Все, что говорится в спецификации W3C HTML5 об элементе <footer>, можно найти по адресу http://www.w3.org/TR/html5/sections.html#the-footer-element.

Элемент <address>

Элемент <address> должен использоваться исключительно для разметки контактной информации для своего ближайшего предка, <article> или <body>. Чтобы не возникало путаницы, следует иметь в виду, что он не предназначен для размещения почтовых адресов и тому подобной информации (если только они действительно не будут представлять собой контактные адреса для рассматриваемого содержимого). Почтовые адреса и другая произвольная контактная информация должна заключаться в старые добрые теги <p>.

Я не сторонник использования элемента <address>, поскольку по собственному опыту знаю, что было бы намного полезнее размещать в собственном элементе именно физический адрес, но это мое личное мнение. Надеюсь, вы сможете найти больше смысла в применении данного элемента.

ПРИМЕЧАНИЕ

Чтобы узнать, что еще говорится в спецификации W3C HTML5 об элементе <address>, обратитесь по адресу http://www.w3.org/TR/html5/sections.html#the-address-element.

Замечания относительно элементов <h1> — <h6>

Только в последнее время я узнал, что для разметки заголовков и подзаголовков использовать теги <h1> — <h6> не рекомендуется. Я имею в виду следующее:

```
<h1>Scones:</h1>
<h2>The most resplendent of snacks</h2>
```

А вот цитата из спецификации HTML5:

> «Элементы <h1> — <h6> не должны использоваться для разметки подзаголовков, субтитров, альтернативных названий и слоганов, если только они не рассматриваются в качестве заголовка для нового раздела или подраздела».

И это еще не самое неоднозначное положение в спецификации!

Как же нам поступать в подобных случаях? В спецификации этому посвящен целый раздел (http://www.w3.org/TR/html5/common-idioms.html#common-idioms). Лично я отдавал предпочтение устаревшему элементу <hgroup>, но, к сожалению, этот корабль уже уплыл (упоминание о нем можно найти в разделе «Устаревшие функции HTML»). Итак, чтобы последовать совету, данному в спецификации, предыдущий пример должен быть переписан следующим образом:

```
<h1>Scones:</h1>
<p>The most resplendent of snacks</p>
```

Семантика HTML5 на текстовом уровне

Кроме уже рассмотренных элементов, определяющих структуру и помогающих сгруппировать данные, в HTML5 предусматривается ряд тегов, которые ранее назывались линейными элементами. Теперь в спецификации HTML5 на эти элементы ссылаются как на элементы семантики на текстовом уровне (http://www.w3.org/TR/html5/text-level-semantics.html#text-level-semantics). Рассмотрим несколько примеров общего характера.

Элемент <b>

Исторически сложилось так, что элемент <b> означает «выделить жирным шрифтом» (http://www.w3.org/TR/html4/present/graphics.html#edef-B). Это было еще в те давние времена, когда частью разметки считался и выбор стиля. Но теперь официально этот элемент можно использовать только в качестве стилевой привязки в коде CSS, поскольку в текущей спецификации HTML5 об элементе <b> говорится следующее:

> «Элемент <b> является представлением фрагмента текста, на который обращается внимание в потребительском смысле, без придания ему какой-либо особой важности и без применения другой голосовой интонации; это могут быть, к примеру, выделяемые в документе ключевые слова, наименования товаров в обзоре, побуждающие к действию слова в интерактивном, управляемом текстом программном средстве или же первый абзац статьи».

Хотя сейчас этому элементу не придается никакого конкретного смысла, поскольку он относится к текстовому уровню, он не предназначен для заключения в него большой группы элементов разметки, поскольку для этой цели следует применять div-элемент. Следует также иметь в виду, что поскольку он исторически использовался для выделения текста жирным шрифтом, то, если вам нужно, чтобы содержимое, помещенное внутрь тега <b>, не выводилось на экран жирным шрифтом, придется переключить свойство font-weight в CSS.

Элемент

Каюсь, я часто пользовался элементом в качестве средства стилевого оформления. Мне следует пересмотреть свои позиции, приняв во внимание определение из HTML5:

> «Элемент акцентирует внимание на своем содержимом».

Следовательно, пока вы не хотите акцентировать чье-либо внимание на заключаемом в элемент содержимом, пользуйтесь тегом <b> или там, где это уместно, тегом <i>.

Элемент <i>

В спецификации HTML5 про элемент <i> говорится следующее:

> «...выделяемый или произносимый с другой интонацией отрывок текста или же выделенный качественно — как выпадающий из общего ряда повествования»

Достаточно сказать, что он не используется просто для выделения чего-либо курсивным шрифтом. Например, им можно воспользоваться для разметки в строке текста какого-либо странного названия:

```
<p>However, discussion on the hgroup element is now frustraneous as
it's now gone the way of the <i>Raphus cucullatus</i>.</p>
```

ПРИМЕЧАНИЕ

В HTML5 существует масса других семантических тегов текстового уровня. Чтобы просмотреть все эти теги, обратитесь к соответствующему разделу спецификации по адресу http://www.w3.org/TR/html5/text-level-semantics.html#text-level-semantics.

Устаревшие функции HTML

Кроме атрибутов языка в ссылках на сценарии, есть и другие части HTML, которыми вы, возможно, привыкли пользоваться и которые теперь в HTML5 считаются устаревшими. Важно понимать, что в HTML5 есть две разновидности устаревших функций: соответствующие и несоответствующие. Соответствующие функции сохраняют свою работоспособность, но в системах проверки в отношении их будут выдаваться предупреждения. На практике следует по возможности избегать их применения, но если все же они будут использоваться, от этого небеса не рухнут на землю. Несоответствующие функции могут по-прежнему влиять на отображение данных в конкретных браузерах, но если их использовать, то можно прослыть весьма рисковым человеком и лишиться выходных!

Перечень устаревших и несоответствующих функций весьма обширен. Признаться, многими из них я никогда не пользовался (а некоторые даже никогда не видел!). Возможно, у вас на них будет аналогичная реакция. Но если интерес-

но, полный перечень устаревших и несоответствующих функций можно найти по адресу http://www.w3.org/TR/html5/obsolete.html. Из наиболее примечательных устаревших и несоответствующих функций можно назвать `strike`, `center`, `font`, `acronym`, `frame` и `frameset`.

Есть также функции, фигурировавшие в ранних предварительных версиях HTML5, которые теперь попали в число отвергнутых. В качестве примера можно привести `hgroup`. Изначально предполагалось использовать этот тег в качестве контейнера для групп заголовков, то есть `h1` как основной заголовок и `h2` как подзаголовок могли помещаться внутрь элемента `hgroup`. Но теперь дискутировать по поводу элемента `hgroup` совершенно бесполезно, поскольку он пошел по пути вымершего Raphus cucullatus (если хотите узнать, кто это, воспользуйтесь Google).

Практическое применение элементов HTML5

Настало время применить некоторые из только что рассмотренных элементов на практике. Вернемся к примеру из главы 1. Если сравнить разметку, показанную ниже, с исходной разметкой из первой главы (напоминаю, что все примеры можно загрузить с сайта http://rwd.education или из репозитория GitHub), то можно увидеть те места, где применены рассмотренные нами новые элементы:

```
<article>
    <header class="Header">
        <a href="/" class="LogoWrapper"><img src="img/SOC-Logo.png"
            alt="Scone O'Clock logo" /></a>
        <h1 class="Strap">Scones: the most resplendent of snacks</h1>
    </header>
    <section class="IntroWrapper">
        <p class="IntroText">Occasionally maligned and misunderstood; the
            scone is a quintessentially British classic.</p>
        <figure class="MoneyShot">
            <img class="MoneyShotImg" src="img/scones.jpg" alt="Incredible
                scones" />
            <figcaption class="ImageCaption">Incredible scones, picture from
                Wikipedia</figcaption>
        </figure>
    </section>
    <p>Recipe and serving suggestions follow.</p>
    <section class="Ingredients">
        <h3 class="SubHeader">Ingredients</h3>
    </section>
    <section class="HowToMake">
        <h3 class="SubHeader">Method</h3>
    </section>
    <footer>
        Made for the book, <a href="http://rwd.education">'Resonsive
            web design with HTML5 and CSS3'</a> by <address><a href="http://
            benfrain">Ben Frain</a></address>
    </footer>
</article>
```

Осмысленный выбор элементов. Чтобы сконцентрироваться на структуре, я убрал существенный объем внутреннего содержимого. Надеюсь, вы согласитесь с тем, что отличить различные разделы разметки друг от друга весьма несложно. Но теперь мне хотелось бы дать один довольно прагматичный совет: мир не рухнет, если вы не станете всякий раз выбирать правильный элемент для каждой отдельно взятой ситуации. Например, какой бы из элементов в предыдущем примере я ни применил, `<section>` или `<div>`, это не выразилось бы в каких-то реальных последствиях. Если вместо предписанного к использованию элемента `<i>` используется `<em>`, я не считаю это преступлением против человечества и ребята из W3C не откроют на вас охоту и не станут вас очернять за неправильный выбор. Просто руководствуйтесь здравым смыслом. И тем не менее, если есть возможность смыслового использования таких элементов, как `<header>` и `<footer>`, их применение обеспечит бóльшую доступность вашего кода для понимания.

Использование WCAG и WAI-ARIA для повышения доступности веб-приложений

Со времени написания первого издания данной книги (2011 и 2012 годы) консорциум W3C добился многого в деле упрощения для разработчиков написания более доступных веб-страниц.

Руководство по обеспечению доступности веб-контента (WCAG)

Руководство WCAG существует с целью предоставления *«единого общего стандарта для доступности веб-контента, отвечающего на международной основе нуждам физических лиц, организаций и правительств»*.

Когда речь заходит о более простых веб-страницах (в противовес единой странице веб-приложений и им подобных средств), имеет смысл сконцентрироваться на руководстве WCAG. В нем предлагается целый ряд (в основном вполне разумных) рекомендаций по обеспечению доступности вашего веб-контента. Каждая рекомендация оценивается по уровням соответствия: A, AA или AAA. Дополнительные сведения об этих уровнях можно найти по адресу http://www.w3.org/TR/UNDERSTANDING-WCAG20/conformance.html#uc-levels-head.

Возможно, окажется, что вы уже придерживаетесь многих из рекомендаций, например, в плане предоставления альтернативного текста для изображений. Краткое изложение рекомендаций можно получить по адресу http://www.w3.org/WAI/WCAG20/glance/Overview.html, после чего создать свой собственный краткий контрольный лист-справочник по образцу, показанному на сайте http://www.w3.org/WAI/WCAG20/quickref/.

Я призываю всех потратить пару часов на изучение списка. Многие рекомендации совсем не трудно выполнить, предоставив тем самым реальные преимущества пользователям.

Стандарт предоставления возможности полноценного использования Интернета людьми с физическими ограничениями (WAI-ARIA)

Стандарт WAI-ARIA предназначен главным образом для решения проблем доступности динамического контента на веб-странице. Он предоставляет для специализированных виджетов (динамических разделов в веб-приложениях) средства описания ролей, состояний и свойств, чтобы они стали узнаваемыми и годными к использованию теми пользователями, которые применяют вспомогательные технологии.

Например, если выведенный на экран виджет показывает постоянно обновляемые биржевые котировки, то как незрячий пользователь получит доступ к странице, чтобы узнать об этом? Стандарт WAI-ARIA пытается решать проблемы подобного рода.

Не используйте роли для семантических элементов. Ранее рекомендовалось добавлять значимые роли к заголовкам и подвалам:

```
<header role="banner">A header with ARIA landmark banner role</header>
```

Но теперь это считается излишним. Если заглянуть в спецификацию для любого из ранее перечисленных элементов, там есть специально выделенный раздел разрешенных значений ARIA-атрибута `role` — Allowed ARIA role attributes. В качестве примера можно привести разъяснения из раздела элемента:

> «Разрешенные значения ARIA-атрибута role: region role (по умолчанию не устанавливается), alert, alertdialog, application, contentinfo, dialog, document, log, main, marquee, presentation, search или status».

Основной частью здесь является «role (по умолчанию не устанавливается)». Это означает, что явное добавление этой ARIA-роли к элементу не имеет никакого смысла, поскольку она подразумевается самим элементом. Теперь это поясняется примечанием к спецификации:

> «В большинстве случаев установка ARIA-атрибута role и/или атрибута aria-*, соответствующего исходной подразумеваемой ARIA-семантике, является излишней и не рекомендуется, поскольку эти свойства уже установлены браузером».

Если вы не в состоянии запомнить более одной рекомендации

Самое простое, что можно сделать с прицелом на применение вспомогательных технологий, — это пользоваться везде, где это возможно, правильными элементами. Элемент `<header>` окажется намного полезнее, чем `div class="Header"`. Точно так же, если у вас на странице есть кнопка, воспользуйтесь элементом `<button>`, а не элементом `<span>` или другим элементом, стилизованным под кнопку. Готов признать, что элемент `<button>` не всегда позволяет применять конкретное стилевое оформление (его, к примеру, нельзя настроить с помощью инструкций `display: table-cell` или `display: flex`), но тогда выберите следующий более подходящий элемент, в качестве которого обычно выступает тег `<a>`.

Развитие стандарта ARIA

ARIA не ограничивается одной лишь ориентацией в отношении ролей. Развивать стандарт помогают полный перечень ролей и краткое описание их пригодности к использованию, доступные по адресу http://www.w3.org/TR/wai-aria/roles.

Чтобы проще было разобраться в теме, я также рекомендую прочитать книгу Хейдона Пикеринга (Heydon Pickering) *Apps For All: Coding Accessible Web Applications* (доступна на сайте https://shop.smashingmagazine.com/products/apps-for-all-coding-accessible-web-applications).

СОВЕТ

Можно бесплатно протестировать свою конструкцию с помощью невизуального доступа к Рабочему столу (non-visual desktop access (NVDA)).

Если разработка ведется под платформу Windows и возникает желание протестировать конструкции, доработанные согласно стандарту ARIA с помощью экранного диктора, можно совершенно свободно воспользоваться средством NVDA. Оно доступно по адресу http://www.nvda-project.org/.

Google теперь также поставляет бесплатное инструментальное средство для разработки доступных сайтов под названием Accessibility Developer Tools, предназначенное для браузера Chrome (доступны также кросс-платформенные версии), которое также стоит проверить в деле.

Существует также постоянно пополняемый арсенал средств, помогающих быстро протестировать ваши собственные конструкции на использование их людьми с нарушенным цветовосприятием. Например, по адресу https://michelf.ca/projects/sim-daltonism/ находится Mac-приложение, позволяющее переключать типы нарушения цветовосприятия и просматривать предварительные результаты на плавающей палитре.

И наконец, OS X также включает утилиту VoiceOver, позволяющую тестировать ваши веб-страницы.

Надеюсь, краткое введение в WAI-ARIA и WCAG обеспечило вам достаточный объем информации, чтобы появилось больше идей о том, как можно подойти к поддержке вспомогательных технологий. Возможно, добавление поддержки таких технологий к вашим следующим проектам на HTML5 окажется проще, чем вы думали.

И в заключение укажу на еще один ресурс, касающийся всех вопросов доступности, содержащий массу полезных ссылок и советов, находящихся на домашней странице проекта A11Y по адресу http://a11yproject.com/.

Медиавозможности, встроенные в HTML5

Многие услышали об HTML5, когда Apple отказалась добавлять поддержку для Flash на свои iOS-устройства. Технология Flash получила на рынке доминирующее положение (некоторые утверждают, что она захватила рынок) в качестве дополнительного модуля для обслуживания видео в браузере. Но вместо использования технологии, являющейся собственностью компании Adobe, Apple в вопросах вывода на экран сложного медиасодержимого решила положиться на HTML5.

Наряду с тем что в HTML5 был достигнут неплохой прогресс в данной области, публичная поддержка HTML5 со стороны Apple дала этой версии языка могучий толчок и помогла ее медиаинструментам стать привлекательными для широких кругов потребителей.

Несложно предположить, что Internet Explorer 8 и более ранние версии этого браузера не поддерживают аудио и видео HTML5. Большинство других современных браузеров (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) справляются с этой задачей вполне успешно.

Добавление видео и аудио средствами HTML5

Работать с видео и аудио в HTML5 довольно просто. Единственной реальной трудностью, связанной с HTML5-медиа, было перечисление альтернативных медиаформатов, поскольку различные браузеры поддерживают разные форматы файлов. Сегодня повсеместной поддержкой как на обычных, так и на мобильных платформах пользуется формат MP4, что делает включение медиаинформации в ваши веб-страницы посредством HTML5 простым и легким. Вот пример ссылки на видеофайл на вашей странице из разряда «проще некуда»:

```
<video src="myVideo.mp4"></video>
```

В HTML5 со всей трудной работой справляется один-единственный тег `<video></video>` (или `<audio></audio>` для аудио). Между открывающим и закрывающим тегами можно вставить текст, информирующий пользователей при возникновении проблем. Существуют также дополнительные атрибуты, которые обычно хочется добавить, например, высота и ширина. Добавим их:

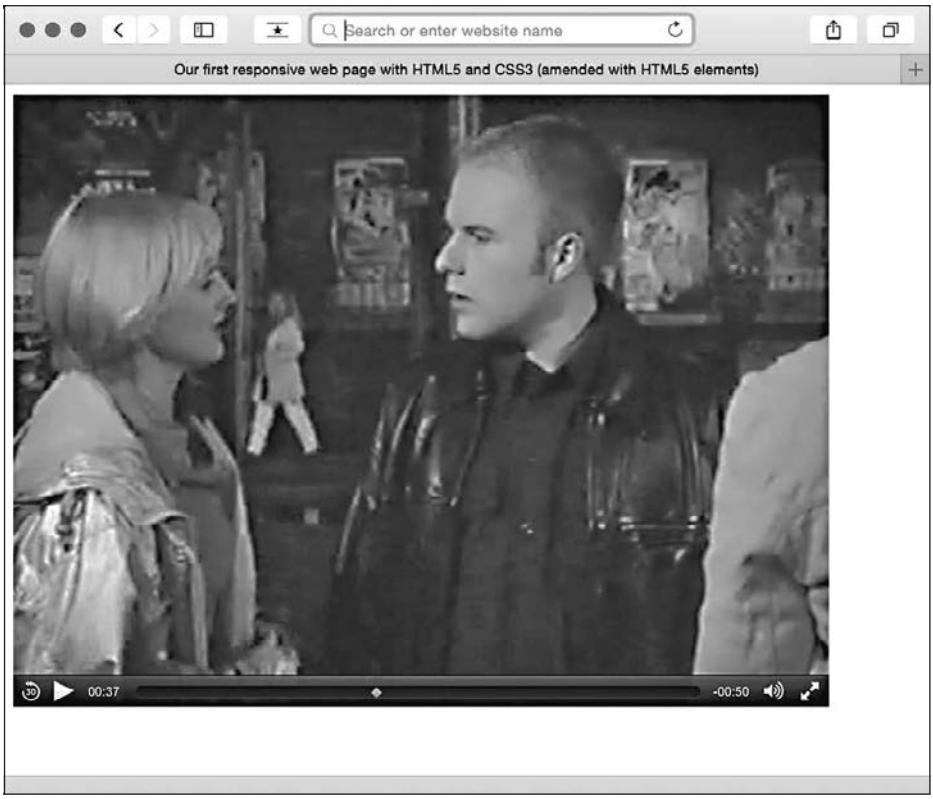
```
<video src="myVideo.mp4" width="640" height="480">What, do you mean
    you don't understand HTML5?</video>
```

Теперь, если добавить предыдущий фрагмент кода на страницу и посмотреть на его работу в Safari, видео появится, но без элементов управления его проигрыванием. Для получения исходных элементов управления проигрыванием нужно добавить атрибут `controls`. Мы также можем добавить атрибут `autoplay` (что не рекомендуется делать, поскольку всем известно, с каким раздражением воспринимаются видео с автовоспроизведением). Все это демонстрируется в следующем фрагменте кода:

```
<video src="myVideo.mp4" width="640" height="480" controls autoplay>
      What, do you   mean you don't understand HTML5?</video>
```

Результат выполнения этого фрагмента показан на следующей копии экрана.

В число прочих атрибутов входят `preload` для управления предварительной загрузкой медиа (те, кто пользовался ранними версиями HTML5, заметят, что `preload` пришел на смену `autobuffer`), `loop` для повторения видео и `poster` для определения кадра заставки для видео. Он пригодится при ожидаемой задержке воспроизведения видео (или когда какое-то время будет затрачиваться на буферизацию). Для применения атрибута достаточно вставить его в тег. Вот как выглядит пример, включающий все эти атрибуты:




```
<video src="myVideo.mp4" width="640" height="480" controls autoplay
preload="auto" loop poster="myVideoPoster.png">What, do you mean you
don't understand HTML5?</video>
```

Резервная возможность для старых браузеров. Тег `<source>` позволяет при необходимости предоставлять резервные возможности. Например, наряду с предоставлением версии видео в формате MP4 при желании обеспечить удобную замену для Internet Explorer 8 и более ранних версий можно добавить резервное видео в формате Flash. Кроме того, если у пользователя в браузере нет подходящей технологии проигрывания, можно предоставить ссылки на скачивание самих файлов. Рассмотрим следующий пример:

```
<video width="640" height="480" controls preload="auto" loop
               poster="myVideoPoster.png">
    <source src="video/myVideo.mp4" type="video/mp4">
    <object width="640" height="480" type="application/x-shockwave-
               flash" data="myFlashVideo.SWF">
        <param name="movie" value="myFlashVideo.swf" />
        <param name="flashvars" value="controlbar=over&amp;image=myVideo
               Poster.jpg&amp;file=myVideo.mp4" />
        <img src="myVideoPoster.png" width="640" height="480"
               alt="__TITLE__"
```

```
            title="No video playback capabilities, please download the
               video below" />
    </object>
    <p><b>Download Video:</b>
        MP4 Format: <a href="myVideo.mp4">"MP4"</a>
    </p>
</video>
```

Этот пример кода находится в файле example2.html, который, как и пробный видеофайл в формате MP4, содержится в папке с кодом к данной главе.

Работа audio и video практически ничем не различается

Тег `<audio>` работает по таким же принципам и с такими же атрибутами (исключая `width`, `height` и `poster`). Основное отличие этих двух тегов друг от друга заключается в том обстоятельстве, что у `<audio>` отсутствует область проигрывания визуального содержимого.

Адаптивное HTML5-видео и iFrames

Мы уже видели, что поддержка устаревших браузеров неизменно приводит к увеличению объема кода. То, что начиналось с одной или двух строк тега `<video>`, заканчивается в конечном итоге десятью и более строками (и дополнительным Flash-файлом), и все это ради того, чтобы осчастливить пользователей устаревших версий Internet Explorer! Что касается меня, то я обычно радуюсь возможности отказаться от отката к Flash-технологии в погоне за меньшим объемом используемого кода, но все же бывают разные обстоятельства.

Теперь единственной проблемой со столь понравившейся нам реализацией видео в HTML5 является отсутствие в ней адаптивности. И действительно, в книге приведен пример, который не реагирует на изменение условий просмотра.

К счастью, для встроенного HTML5-видео все это можно легко исправить. Просто уберите из разметки все атрибуты высоты и ширины (например, удалите `width="640" height="480"`) и добавьте в CSS следующий код:

```
video { max-width: 100%; height: auto; }
```

Но, вполне успешно работая с файлами, которые могут храниться на локальном устройстве, этот прием не решает проблемы видео, встроенного в `iFrame` (низкий поклон YouTube, Vimeo и другим сайтам). Следующий код позволяет добавить трейлер фильма «Успеть до полуночи» из YouTube:

```
<iframe width="960" height="720" src="https://www.youtube.com/
        watch?v=B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>
```

Но если добавлять этот код к странице в неизменном виде, то даже при добавлении предшествующего CSS-правила, если ширина окна просмотра будет менее 960 пикселов, произойдет обрезка.

Проще всего решить эту проблему с помощью небольшого CSS-трюка, впервые примененного французским CSS-специалистом Тьерри Кобленцем (Thierry Koblentz), который создал, по сути, блок с правильным соотношением сторон для содержащегося в нем видео. Я не стану пересказывать собственные объяснения этого мага, которые можно найти по адресу http://alistapart.com/article/creating-intrinsic-ratios-for-video.

Если лень, вам даже не нужно самим вычислять и подставлять соотношение сторон, поскольку существует онлайн-сервис, способный сделать это за вас. Просто обратитесь по адресу http://embedresponsively.com/ и поместите в адресную строку этого сайта URL-адрес вашего `iFrame`. Сайт выдаст вам простой фрагмент кода, который можно вставить в свою страницу. К примеру, для трейлера фильма «Успеть до полуночи» будет получен следующий результат:

```
<style>.embed-container { position: relative; padding-bottom: 56.25%;
height: 0; overflow: hidden; max-width: 100%; height: auto; } .embed-
container iframe, .embed-container object, .embed-container embed {
position: absolute; top: 0; left: 0; width: 100%; height: 100%; }</
style><div class='embed-container'><iframe src='http://www.youtube.
com/embed/B1_N28DA3gY' frameborder='0' allowfullscreen></iframe></div>
```

Вот и все, просто добавьте этот фрагмент к своей странице и получите полностью адаптируемое YouTube-видео (примечание: дети, не берите пример с мистера Де Ниро, курить вредно!).

Замечание относительно приоритетности автономной работы

Я уверен, что идеальный способ создания адаптивных веб-страниц и веб-приложений основан на приоритетности автономной работы. Этот подход означает, что сайты и приложения будут продолжать работать и загружаться даже без интернет-соединения. Для достижения этой цели предназначены автономные веб-приложения на HTML5 (http://www.w3.org/TR/2011/WD-html5-20110525/offline.html).

Хотя такие приложения имеют довольно широкую поддержку (http://caniuse.com/#feat=offline-apps), к сожалению, этому решению еще далеко до идеала. При всей относительной простоте его настройки существует целый ряд ограничений и подводных камней. Описание всех препятствий не входит в задачи данной книги. Поэтому я рекомендую прочитать написанную с юмором и довольно подробную статью на эту тему Джейка Арчибальда (Jake Archibald), которую можно найти по адресу http://alistapart.com/article/application-cache-is-a-douchebag.

Я же по этому поводу придерживаюсь мнения, что, несмотря на возможность получения положительного опыта приоритетности автономной работы с помощью автономных веб-приложений (неплохое руководство по способам достижения этой цели можно найти по адресу http://diveintohtml5.info/offline.html) и LocalStorage (или какой-нибудь комбинации этих двух средств), до наилучшего решения нам еще слишком далеко. Я возлагаю свои надежды на Service Workers (http://www.w3.org/TR/service-workers/).

На момент написания данной книги Service Workers все еще пребывала в статусе относительно новой спецификации, но в качестве неплохого обзора, я предлагаю посмотреть следующее 15-минутное введение: https://www.youtube.com/watch?v=4uQMl7mFB6g. Прочитайте еще это введение: http://www.html5rocks.com/en/tutorials/service-worker/introduction/ — и проверьте возможность поддержки на сайте https://jakearchibald.github.io/isserviceworkerready/.

Надеюсь, к тому времени, когда я займусь подготовкой третьего издания этой книги (если, конечно, возьмусь за него), появится возможность дать полный обзор и описание реализации этой технологии. Скрестим пальцы на удачу.

Резюме

В данной главе был рассмотрен довольно объемный материал. Все, начиная с основ создания страницы, проходящей проверку на соответствие стандартам HTML5, и заканчивая встраиваемым в вашу разметку сложным медиасодержимым (видео) и вопросами обеспечения его адаптивного поведения.

Хотя это и не имеет отношения к адаптивному дизайну, мы разобрались также в способах написания семантически обогащенного и осмысленного кода и рассмотрели вопрос смыслового наполнения страниц и обеспечения их пригодности для тех пользователей, которые зависят от применения вспомогательных технологий.

Текущая глава была слишком перегружена разметкой, поэтому теперь поменяем задачу. В следующих двух главах мы собираемся постичь всю мощь и гибкость CSS. Сначала рассмотрим эффективность применения селекторов CSS уровня 3 и 4, новые единицы измерения CSS, относящиеся к окнам просмотра, и такие возможности, как использование функции `calc` и HSL-цвета. Все это позволит нам создавать более быстрые, имеющие более впечатляющие возможности и легче сопровождаемые адаптивные конструкции.

5 CSS3. Селекторы, разметка, цветовые режимы и новые возможности

За последние несколько лет в CSS-технологию введено множество новых функций. Одни позволяют анимировать и преобразовать элементы, другие дают возможность создавать фоновые изображения, эффекты градиентов, масок и фильтров, а третьи позволяют оживлять SVG-элементы.

Все эти возможности будут рассматриваться в следующих нескольких главах. Я думаю, что в первую очередь было бы полезно изучить ряд основ CSS, претерпевших изменения за несколько последних лет: порядок выбора элементов на странице, единицы измерения, доступные для изменения стиля и размеров наших элементов, и способы, благодаря которым существующие (и будущие) псевдоклассы и псевдоэлементы повышают эффективность использования CSS-таблиц. Кроме того, мы рассмотрим способ разветвления в коде CSS, позволяющий задействовать возможности, поддерживаемые в различных браузерах.

В этой главе будут рассмотрены следующие вопросы:

- анатомия правил CSS (что определяет правило, объявления и пары «свойство — значение»);
- простые и полезные CSS-трюки для адаптивных конструкций (использование нескольких столбцов, перенос слов на новые строки, усечение текста с использованием многоточия, создание областей прокрутки);
- предоставление возможности разветвления функций в CSS (способы применения одних правил в одних браузерах, а других правил — в других браузерах);
- способы применения селекторов по соответствующим подстрокам значений атрибутов с целью выбора HTML-элементов;
- что такое селекторы на основе порядковых номеров и как ими пользоваться;
- что такое псевдоклассы и псевдоэлементы (`:empty`, `::before`, `::after`, `:target`, `:scope`);
- новые селекторы в модуле CSS Level 4 Selectors (`:has`);
- что такое переменные CSS и пользовательские свойства и как они записываются;

- что такое CSS-функция `calc` и как ею можно воспользоваться;
- использование единиц измерения, относящихся к окнам просмотра (`vh`, `vw`, `vmin` и `vmax`);
- как пользоваться шрифтовым веб-оформлением с помощью `@font-face`;
- режимы цветовых настроек RGB и HSL с альфа-прозрачностью.

Разве можно знать абсолютно все?

Абсолютными знаниями не обладает никто. Я работаю с CSS на протяжении 10 лет и каждую неделю открываю для себя что-то новое (или хорошо забытое старое). Поэтому я не думаю, что стоит гнаться за усвоением каждого возможного сочетания свойств и значений CSS. Разумнее выработать общее представление об имеющихся возможностях.

Исходя из этого, в данной главе я собираюсь сосредоточиться на технологиях, единицах измерения и селекторах, которые, на мой взгляд, наиболее полезны при создании адаптивных веб-конструкций. Надеюсь, после изучения главы вы получите тот объем знаний, который будет необходим для решения большинства проблем, встречающихся на пути разработки адаптивного веб-дизайна.

Анатомия правила CSS

Перед исследованием всего арсенала средств, предлагаемого CSS3, и во избежание путаницы установим терминологию, которой будем придерживаться для описания правила CSS. Рассмотрим следующий пример:

```
.round { /* селектор */
    border-radius: 10px; /* объявление */
}
```

Это правило состоит из селектора (`.round`), после которого следует объявление (`border-radius: 10px;`). В свою очередь, объявление определяется свойством (`border-radius:`) и значением (`10px;`). Наши мнения совпали? Отлично, теперь давайте ускоримся.

НЕ ЗАБУДЬТЕ ПРОВЕРИТЬ НАЛИЧИЕ ПОДДЕРЖКИ ДЛЯ ВАШИХ ПОЛЬЗОВАТЕЛЕЙ

По мере углубления в CSS3 не забудьте посещать сайт http://caniuse.com/, если, конечно, вам интересно, каков текущий уровень браузерной поддержки конкретной возможности CSS3 или HTML5. Кроме отображения сведений о поддержке различными версиями браузеров (которые можно найти в отношении конкретной возможности), на сайте также приведены текущие статистические данные об уровнях использования браузеров, взятые с сайта http://gs.statcounter.com/.

Простые и полезные трюки CSS

Я пришел к выводу, что в своей повседневной работе постоянно использую одни возможности CSS3 и весьма редко применяю другие. Я подумал, что было бы

полезно поделиться знаниями о тех возможностях, которые использую наиболее часто. В CSS3 имеется множество полезных вещей, способных облегчить жизнь, особенно тем, кто занимается адаптивным веб-дизайном. С их помощью довольно легко решаются проблемы, которые раньше вызывали головную боль.

Использование CSS при создании многоколоночных разметок для адаптивных конструкций

Вам когда-нибудь хотелось, чтобы текстовый блок появлялся в нескольких колонках? Эту задачу можно решить, разложив содержимое по разным элементам разметки и задав им соответствующие стилевые настройки. Но внесение изменений в разметку ради достижения определенного стилевого оформления — далеко не идеальное решение. Способ совсем не сложного разнесения одного или нескольких фрагментов содержимого по нескольким колонкам описывается в CSS-спецификации многоколоночной разметки. Рассмотрим следующую разметку:

```
<main>
    <p>lloremipsimLoremipsum dolor sit amet, consectetur
<!-- БОЛЬШОЙ ОБЪЕМ ТЕКСТА -->
</p>
    <p>lloremipsimLoremipsum dolor sit amet, consectetur
<!-- БОЛЬШОЙ ОБЪЕМ ТЕКСТА -->
</p>
</main>
```

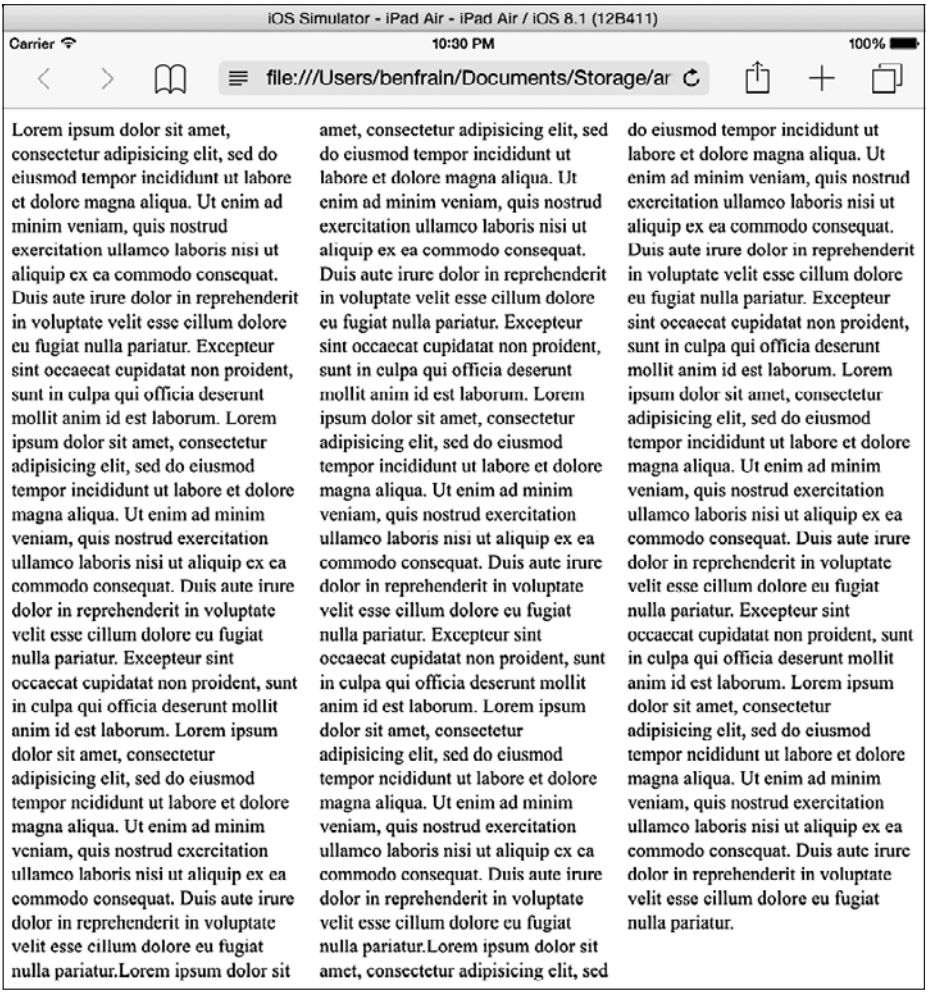
Используя CSS-свойство, позволяющее выводить содержимое в нескольких колонках, можно применить несколько способов распределения содержимого по нескольким колонкам. Можно сделать колонки определенной ширины (например, 12 em) или указать, что содержимое должно распределяться по конкретному количеству колонок (например, по трем).

Посмотрим на код, необходимый для выполнения каждого из этих сценариев. Для колонок заданной ширины используется следующий синтаксис:

```
main {
    column-width: 12em;
}
```

Это будет означать, что независимо от ширины окна просмотра содержимое будет распределяться по колонкам шириной 12 em. При изменении размеров окна просмотра количество выводимых на экран колонок будет динамически изменяться. Можно посмотреть все это в браузере, загрузив в него пример из файла каталога example_05-01 (или из репозитория GitHub — https://github.com/benfrain/rwd).

Посмотрим, как эта страница выводится на экран iPad в портретной ориентации (при ширине окна просмотра 768 пикселов).



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

А теперь посмотрим на нее в браузере Chrome на экране настольного компьютера (с шириной окна просмотра около 1100 пикселов).

Вот вам и простые адаптируемые текстовые колонки, полученные при минимуме усилий. Лично мне это нравится!

Фиксированное количество колонок при изменяемой ширине

Если нужно получить неизменное количество колонок при изменяемой их ширине, можно написать следующее правило:

```
main {
    column-count: 4;
}
```

Добавление промежутков и разделителей колонок

Можно пойти еще дальше и добавить определенный промежуток между колонками, а также разделитель колонок:

```
main {
    column-gap: 2em;
    column-rule: thin dotted #999;
    column-width: 12em;
}
```

При этом будет получен следующий результат.

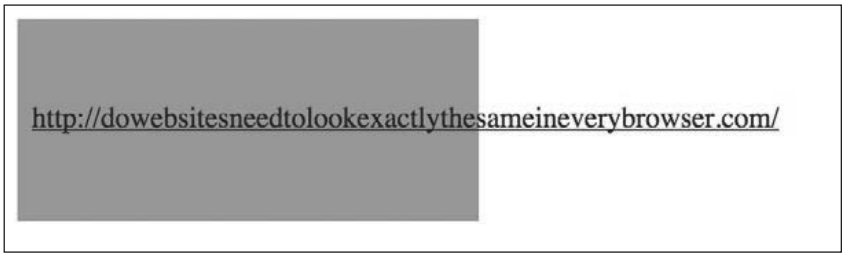
Спецификацию, касающуюся модуля CSS3 Multi-Column Layout Module, можно найти по адресу http://www.w3.org/TR/css3-multicol/.

На данный момент, несмотря на объявленный W3C статус CR, при объявлении колонок вам для максимальной совместимости, скорее всего, понадобится указать префиксы производителей.

Единственное предостережение, которое мне хотелось бы дать в отношении применения свойств CSS, задающих использование нескольких колонок, касается того, что при растянутом по колонкам тексте впечатление пользователей от страницы может быть подпорчено, поскольку, возможно, для чтения колонок им придется прокручивать страницу вверх-вниз, а это может стать весьма утомительным занятием.

Перенос слов на новые строки

Сколько раз вы впадали в уныние, когда приходилось вводить длинный URL-адрес, имея для этого весьма узкое поле ввода? Посмотрите на пример, находящийся по адресу rwd.education/code/example_05-04. Проблему можно разглядеть и в следующей копии экрана, заметив, что URL-адрес выходит за пределы выделенного ему пространства.



Данная проблема решается довольно легко путем простого CSS3-объявления, которое, как нарочно, работает точно так же и в устаревших версиях Internet Explorer вплоть до версии 5.5! Просто добавьте:

```
word-wrap: break-word;
```

к элементу-контейнеру — и получите эффект, показанный в следующей копии экрана.

Вуаля, теперь длинные URL-адреса отлично переносятся на новую строку!

Усечение текста с добавлением многоточия

По укоренившейся традиции усечение текста всегда было прерогативой технологий на серверной стороне. Но сегодня можно произвести усечение текста с добавлением многоточия с помощью только лишь кода CSS. Посмотрим, как это делается.

Рассмотрим следующую разметку (в Интернете этот пример можно найти по адресу rwd.education/code/ch5/example_05-03/):

```
<p class="truncate">OK, listen up, I've figured out the key eternal
happiness. All you need to do is eat lots of scones.</p>
```

А нам нужно произвести усечение текста до ширины 520 пикселов, чтобы он имел следующий вид.

OK, listen up, I've figured out the key eternal happiness. All you need to do is …

Этого эффекта можно добиться с помощью следующего кода CSS:

```
.truncate {
    width: 520px;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: no-wrap;
}
```

СОВЕТ

Спецификацию, касающуюся свойства text-overflow, можно найти по адресу http://dev.w3.org/csswg/css-ui-3/.

Когда ширина содержимого превышает ширину, заданную в определении (которую можно также задать как 100 %, если содержимое находится в подстраиваемом под экран контейнере), оно будет усечено. Пара «свойство — значение» `white-space: no-wrap` используется для обеспечения отключения переноса внутри элемента-контейнера.

Создание панелей, прокручивающихся по горизонтали

Надеюсь, вы понимаете, о чем речь? Панели с прокруткой по горизонтали широко представлены в магазине iTunes и в Apple TV и используются как демонстрационные панели соответствующего содержимого (фильмы, альбомы и т. д.). Когда горизонтального пространства вполне хватает, в поле зрения попадают все объекты. Когда пространство ограничено (например, на мобильных устройствах), панель можно прокручивать из стороны в сторону.

Прокручиваемые панели особенно хорошо работают на современных устройствах под управлением Android и iOS. Если у вас есть современное iOS- или Android-устройство, посмотрите следующий пример на нем, а также в браузере настольного компьютера типа Safari или Chrome: rwd.education/code/ch5/example_05-02/.

Я создал панель для лучших фильмов 2014 года, которая имеет на iPhone следующий вид (см. стр. 121).

Сильно стараться мне не пришлось. Ключевым в данной технологии является свойство `white-space`, которое появилось еще в версии CSS 2.1 (http://www.w3.org/TR/CSS2/text.html). Но я собираюсь использовать его совместно с новым механизмом разметки Flexbox, ведь вы же не станете мне возражать?

Чтобы проявился эффект, положенный в основу данной технологии, нам нужно иметь окружающее пространство шириной меньше суммарной ширины содержащейся информации, а также настроить ширину по оси *X* на автоматически выбираемое значение. Тогда прокрутка будет работать только при дефиците пространства:

```
.Scroll_Wrapper {
    width: 100%;
    white-space: nowrap;
    overflow-x: auto;
    overflow-y: hidden;
}

.Item {
    display: inline-flex;
}
```

Используя объявление `white-space: nowrap`, мы предписываем браузеру не переносить эти элементы на новую строку при обнаружении пробельного символа. Затем, чтобы все оставалось в одной строке, для всех первых дочерних элементов этого контейнера задаем линейное отображение. Здесь мы воспользовались значением `inline-flex`, но точно так же могли бы применить `inline`, `inline-block` или `inline-table`.

ПСЕВДОЭЛЕМЕНТЫ ::BEFORE И ::AFTER

Если посмотреть на код, выложенный в качестве образца, можно увидеть, что для отображения номера рекламного плаката применяется псевдоэлемент ::before. При использовании псевдоэлементов следует помнить, что для отображения ::before или ::after у них должно иметься значение содержимого, даже если это просто пробел. Когда эти псевдоэлементы отображаются на экране, они начинают вести себя соответственно как первый и последний дочерние элементы того элемента, в отношении которого применяются.

Чтобы все выглядело немного эстетичнее, я собираюсь скрыть полосу прокрутки там, где это возможно сделать. К сожалению, в силу специфики браузеров код придется дополнять вручную (Autoprefixer с этим не справится, поскольку на эти свойства распространяются права собственности). Я также собираюсь добавить инерционность прокрутки, необходимую для сенсорных устройств с браузерами на движке WebKit (обычно это устройства под управлением iOS). Теперь обновленное правило `.Scroll_Wrapper` будет иметь следующий вид:

```
.Scroll_Wrapper {
    width: 100%;
    white-space: nowrap;
    overflow-x: auto;
    overflow-y: hidden;
    /* Придание инерционности прокрутке на
    сенсорных устройствах на базе WebKit */
    -webkit-overflow-scrolling: touch;
    /* Удаление полос прокрутки в поддерживающих
    это свойство версиях IE */
    -ms-overflow-style: none;
}

/* Предотвращение появления полосы прокрутки в браузерах с движком WebKit */
.Scroll_Wrapper::-webkit-scrollbar {
    display: none;
}
```

При дефиците пространства мы получаем симпатичную горизонтальную панель без полосы прокрутки. В противном случае на экране помещается сразу все содержимое.

Но в отношении данной модели нужно сделать два предостережения. Во-первых, на момент написания книги в браузере Firefox не было свойства, позволяющего скрывать полосы прокрутки. Во-вторых, старые Android-устройства не способны выполнять горизонтальную прокрутку (так оно и есть). Поэтому я, как правило, дополняю данную модель вспомогательным средством обнаружения возможностей. Как это работает, мы рассмотрим в следующем разделе.

Предоставление возможности разветвления функций в CSS

При создании адаптивных веб-конструкций и стремлении получить единую конструкцию, работающую повсеместно на любом устройстве, часто приходится сталкиваться с ситуациями, при которых те или иные функции или технологии не поддерживаются какими-либо устройствами. В таких случаях вам, скорее всего, захочется создать в коде CSS соответствующее разветвление: если браузер поддерживает функцию, предоставить один фрагмент кода, а если не поддерживает — другой. В JavaScript такие ситуации разрешаются с помощью инструкций `if-else` или `switch`.

На данный момент имеется два возможных подхода. Один из них полностью основан на применении CSS, но не очень широко поддерживается браузерами, а другой можно воплотить в жизнь только с помощью библиотеки JavaScript, но зато он пользуется значительно более широкой поддержкой браузеров. Рассмотрим их по очереди.

Запросы возможностей

Решением по разветвлению кода в CSS, присущим самой этой технологии, является использование запросов возможностей (feature queries), представляющих собой составную часть условных правил — CSS Conditional Rules Module Level 3 (http://www.w3.org/TR/css3-conditional/). Но на данный момент условные правила CSS не имеют поддержки в Internet Explorer (применительно к версии 11) и Safari (включая устройства под управлением iOS вплоть до версии 8.1), то есть они не пользуются достаточно широкой поддержкой.

Синтаксис у запросов возможностей такой же, как и у медиазапросов. Рассмотрим следующий код:

```
@supports (flashing-sausages: lincolnshire) {
    body {
        sausage-sound: sizzling;
        sausage-color: slighty-burnt;
        background-color: brown;
    }
}
```

В данном случае стили будут применены только в том случае, если браузер поддерживает свойство `flashing-sausages`. Я абсолютно уверен в том, что ни один браузер никогда не будет поддерживать свойство под названием `flashing-sausages`, поэтому ни один из стилей внутри блока `@supports` применяться не будет.

Рассмотрим более практичный пример. Зададим использование Flexbox в случае его поддержки браузером и предусмотрим резервную технологию разметки на случай, если Flexbox не поддерживается. Рассмотрим следующий пример:

```
@supports (display: flex) {
    .Item {
        display: inline-flex;
    }
}

@supports not (display: flex) {
    .Item {
        display: inline-block;
    }
}
```

Здесь один блок кода определяется на случай поддержки возможности, а другой блок — на случай отсутствия такой поддержки. Эта модель подойдет в том случае, если браузер поддерживает `@supports` (я понимаю, что в этом довольно

легко запутаться), но если он не обеспечивает такой поддержки, не будет применен ни один из этих стилей.

Если нужно обеспечить работу на устройствах, не поддерживающих `@supports`, лучше сначала написать исходное объявление и уже после него — то объявление, которое обусловлено запросом `@supports`, чтобы предыдущее правило отменялось в случае поддержки `@supports` и блок `@supports` игнорировался, если его поддержка отсутствует. Поэтому предыдущий пример требует следующей переработки:

```
.Item {
    display: inline-block;
}

@supports (display: flex) {
    .Item {
        display: inline-flex;
    }
}
```

Комбинирование условий

Условия также можно комбинировать. Предположим, нам нужно применить некоторые правила лишь при одновременной поддержке Flexbox и `pointer: coarse` (если вы пропустили соответствующий материал, то связанное с взаимодействием со страницей медиасвойство `pointer` рассматривалось в главе 2). Комбинированное условие может иметь следующий вид:

```
@supports ((display: flex) and (pointer: coarse)) {
    .Item {
        display: inline-flex;
    }
}
```

Здесь мы воспользовались ключевым словом `and`, но можно было использовать также `or` или заменить одно слово другим. Например, если стили нужно применить при условии поддержки двух предыдущих пар «свойство — значение» или при поддержке 3D-преобразований:

```
@supports ((display: flex) and (pointer: coarse)) or
                              (transform:translate3d(0, 0, 0)) {
    .Item {
        display: inline-flex;
    }
}
```

Обратите в предыдущем примере внимание на дополнительный набор круглых скобок, отделяющих условия для `flex` и `pointer` от условия для `transform`.

К сожалению, как уже говорилось, до широкой поддержки `@supports` пока еще слишком далеко. А что же тогда делать разработчику адаптивного веб-дизайна? Не впадать в уныние, ведь у нас есть великолепное JavaScript-средство, позволяющее легко преодолеть подобные трудности.

Modernizr

Пока `@supports` не получит более широкую реализацию в браузерах, мы можем воспользоваться JavaScript-средством под названием Modernizr. На данный момент это самый надежный способ реализации ветвлений в вашем коде.

Когда ветвление требуется в CSS, я стараюсь воспользоваться подходом, предполагающим постепенное усложнение, что означает применение сначала простого и доступного кода, который будет предоставлять на крайний случай функциональный дизайн для устройств с самыми скромными возможностями. Затем этот код постепенно усложняется для устройств с более развитыми возможностями.

СОВЕТ

Тема постепенного усложнения будет раскрыта в главе 10.

Посмотрим, как можно реализовать постепенное усложнение и ветвление нашего кода CSS с использованием Modernizr.

Обнаружение возможностей с помощью Modernizr. Если вы занимаетесь веб-разработкой, то, скорее всего, уже слышали о Modernizr, даже если и не пользовались этим средством. Оно представляет собой библиотеку JavaScript, включаемую в страницу и тестирующую браузер на наличие определенных возможностей. Чтобы приступить к использованию Modernizr, нужно просто включить ссылку на загруженный файл в раздел заголовка страницы:

```
<script src="/js/libs/modernizr-2.8.3-custom.min.js"></script>
```

Этот код при загрузке браузером страницы запускает всевозможные тесты из состава библиотеки. Если браузер проходит тест, Modernizr добавляет соответствующий класс к корневому тегу `HTML`, что вполне сообразуется с нашими целями.

Например, после того, как Mondernizr выполнит свои задачи, классы в теге `HTML` могут приобрести следующий вид:

```
<html class="js no-touch cssanimations csstransforms csstransforms3d
csstransitions svg inlinesvg" lang="en">
```

В данном случае тестированию подверглись только несколько возможностей: анимация, преобразования, применение SVG, линейной SVG и поддержка сенсорных экранов. При наличии этих классов код может подвергнуться ветвлению:

```
.widget {
    height: 1rem;
}

.touch .widget {
    height: 2rem;
}
```

В предыдущем примере элемент `widget` в обычных условиях получает высоту `1rem`, но если в `HTML` присутствует класс `touch` (благодаря работе Modernizr), то `widget` получит высоту `2rem`.

Можно также поменять логику на противоположную:

```
.widget {
    height: 2rem;
}

.no-touch .widget {
    height: 1rem;
}
```

Таким образом, исходной для элементов будет высота `2rem`, и она будет уменьшена, если присутствует класс `no-touch`.

Какая бы структура вам ни понадобилась, Modernizr предоставляет широкую поддержку ветвления при наличии возможностей. Особая польза от применения этого средства проявится при потребности в использовании таких возможностей, как `transform3d`, с сохранением предоставления работоспособной подмены для тех браузеров, которые не в состоянии справиться с такими возможностями.

СОВЕТ

Modernizr может предоставить точные тесты практически для всего, что вы можете пожелать использовать в качестве основы для ветвления кода, но есть и исключения. Например, общеизвестны трудности точного тестирования возможности overflow-scrolling. В тех ситуациях, когда класс устройств не может быть успешно определен, вероятно, будет разумнее выполнить ветвление кода на результате тестирования другой возможности. Например, старые версии Android испытывают трудности с горизонтальной прокруткой, а ветвление можно организовать по результатам появления класса no-svg (поскольку Android 2-2.3 не поддерживает и SVG). И наконец, можно объединить тесты для создания собственного пользовательского теста. Эта тема слегка выходит за рамки вопросов, рассматриваемых в книге, но если она вас заинтересует, обратитесь к материалам по адресу http://benfrain.com/combining-modernizr-tests-create-custom-convenience-forks/.

Новые селекторы в CSS3 и порядок их использования

В CSS3 имеются очень эффективные средства выбора элементов внутри страницы. Может, это прозвучит для вас слишком оптимистично, но уж поверьте, они смогут облегчить вашу жизнь и заставят полюбить CSS3! И я готов ответить за столь смелое утверждение.

Селекторы атрибутов в CSS3

Возможно, вы уже пользовались селекторами атрибутов CSS для создания правил. Рассмотрим, к примеру, следующее правило:

```
img[alt] {
    border: 3px dashed #e15f5f;
}
```

Оно будет нацелено на каждый использованный в разметке тег изображения, у которого имеется атрибут `alt`. Или же предположим, что нам нужно выбрать все элементы с атрибутом `data-sausage`:

```
[data-sausage] {
    /* Стили */
}
```

Для указания атрибута понадобятся всего лишь квадратные скобки.

СОВЕТ

Атрибут типа data-* был введен в HTML5 для предоставления места пользовательским данным, которые не могут быть должным образом сохранены с помощью других механизмов. Описание спецификации можно найти по адресу http://www.w3.org/TR/2010/WD-html5-201 01019/elements.html.

Можно сузить область выбора, указав значение атрибута. Рассмотрим, к примеру, следующее правило:

```
img[alt="sausages"] {
    /* Стили */
}
```

Оно будет нацелено только на те изображения, которые имеют атрибут `alt` со значением `sausages`, например:

```
<img class="oscarMain" src="img/sausages.png" alt="sausages" />
```

До сих пор мы довольствовались тем, что предоставлялось спецификацией CSS2. Интересно, а что же нового появилось с выходом спецификации CSS3?

Селекторы CSS3, соответствующие подстрокам значений атрибутов

CSS3 позволяет выбирать элементы на основе подстрок значений их атрибутов. Хотя данная формулировка воспринимается с трудом, но ничего сложного здесь нет! Вариантов соответствия подстроке значения атрибута всего три:

- подстрока находится в начале значения;
- значение содержит экземпляр подстроки;
- значение заканчивается подстрокой.

Посмотрим, как они выглядят.

Селектор значения атрибута по подстроке, находящейся в его начале

Рассмотрим следующую разметку:

```
<img src="img/ace-film.jpg" alt="film-ace">
<img src="img/rubbish-film.jpg" alt="film-rubbish">
```

Оба указанных в ней изображения можно выбрать по подстроке, находящейся в начале значения их атрибута:

```
img[alt^="film"] {
    /* Стили */
}
```

Ключевым здесь выступает символ ^ (который называется «карет», также его часто называют колпаком), означающий «начинается с». Поскольку оба атрибута `alt` начинаются с `film`, селектор выбирает оба тега `img`.

Селектор значения атрибута по имеющемуся в нем экземпляру подстроки

Селектор значения атрибута по имеющемуся в нем экземпляру подстроки имеет следующий синтаксис:

```
[атрибут*="значение"] {
    /* Стили */
}
```

Если нужно, этот селектор, как и все селекторы атрибутов, может использоваться в сочетании с селектором типа (который ссылается на фактически используемый элемент HTML), хотя лично я поступлю таким образом только в случае крайней необходимости (в том случае, если понадобится сменить тип используемого элемента).

Обратимся к примеру и рассмотрим следующую разметку:

```
<p data-ingredients="scones cream jam">Will I get selected?</p>
```

Этот элемент можно выбрать следующим образом:

```
[data-ingredients*="cream"] {
    color: red;
}
```

Ключевым здесь выступает символ *, который в данном контексте означает «содержит».

Селектор «начинается с» с этой разметкой работать не будет, поскольку строка, являющаяся значением атрибута, не начинается с `"cream"`. Но в ней содержится `"cream"`, следовательно, селектор значения атрибута, работающий по принципу «содержит экземпляр», данный элемент обязательно найдет.

Селектор значения атрибута по подстроке, находящейся в его конце

Селектор значения атрибута по подстроке, находящейся в его конце, имеет следующий синтаксис:

```
[атрибут$="значение"] {
    /* Стили */
}
```

Разобраться в нем поможет следующий пример. Рассмотрим его разметку:

```
<p data-ingredients="scones cream jam">Will I get selected?</p>
<p data-ingredients="toast jam butter">Will I get selected?</p>
<p data-ingredients="jam toast butter">Will I get selected?</p>
```

Предположим, нам нужно выбрать тот элемент, в значении атрибута `data-ingredients` которого имеются слова `scones`, `cream` и `jam` (то есть первый элемент). Мы не можем воспользоваться селектором подстроки значения атрибута, работающим по принципу «содержит экземпляр» (он выберет все три варианта) или «начинается с» (он выберет только последний элемент). Но мы можем воспользоваться селектором подстроки значения атрибута, работающим по принципу «заканчивается подстрокой»:

```
[data-ingredients$="jam"] {
    color: red;
}
```

Ключевым символом здесь является знак доллара (`$`), означающий «заканчивается подстрокой».

Особенность выбора по атрибутам

Важно уяснить, что у выбора по атрибутам есть одна особенность: значения атрибутов рассматриваются как цельные строковые значения. Рассмотрим следующее правило CSS:

```
[data-film^="film"] {
    color: red;
}
```

Как ни странно, этот селектор в отношении следующего значения атрибута не сработает, даже притом что одно из слов внутри атрибута начинается с сочетания `film`:

```
<span data-film="awful moulin-rouge film">Moulin Rouge is dreadful</span>
```

Причина в том, что в данном случае атрибут `data-film` начинается не с `film`, а с `awful` (и если вы видели фильм «Мулен Руж», то знаете, что у него также жуткое начало, так и не переходящее затем в более позитивное русло).

Кроме применения недавно рассмотренных селекторов на совпадение с подстрокой есть еще несколько способов обхода этой проблемы. Можно воспользоваться селектором списка значений, разделенных пробелами (обратите внимание на символ тильды), пользующимся поддержкой даже старых браузеров вплоть до Internet Explorer 7:

```
[data-film~="film"] {
    color: red;
}
```

Можно выбрать весь атрибут:

```
[data-film="awful moulin-rouge film"] {
    color: red;
}
```

Или же, если нужно сделать выбор только при наличии в значении атрибута двух строк, можно объединить эту пару (или столько строк, сколько нужно) в селекторах подстрок атрибутов, работающих по принципу «содержит экземпляр»:

```
[data-film*="awful"][data-film*="moulin-rouge"] {
    color: red;
}
```

Правильного варианта действий на все случаи жизни не существует, все зависит от сложности структуры той строки, на основании которой вы пытаетесь осуществить свой выбор.

Селекторы атрибутов позволяют выбрать элементы, чьи идентификаторы и классы начинаются с цифр

До появления HTML5 разметка, в которой идентификаторы и имена классов начинаются с цифр, считалась недопустимой. В HTML5 эти ограничения устранены. Что касается идентификаторов (ID), нужно по-прежнему учитывать некоторые обстоятельства. В идентификаторе не должно быть пробелов, и для страницы он должен быть уникальным. Дополнительные сведения можно найти по адресу http://www.w3.org/html/wg/drafts/html/master/dom.html.

Теперь, несмотря на то что значения идентификаторов и классов в HTML5 могут начинаться с цифр, спецификация CSS по-прежнему не позволяет использовать селекторы идентификаторов и классов, начинающиеся с цифр (http://www.w3.org/TR/CSS21/syndata.html).

К счастью, этот запрет легко обойти, воспользовавшись селектором атрибутов, например `[id="10"]`.

Структурные псевдоклассы CSS3

В CSS3 предоставляется более эффективный механизм выбора элементов на основе их местоположения в структуре DOM.

Рассмотрим обработку весьма распространенной конструкции, при которой ведется работа над панелью навигации для более широкого окна просмотра и нам требуется, чтобы в левой стороне экрана были все ссылки, кроме последней.

Согласно сложившейся практике нам понадобилось бы решать эту задачу путем добавления к последней ссылке имени класса, что дало бы возможность выбрать именно эту ссылку:

```
<nav class="nav-Wrapper">
    <a href="/home" class="nav-Link">Home</a>
    <a href="/About" class="nav-Link">About</a>
    <a href="/Films" class="nav-Link">Films</a>
    <a href="/Forum" class="nav-Link">Forum</a>
    <a href="/Contact-Us" class="nav-Link nav-LinkLast">Contact Us</a>
</nav>
```

Но такое решение само по себе может стать проблематичным. Например, иногда весьма трудно даже получить такую систему управления содержимым, которая может добавить класс к последнему элементу списка. К счастью, подобные обстоятельства больше не вызывают опасений. Эту задачу, как и многие другие, теперь можно решить с помощью структурных псевдоклассов CSS3.

Селектор :last-child

В CSS 2.1 уже был селектор, применяющийся для выбора первого элемента списка:

```
div:first-child {
    /* Стили */
}
```

А в CSS3 добавился селектор, который также может соответствовать последнему элементу:

```
div:last-child {
    /* Стили */
}
```

Посмотрим, как с помощью этого селектора решается наша предыдущая задача:

```
@media (min-width: 60rem) {
    .nav-Wrapper {
        display: flex;
    }
    .nav-Link:last-child {
        margin-left: auto;
    }
}
```

Есть также селекторы для тех случаев, когда имеется только один элемент `:only-child` и только один элемент заданного типа `:only-of-type`.

Селекторы nth-child

Селекторы `nth-child` позволяют решать куда более трудные задачи. Давайте с той же разметкой, что и прежде, посмотрим, как селекторы `nth-child` позволяют выбрать любую ссылку (или ссылки) внутри списка.

Прежде всего посмотрим, как можно выбрать каждый второй элемент списка. Это можно сделать с помощью следующего кода:

```
.nav-Link:nth-child(odd) {
    /* Стили */
}
```

А так можно выбрать каждый первый элемент:

```
.nav-Link:nth-child(even) {
    /* Стили */
}
```

Усвоение порядка работы nth-правил

Непосвященные могут проявить излишнюю настороженность по отношению к селекторам на nth-основе. Но, как только вы усвоите логику и синтаксис, вы будете удивлены тому, на что они способны. Взглянем на предоставляемые ими возможности.

CSS3 обеспечивает невероятную гибкость, позволяя нам использовать ряд правил на nth-основе:

- `nth-child(n)`;
- `nth-last-child(n)`;
- `nth-of-type(n)`;
- `nth-last-of-type(n)`.

Мы уже видели, как в выражениях на nth-основе можно использовать значения `(odd)` или `(even)`, а вот параметр `(n)` может использоваться следующими двумя способами:

- в виде целого числа, например `:nth-child(2)`, что приведет к выбору второго элемента;
- в виде числового выражения, например `:nth-child(3n+1)`, благодаря чему выбор начнется с первого элемента и продолжится выбором каждого третьего элемента.

Разобраться со свойством на основе целого числа нетрудно, нужно просто ввести номер того элемента, который следует выбрать.

Версия селектора с числовым выражением может немного озадачить неискушенных в ее использовании. Если у вас с математикой проблем нет, то я извиняюсь за следующий раздел, а всем остальным советую как следует изучить его.

Разбираемся с математикой. Рассмотрим десять линейных блоков span на странице (посмотреть их в работе можно, запустив пример из файла каталога example_05-05):

```
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
<span></span>
```

Изначально им будет задан следующий стиль:

```
span {
    height: 2rem;
    width: 2rem;
    background-color: blue;
    display: inline-block;
}
```

Нетрудно представить, что в результате мы получим выстроенные в линию десять квадратов.

А теперь посмотрим, как можно выбрать различные элементы с помощью селекции на nth-основе.

Из соображений практичности при разборе выражения в скобках я начну с правого края. К примеру, если я хочу определить, что будет выбрано с помощью выражения (2*n* + 3), то начинаю с крайнего справа числа (число 3 указывает на третий элемент слева) и знаю, что наряду с ним, начиная с его позиции, будет выбран каждый второй элемент. Поэтому добавление правила:

```
span:nth-child(2n+3) {
    color: #f90;
    border-radius: 50%;
}
```

даст на экране браузера следующий результат.

Как видите, наш nth-селектор нацелен на третий элемент списка, а также на каждый последующий второй элемент после него (если бы в списке было 100 элементов, то продолжался бы выбор каждого второго из них).

А как выбрать каждый последующий элемент, начиная со второго? Хотя можно воспользоваться кодом `:nth-child(1n+2)`, первая цифра вообще-то не нужна, поскольку, если не утверждается ничто иное, переменная `n` равна единице. Поэтому можно просто написать `:nth-child(n+2)`. По аналогии с этим, если нам нужно выбрать каждый третий элемент, вместо записи `:nth-child(3n+3)` можно просто указать `:nth-child(3n)`, поскольку каждый третий элемент все равно будет начинаться с третьего элемента и нет необходимости указывать именно на него. В выражении могут применяться также отрицательные числа. Например, при использовании выражения `:nth-child(3n-2)` выбор начинается с `-2`, а затем выбирается каждый третий элемент.

Можно также изменить направление. По умолчанию, как только найдена первая часть заданного выбора, последующие части ищутся вниз по элементам в DOM-дереве (и поэтому выбор в нашем примере идет слева направо). Но направление можно изменить на обратное, поставив минус, например:

```
span:nth-child(-2n+3) {
    background-color: #f90;
    border-radius: 50%;
}
```

В этом примере также ищется третий элемент, но затем поиск идет в обратном направлении с целью выбора каждого второго элемента (вверх по DOM-дереву, поэтому в нашем примере — справа налево).

Надеюсь, теперь-то вы усвоили логику выражений на nth-основе?

Разница между `nth-child` и `nth-last-child` заключается в том, что вариант `nth-last-child` работает в противоход распространению дерева документа. Например, `:nth-last-child(-n+3)` начинает выбор с 3 с конца, а затем выбирает все элементы после него. Результат применения данного правила будет иметь на экране браузера следующий вид.

И в заключение рассмотрим селекторы `:nth-of-type` и `:nth-last-of-type`. В предыдущих примерах вычисление шло в отношении всех дочерних элементов независимо от их типа (следует твердо усвоить, что селектор `nth-child` нацеливается на все дочерние элементы одного и того же DOM-уровня независимо от их классов), а вот селекторы `:nth-of-type` и `:nth-last-of-type` позволяют указывать тип подлежащих выбору элементов. Рассмотрим следующую разметку (файл с которой можно найти в каталоге example_05-06):

```
<span class="span-class"></span>
<span class="span-class"></span>
<span class="span-class"></span>
<span class="span-class"></span>
<span class="span-class"></span>
<div class="span-class"></div>
<div class="span-class"></div>
<div class="span-class"></div>
<div class="span-class"></div>
<div class="span-class"></div>
```

Если воспользоваться следующим селектором:

```
.span-class:nth-of-type(-2n+3) {
    background-color: #f90;
    border-radius: 50%;
}
```

то независимо от наличия у всех элементов одного и того же класса `span-class` нацеливание произойдет только на `span`-элементы (поскольку они относятся к выбираемому типу). В результате будут выбраны следующие элементы.

Вскоре мы увидим, как эта же задача решается с помощью селекторов из спецификации CSS4.

СЧЕТЧИКИ В CSS3 РАБОТАЮТ НЕ ТАК, КАК В JAVASCRIPT И JQUERY!

Если вы привыкли пользоваться JavaScript и jQuery, то должны быть в курсе, что подсчет там ведется с нуля и выше (используется индексация, начинающаяся с нуля). Например, при выборе элемента в JavaScript или jQuery целочисленное значение 1 будет фактически указывать на второй по счету элемент. Но в CSS3 счет начинается с 1, поэтому значение 1 соответствует первому элементу.

Выбор на nth-основе в адаптивных веб-конструкциях

Просто для того, чтобы завершить этот небольшой раздел, хочу проиллюстрировать реальную проблему адаптивного веб-дизайна и способ применения выбора на `nth`-основе для ее решения.

Помните горизонтальную прокручиваемую панель, файлы для создания которой можно найти в каталоге example_05-02? Посмотрим, как это может выглядеть в той ситуации, при которой осуществить горизонтальную прокрутку невозможно. Используя ту же разметку, поместим верхнюю десятку самых кассовых фильмов 2014 года в решетку grid-системы. Для некоторых окон просмотра решетка будет шириной всего в два элемента, по мере увеличения окна просмотра будут показаны три элемента, при еще более крупных размерах — четыре элемента. Но при этом возникает проблема. Независимо от размера окна просмотра нам бы не хотелось, чтобы у самой нижней строки элементов имелась нижняя граница. Файл с соответствующим кодом можно найти в каталоге example_05-09.

Результат при ширине четыре элемента можно увидеть на следующей странице.

Видите эту совершенно ненужную границу ниже последних двух элементов? Именно ее и следует удалить. Но мне нужно получить надежное решение, чтобы граница удалялась и при другом составе элементов последней строки. Теперь, поскольку при различных окнах просмотра в каждой строке содержится разное количество элементов, нам также нужно для разных окон просмотра изменить выбор на `nth`-основе. Для краткости я покажу вам выбор, соответствующий четырем элементам в строке (для самого большого из окон просмотра). Чтобы увидеть измененный выбор при различных окнах просмотра, можете просмотреть образец кода:

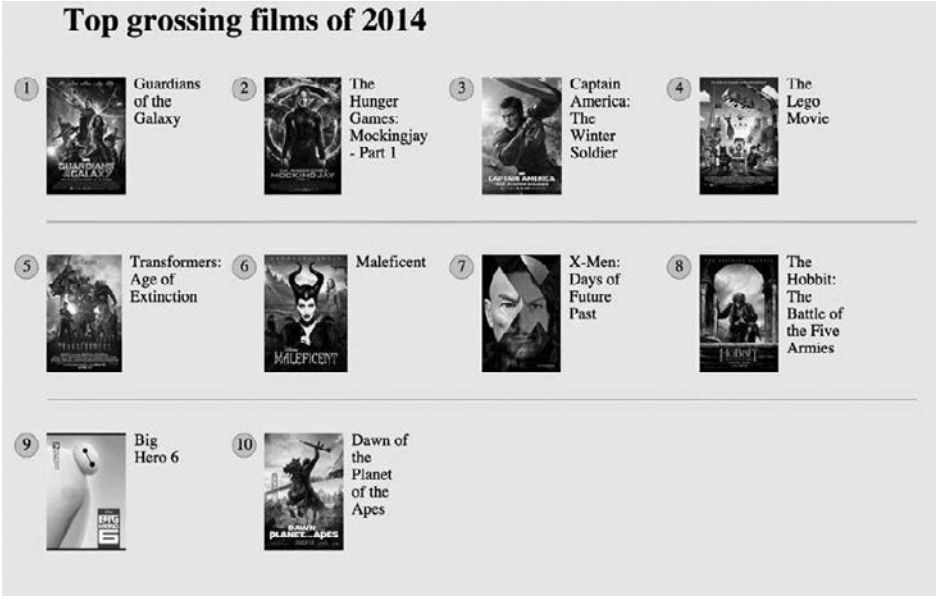
```
@media (min-width: 55rem) {
    .Item {
        width: 25%;
    }
```


```
    /* Получить каждый четвертый элемент, а из них
       взять только тот, который находится в составе
       последних четырех элементов */
    .Item:nth-child(4n+1):nth-last-child(-n+4),
    /* Теперь получить каждый элемент, следующий
        за той же самой коллекцией. */
    .Item:nth-child(4n+1):nth-last-child(-n+4) ~ .Item {
        border-bottom: 0;
    }
}
```

ПРИМЕЧАНИЕ

Как видите, здесь выстроена цепочка из селекторов псевдоклассов на nth-основе. Важно понимать, что предыдущие селекторы не фильтруют выбор для следующих, а элемент должен соответствовать каждому из выборов. Для предыдущего примера первым должен быть элемент из четырех, он должен быть также одним из четырех последних.

Отлично! Благодаря выбору на `nth`-основе у нас получился стойкий набор правил для удаления самой нижней границы независимо от размера окна просмотра или количества показываемых на экране элементов.



Селектор отрицания (:not)

Еще одним полезным селектором является селектор псевдокласса отрицания. Он пригодится для выбора всего, что не подпадает под какой-либо другой выбор. Рассмотрим следующий код:

```
<div class="a-div"></div>
<div class="a-div"></div>
<div class="a-div"></div>
<div class="a-div not-me"></div>
<div class="a-div"></div>
```

И вот эти стили:

```
div {
    display: inline-block;
    height: 2rem;
    width: 2rem;
    background-color: blue;
}

.a-div:not(.not-me) {
    background-color: orange;
    border-radius: 50%;
}
```

Последнее правило предписывает каждому элементу, имеющему класс `.a-div`, быть оранжевого цвета и иметь скругление, за исключением `div`-контейнера, у которого также имеется класс `.not-me`. Файл с этим кодом находится в каталоге

example_05-07 образцов кода (напоминаю, что все образцы можно взять на сайте http://rwd.education/).

СОВЕТ

До сих пор мы рассматривали в основном то, что называется структурными псевдоклассами (вся информация о них доступна по адресу http://www.w3.org/TR/selectors/). Но в CSS3 имеется гораздо больше селекторов. При работе над веб-приложением стоит просмотреть полный перечень элементов пользовательского интерфейса, относящихся к псевдоклассам (http://www.w3.org/TR/selectors/), поскольку они могут, к примеру, нацеливать правила на основе того, выбраны некоторые элементы или нет.

Селектор пустого элемента (:empty)

Мне приходилось сталкиваться с ситуациями, когда элемент, включающий внутренние отступы, получает динамически вставляемое в него содержимое. Иногда он получает содержимое, а иногда — нет. Проблема в том, что, даже когда он не включает содержимое, я все равно вижу отступ. Рассмотрим код HTML и код CSS, которые находятся в файле каталога example_05-08:

```
<div class="thing"></div>
.thing {
    padding: 1rem;
    background-color: violet;
}
```

При отсутствии в этом `div`-контейнере содержимого все равно виден его фоновый цвет, задаваемый свойством `background-color`. К счастью, его нетрудно спрятать:

```
.thing:empty {
    display: none;
}
```

Но все же селектор `:empty` нужно применять с оглядкой. Например, можно подумать, что данный контейнер пуст:

```
<div class="thing"> </div>
```

Но ведь это не так! Обратите внимание на пробел в контейнере. Пробельный символ не означает пустоту!

И чтобы не запутаться, следует знать, что комментарий не влияет на наличие или отсутствие пробельного символа в элементе. Например, этот элемент-контейнер все равно будет считаться пустым:

```
<div class="thing"><!--I'm empty, honest I am--></div>
```

ДОПОЛНЕНИЕ К ПСЕВДОЭЛЕМЕНТАМ

Псевдоэлементы используются со времен появления CSS2, но в спецификации CSS3 синтаксис их использования был слегка пересмотрен. Чтобы освежить вашу память, скажу, что до сих пор p:first-line указывал на первую строку тега <p> или p:first-letter указывал на первую букву. А CSS3 требует отделять эти псевдоэлементы от псевдоклассов, таких как nth-child().

Поэтому теперь нужно использовать запись p::first-letter. Но следует заметить, что Internet Explorer 8 и предыдущие версии не понимают синтаксис с двойным двоеточием, они понимают только синтаксис с одним двоеточием.

Работа с :first-line независимо от размеров окна просмотра

Одно из качеств псевдоэлемента `:first-line`, которое может вам особенно пригодиться, заключается в его работе с учетом специфики окна просмотра. Например, если написать следующее правило:

```
p::first-line {
    color: #ff0cff;
}
```

то, как и ожидалось, первая строка будет выведена на экран в ужасном розовом оттенке. Но при другом размере окна просмотра в ней будет выведена другая подборка текста.

Следовательно, при реализации адаптивного дизайна вам предоставляется удобный способ визуального отображения первой строки в стиле, отличающемся от стиля других строк, без необходимости внесения изменений в разметку (браузер выведет ее на экран, не считаясь с тем, как она обозначена в разметке).

Пользовательские свойства и переменные в CSS

Благодаря популярности препроцессоров CSS-таблиц эти таблицы стали приобретать все больше функций, присущих программированию. Первыми из таких функций стали пользовательские свойства. Нам они больше известны как переменные, хотя это не является их обязательным и единственным предназначением. Полную спецификацию можно найти по адресу http://dev.w3.org/csswg/css-variables/. Но имейте в виду, что на начало 2015 года реализация этой функции в браузерах была весьма скудной — ограничивалась браузером Firefox.

Пользовательские свойства CSS позволяют хранить в таблицах стилей информацию, которая затем может использоваться в таблице стилей или, возможно, оказывать влияние на сценарий JavaScript. Очевидным вариантом использования будет хранение имени семейства шрифтов с последующей ссылкой на него:

```
:root {
    --MainFont: 'Helvetica Neue', Helvetica, Arial, sans-serif;
}
```

Здесь с целью хранения пользовательского свойства в корне документа используется псевдокласс `:root` (хотя эти свойства можно хранить в любом выбранном вами правиле).

СОВЕТ

В структуре документа псевдокласс :root всегда ссылается на самый верхний родительский элемент. В HTML-документе им всегда будет тег HTML, но в SVG-документе, который рассматривается в главе 7, он будет ссылаться на другой элемент.

Пользовательское свойство всегда начинается с двух дефисов, за которыми следует выбранное пользователем имя, а его окончание обозначается, как и любое другое свойство в CSS, точкой с запятой.

Сослаться на значение этого свойства можно с помощью записи `var()`:

```
.Title {
    font-family: var(--MainFont);
}
```

Очевидно, таким образом вы можете хранить любое количество пользовательских свойств. Основным преимуществом такого хранения является возможность изменения значения внутри переменной, и каждое использующее эту переменную правило получает новое значение без его непосредственного изменения.

Предусматривается, что в будущем эти свойства смогут проходить синтаксический анализ и использоваться кодом JavaScript. Более подробную информацию о такого рода невероятных вещах можно найти в описании нового модуля CSS Extensions: http://dev.w3.org/csswg/css-extensions/.

CSS-функция calc

Сколько раз приходилось создавать код разметки и думать о чем-нибудь вроде следующего: «Нужно от половины ширины родительского элемента отнять точно 10 пикселов». Возможность выполнять подобные вычисления особенно пригодилась бы при разработке адаптивного веб-дизайна, поскольку нам никогда заранее не известен размер экрана, на котором будут просматриваться наши веб-страницы. К счастью, в CSS теперь есть способ решения этой задачи. Он называется функцией `calc()`. Рассмотрим пример CSS-кода:

```
.thing {
    width: calc(50% - 10px);
}
```

В этой функции поддерживаются сложение, вычитание, деление и умножение, что позволяет решить множество проблем, которые ранее решались только с применением JavaScript.

Эта функция пользуется весьма неплохой поддержкой со стороны браузеров, но заметным исключением являются Android 4.3 и более ранние версии. Соответствующую спецификацию можно найти по адресу http://www.w3.org/TR/css3-values/.

Селекторы CSS Level 4

В спецификации CSS Selectors Level 4, последней доступной версией которой была редакторская правка от 14 декабря 2014 года (http://dev.w3.org/csswg/selectors-4/), имелось несколько новых типов селекторов. Но пока шла работа над книгой, их реализация в браузерах так и не появилась. Поэтому мы рассмотрим только один пример, поскольку спецификация еще может претерпеть всяческие изменения. Им будет селектор реляционного псевдокласса (Relational Pseudo-Class) из раздела «Логические комбинации» (http://dev.w3.org/csswg/selectors-4/) самой последней правки.

Псевдокласс :has

Этот селектор имеет следующий формат:

```
a:has(figcaption) {
    padding: 1rem;
}
```

Он предписывает добавление отступа к любому элементу `a`-тега, содержащему `figcaption`. Выбор можно инвертировать, если задать его в сочетании с псевдоклассом отрицания:

```
a:not(:has(figcaption)) {
    padding: 1rem;
}
```

Отступ будет добавлен в том случае, если элемент `a`-тега не будет содержать `figcaption`.

Буду честен и скажу: в данной правке не так уж много новых селекторов, которые произвели на меня впечатление. Но кто знает, как все обернется к тому моменту, когда они станут пригодны к использованию в браузерах.

Адаптивные меры длины, выражаемые в процентных отношениях применительно к окнам просмотра (vmax, vmin, vh, vw)

Давайте сменим направление. Мы рассматривали тему выбора элементов в нашем адаптивном мире. А как насчет задания их размеров? В CSS Values and Units Module Level 3 (http://www.w3.org/TR/css3-values/) вводятся единицы измерения, выражаемые относительно размеров окна просмотра. Для адаптивного веб-дизайна это очень полезное нововведение, поскольку каждая единица измерения представляет собой процентное отношение к длине окна просмотра:

- vw — для ширины окна просмотра;
- vh — для высоты окна просмотра;
- vmin — для минимального размера окна просмотра, равна самому малому значению — либо vw, либо vh;

- vmax — для максимального размера окна просмотра, равна самому большому значению — либо vw, либо vh.

Все эти меры длины пользуются неплохой поддержкой браузеров (http://caniuse.com/).

Хотите окно, высота которого задается как 90 % от высоты окна браузера? Проще некуда:

```
.modal {
    height: 90vh;
}
```

При всей пользе, ожидаемой от использования единиц измерения относительно окон просмотра, в некоторых браузерах их применение реализовано довольно странно. К примеру, в Safari под управлением iOS 8 при изменении просматриваемой области в ходе прокрутки с верхней части страницы адресная строка сжимается, но никакие изменения в декларируемую высоту окна просмотра не вносятся.

Но, наверное, более полезно применять эти единицы измерения в сочетании с шрифтовыми настройками. Например, создать текст, масштабирующийся в зависимости от размеров окна просмотра, не так-то просто.

Я могу показать вам решение этой задачи прямо сейчас. Но мне хотелось бы использовать вполне определенный шрифт, чтобы независимо от того, в какой системе просматривается пример, Windows, Mac или Linux, мы видели одно и то же.

Хорошо, сознаюсь вам, что это будет простой трюк, позволяющий мне проиллюстрировать возможности использования веб-шрифтов с помощью CSS3.

Шрифтовое веб-оформление

Многие годы в веб-приложениях приходилось довольствоваться скучным выбором «безопасных» шрифтов. Когда для дизайна были важны какие-то особые шрифты, приходилось подставлять графический элемент, а для смещения текста от границ окна просмотра применять правило отступа `text-indent`. Это была та еще работенка!

Но кроме этого, для добавления на страницу особого типографского оформления имелось несколько оригинальных методов. В sIFR (http://www.mikeindustries.com/blog/sifr/) и Cufyn (http://cufon.shoqolate.com/generate/) для переделки текстовых элементов с целью их появления в виде шрифтов, для которых они предназначались, использовались технологии Flash и JavaScript соответственно. К счастью, в CSS3 предоставляются средства пользовательского шрифтового оформления, уже готовые к завоеванию широкой популярности.

CSS-правило @font-face

CSS-правило `@font-face` появилось еще в CSS2 (но затем из CSS 2.1 оно было удалено). Оно даже пользовалось частичной поддержкой Internet Explorer 4 (кроме шуток!). А как обстоят дела, когда речь заходит о CSS3?

Как выясняется, правило `@font-face` было снова введено для модуля шрифтов CSS3 Fonts Module (http://www.w3.org/TR/css3-fonts). Из-за исторически сложившейся правовой тяжбы, касающейся использования шрифтов в веб-приложениях, серьезные подвижки в фактическом признании данного решения для шрифтового оформления начались только в последние годы.

Как обычно, когда в веб-мире речь идет о привлечении каких-либо ресурсов, единого формата файлов не существует. Точно так же, как изображения могут приходить в JPG, PNG, GIF и других форматах, у шрифтов имеется собственный набор форматов на выбор. Для Internet Explorer (и больше никакого другого браузера) в качестве наиболее предпочтительных были выбраны шрифты формата Embedded OpenType с расширениями имен файлов .eot. Для других браузеров предпочтение отдавалось более распространенным шрифтам формата TrueType с расширениями имен файлов .ttf. В то же время существуют шрифты форматов SVG-графики и Web Open Font Format с расширениями имен файлов .woff или .woff2.

Сейчас, чтобы охватить различные браузерные реализации, необходимо обслуживать несколько файловых версий одного и того же шрифта.

Но есть и хорошая новость: добавление каждого пользовательского шрифта для любого браузера не составляет особого труда. Посмотрим, как это делается!

Реализация веб-шрифтов с помощью @font-face

CSS предоставляет так называемое эт-правило (по названию символа @ — «эт») `@font-face`, позволяющее ссылаться на онлайн-шрифты, которые затем можно будет использовать для отображения текста.

Существует большое количество великолепных источников, позволяющих просматривать и получать веб-шрифты, среди которых как бесплатные, так и платные. Что касается источников бесплатных шрифтов, то я отдаю предпочтение Font Squirrel (http://www.fontsquirrel.com/), хотя Google также предоставляет бесплатные веб-шрифты, которые в конечном счете подаются с помощью правила `@font-face` (http://www.google.com/webfonts). Существуют также отличные платные сервисы от Typekit (http://www.typekit.com/) и Font Deck (http://www.fontdeck.com/).

Для данного упражнения я собираюсь загрузить Roboto. Это шрифт, используемый последними Android-телефонами, и если у вас имеется такой телефон, он должен быть вам знаком. В противном случае все, что вам нужно знать о нем, сводится к тому, что это весьма привлекательный шрифт интерфейса, разработанный для обеспечения высокой разборчивости на небольших экранах. Вы сами можете его получить по адресу http://www.fontsquirrel.com/fonts/roboto.

ПРИМЕЧАНИЕ

Если есть возможность загрузить поднабор своего шрифта, применяемый для языка, который вы собираетесь использовать, именно так и нужно сделать. Это означает, что в результате будет получен файл, намного меньший по размеру, чем тот файл, в котором содержатся образы символов для языков, которые вы не собираетесь использовать.

После загрузки набора `@font-face` загляните в ZIP-файл, где обнаружатся папки различных шрифтов Roboto. Я выбираю версию Roboto Regular. Внутри этой папки шрифт присутствует в различных файловых форматах (WOFF, TTF, EOT и SVG), вдобавок там имеется файл stylesheet.css, содержащий стек шрифтов. Например, правило для Roboto Regular имеет следующий вид:

```
@font-face {
    font-family: 'robotoregular';
    src: url('Roboto-Regular-webfont.eot');
    src: url('Roboto-Regular-webfont.eot?#iefix') format('embedded-opentype'),
        url('Roboto-Regular-webfont.woff') format('woff'),
        url('Roboto-Regular-webfont.ttf') format('truetype'),
        url('Roboto-Regular-webfont.svg#robotoregular')
    format('svg');
        font-weight: normal;
        font-style: normal;
}
```

Во многом по такому же принципу, как и при работе с префиксами производителей, браузер будет применять стили из этого перечня свойств (начиная с имеющего приоритет самого нижнего свойства, если оно применимо), игнорируя те из них, которые ему непонятны. Поэтому, какой бы браузер ни применялся, использоваться будет тот шрифт, которым он сможет воспользоваться.

Теперь, несмотря на то что этот блок кода вполне достоин копирования и вставки, важно обратить внимание на пути, по которым можно найти сохраненные шрифты. Например, я склонен копировать шрифты из ZIP-файла и сохранять их в папке с незатейливым названием fonts, находящейся на том же уровне, что и моя папка css. Следовательно, поскольку я обычно копирую это правило со стеком шрифтов в свою основную таблицу стилей, мне нужно изменить пути. И тогда правило приобретет следующий вид:

```
@font-face {
    font-family: 'robotoregular';
    src: url('../fonts/Roboto-Regular-webfont.eot');
    src: url('../fonts/Roboto-Regular-webfont.eot?#iefix')
    format('embedded-opentype'),
        url('../fonts/Roboto-Regular-webfont.woff') format('woff'),
        url('../fonts/Roboto-Regular-webfont.ttf')
    format('truetype'),
        url('../fonts/Roboto-Regular-webfont.svg#robotoregular')
    format('svg');
    font-weight: normal;
    font-style: normal;
}
```

Затем наступает время установки для соответствующего правила стилевого оформления нужного шрифта и его плотности (если необходимо). Посмотрите на файл примера из каталога example_05-10 — там такая же разметка, как и в файле примера из каталога example_05-09, мы просто объявляем это семейство шрифтов в качестве исходного:

```
body {
    font-family: robotoregular;
}
```

Дополнительное преимущество веб-шрифтов проявляется в том, что, если в комплексе используются те же шрифты, что и в вашем коде, вы можете подключить размеры непосредственно из файла комплекса. Например, если в Photoshop размер шрифта составляет 24 пиксела, мы либо подключаем это значение в неизменном виде, либо переводим его в более гибкую единицу измерения, такую как rem (при условии, что размер основного шрифта составляет 16 пикселов, 24 / 16 = 1,5 rem).

Но, как упоминалось ранее, теперь в нашем распоряжении есть размеры, задаваемые относительно размеров окна просмотра. Здесь ими можно воспользоваться для масштабирования текста относительно размеров окна просмотра:

```
body {
    font-family: robotoregular;
    font-size: 2.1vw;
}

@media (min-width: 45rem) {
    html,
    body {
        max-width: 50.75rem;
        font-size: 1.8vw;
    }
}

@media (min-width: 55rem) {
    html,
    body {
        max-width: 78.75rem;
        font-size: 1.7vw;
    }
}
```

Если открыть этот пример в браузере и изменить размеры окна просмотра, можно увидеть, что благодаря всего лишь нескольким строкам кода CSS мы получили текст, масштабируемый под доступное пространство. Великолепно!

Предостережение, касающееся пользовательского шрифтового оформления с применением @font-face в адаптивных веб-конструкциях

В целом метод шрифтового оформления веб-приложений `@font-face` представляется весьма эффективным средством. Единственное предостережение, касающееся использования этой технологии с адаптивными конструкциями, относится к размеру файла шрифта. Если устройство, на экран которого выводится наш пример, требует для шрифта Roboto Regular формата SVG, то по сравнению с применением стандартных веб-безопасных шрифтов, например Arial, необходимо извлечь

дополнительные 34 Кбайт данных. Нами в примере использовался английский поднабор символов, что сократило размер файла, но такая возможность предоставляется далеко не всегда. Если нужно добиться от сайта наивысшей производительности, следует контролировать размеры пользовательских шрифтов и разумно подходить к их использованию.

Новые форматы цвета в CSS3 и альфа-прозрачность

До сих пор в данной главе рассматривались вопросы предоставляемых CSS3 новых эффективных возможностей выбора элементов и добавления к нашим конструкциям пользовательского шрифтового оформления. Теперь будут рассмотрены обеспечиваемые CSS3 способы работы с цветом, которые раньше были просто недоступны.

Прежде всего, CSS3 предоставляет два новых способа объявления цвета: RGB и HSL. Кроме того, эти два формата позволяют нам использовать вместе с ними альфа-канал (соответственно RGBA и HSLA).

Цвет в формате RGB

Красный, зеленый и синий — *Red*, *Green* и *Blue* (RGB) — это система задания цвета, которая существует уже не одно десятилетие. Она работает путем определения значений для красного, зеленого и синего компонентов цвета. Например, красный цвет может быть определен в CSS в виде шестнадцатеричного значения (HEX-значения) `#fe0208`:

```
.redness {
    color: #fe0208;
}
```

СОВЕТ

Если хотите получить описание способа более интуитивно понятного восприятия значений, выраженных шестнадцатеричными числами, я могу порекомендовать следующую статью в Smashing Magazine: http://www.smashingmagazine.com/2012/10/04/the-code-side-of-color/.

Но с помощью CSS3 этот цвет может быть указан также в виде RGB-значения:

```
.redness {
    color: rgb(254, 2, 8);
}
```

Большинство приложений редактирования изображений показывают цвета на панели их выбора в виде как HEX-, так и RGB-значений. У панели выбора цвета программы Photoshop имеются R-, G- и B-области, показывающие значения для каждого из этих каналов. Например, значение R может выражаться числом 254, G — числом 2, а B — числом 8. Все это весьма просто преобразуется в значение цве-

та для CSS. В CSS после определения режима задания цвета (например, RGB) значения для красного, зеленого и синего цветов отделяются друг от друга запятыми и записываются внутри скобок именно в таком порядке, как показано в предыдущем примере кода.

Цвет в формате HSL

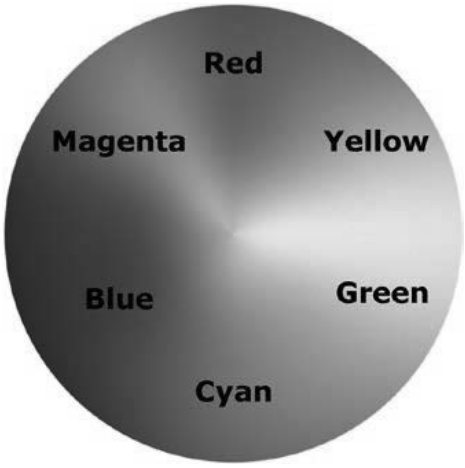
Кроме формата RGB, CSS3 позволяет использовать для объявления цвета значения в формате «тон, насыщенность и яркость» — *Hue*, *Saturation* и *Lightness* (HSL).

НЕ НУЖНО ПУТАТЬ HSL И HSB!

Не следует ошибочно полагать, что значение в формате Hue, Saturation и Brightness (HSB), показанное на панели выбора цвета в таких приложениях редактирования изображений, как Photoshop, — это то же самое, что и HSL. Это не так!

Востребованность формата HSL объясняется относительной простотой понимания того, какой именно цвет будет представлен при заданных им значениях. Например, если вы не обладаете какими-то сверхспособностями по подбору цвета, держу пари, что вам не удастся с ходу сказать, какой это цвет: `rgb(255, 51, 204)`. Есть желающие? Только не я. А вот покажите мне HSL-значение `hsl(315, 100%, 60%)`, и я могу высказать догадку, что это где-то между пурпурным и красным цветом (хотя это ярко выраженный розовый цвет). Как я догадался? Да очень просто.

HSL-формат основан на цветовом диске в 360°. Вот как он выглядит.



Первая цифра в цветовом определении формата HSL представляет собой тон — Hue. Глядя на диск, можно понять, что желтый (Yellow) находится на отметке 60°, зеленый (Green) — 120°, светло-голубой (Cyan) — 180°, синий (Blue) — 240°, малиновый (Magenta) — 300° и, наконец, красный (Red) — 360°. Следовательно, если вышеупомянутый цвет в формате HSL имеет тон 315, нетрудно понять, что он будет располагаться между малиновым (Magenta), который находится на отметке 300°, и красным (Red), находящимся на отметке 360°.

Следующие два значения в HSL-определении задают насыщенность и яркость и указываются в процентах. Ими просто задается изменение базовой насыщенности. Для более насыщенного, красочного вида во втором значении указывается более высокий процент. Последнее значение управляет яркостью (Lightness) и может варьироваться от 0 для чистого черного цвета до 100 % для абсолютно белого цвета.

Следовательно, если цвет определен в виде HSL-значений, проще будет создать и его вариации, изменяя процентные показатели насыщенности и яркости. Например, красный цвет может быть определен в HSL-значениях следующим образом:

```
.redness {
    color: hsl(359, 99%, 50%);
}
```

Если нужен чуть более темный цвет, можно воспользоваться теми же HSL-значениями, изменив всего лишь процентный показатель яркости (последнее из значений):

```
.darker-red {
    color: hsl(359, 99%, 40%);
}
```

В заключение, если вы в состоянии запомнить для цветового диска HSL мнемоническое правило *Young Guys Can Be Messy Rascals* (или любое другое удобное для вас выражение), то сможете приблизительно задать значение цвета в формате HSL, для чего не придется обращаться к панели выбора цвета, а кроме этого, задать еще и его оттенок. Покажите этот трюк на корпоративной вечеринке в обществе парней и девушек из компании разработчиков на Ruby, Node или .NET и сможете быстро набрать очки, повысив свой престиж!

Альфа-каналы

До сих пор мне приходилось терпеть ваше едва сдерживаемое удивление по поводу того, к чему вся эта возня вокруг HSL или RGB и почему нужно отдавать им предпочтение над старым, годами проверенным форматом с использованием HEX-значений. Основное отличие HSL и RGB от HEX заключается в том, что эти два формата позволяют использовать канал альфа-прозрачности. Это дает возможность видеть то, что находится под верхним элементом.

Объявление цвета в формате HSLA похоже по синтаксису на стандартное HSL-правило. Но дополнительно нужно указать значение как `hsla`, а не просто как `hsl` и добавить показатель прозрачности, задаваемый между 0 (полная прозрачность) и 1 (полная непрозрачность), например:

```
.redness-alpha {
    color: hsla(359, 99%, 50%, .5);
}
```

Синтаксис RGBA следует тому же соглашению, что и его HSLA-эквивалент:

```
.redness-alpha-rgba {
    color: rgba(255, 255, 255, 0.8);
}
```

А ПОЧЕМУ БЫ ПРОСТО НЕ ВОСПОЛЬЗОВАТЬСЯ СВОЙСТВОМ OPACITY?

В CSS3 элементам также позволяется иметь установку непрозрачности, задаваемую объявлением свойства opacity. Значение устанавливается между 0 и 1 с десятичным шагом (например, непрозрачность, установленная в .1, означает 10 %). Но данное свойство отличается от RGBA и HSLA тем, что с его помощью устанавливается непрозрачность для всего элемента. А установка значения для HSLA или RGBA позволяет иметь уровень альфа-прозрачности конкретной части элемента. Например, у элемента может быть HSLA-значение для фона, а для текста внутри него задан сплошной цвет.

Работа над цветовым оформлением с CSS Color Module Level 4

Несмотря на то что эта функциональная возможность находится на весьма ранних стадиях спецификации, возможно, в скором будущем вы сможете получить удовольствие от работы над цветовым оформлением в CSS с помощью функции `color()`. А пока для нее не появится широкая браузерная поддержка, с подобными манипуляциями неплохо справляются пре- и постпроцессоры CSS (сделайте себе приятное и купите прямо сейчас книгу по этой теме, я рекомендую остановить выбор на *Sass and Compass for Designers* Бена Фрэйна).

Проследить за ходом разработки спецификации CSS Color Module Level 4 можно по адресу http://dev.w3.org/csswg/css-color-4/.

Резюме

В данной главе были изучены способы, позволяющие легко выбирать практически все, что нам нужно на странице, с помощью новых селекторов CSS3. Были также рассмотрены способы создания за рекордно короткие сроки адаптивных колонок и прокручиваемых панелей с содержимым, а также решения часто возникающих и весьма неприятных проблем с не помещающимися в области ввода длинными URL-адресами путем организации их многострочного вывода. Кроме того, теперь мы получили представление о новом цветовом модуле CSS3 и о том, как применять цвета с RGB и HSL, дополненные прозрачными альфа-слоями, для получения интересных эстетических эффектов.

В данной главе также был рассмотрен вопрос добавления к веб-конструкциям шрифтового оформления с помощью правила `@font-face`, которое наконец-то освободило нас от оков рутинного выбора веб-безопасных шрифтов. Несмотря на все эти впечатляющие новые возможности и технологии, мы из всего, что можно будет сделать с CSS3, коснулись лишь того, что лежит на поверхности. Продолжим наш путь и расширим круг изучаемых способов, позволяющих CSS3 максимально ускорить работу, повысить эффективность и возможности сопровождения адаптивной конструкции с помощью имеющихся в CSS3 текстовых и блочных теней, градиентов и возможностей применения многочисленных фонов.

6 Создание эстетически привлекательных эффектов средствами CSS3

Особая важность средств CSS3 для создания впечатляющих эстетически привлекательных эффектов в адаптивной конструкции обусловливается предоставлением во многих ситуациях возможностей замены ими изображений. Тем самым экономится ваше время, код приобретает повышенную гибкость, облегчается его поддержка, а для конечного пользователя страница становится «легковеснее». Эти преимущества пригодились бы даже в обычной конструкции, рассчитанной на использование на настольной системе и имеющей фиксированную ширину, но все же они важнее в адаптивном дизайне, поскольку использование CSS в характерных для него ситуациях облегчает создание различных эстетически привлекательных эффектов для разных окон просмотра.

В данной главе будут рассмотрены следующие вопросы:

- создание с помощью CSS теней для текста;
- создание с помощью CSS теней для блоков;
- создание с помощью CSS градиентных фонов;
- использование с помощью CSS нескольких фонов;
- использование градиентных фонов CSS3 для создания узоров;
- реализация фоновых изображений с высоким разрешением с помощью медиазапросов;
- использование фильтров CSS и их влияние на производительность.

А теперь приступим к изучению перечисленных вопросов.

ПРЕФИКСЫ ПРОИЗВОДИТЕЛЕЙ

При реализации экспериментальных возможностей CSS не забывайте добавлять соответствующие префиксы производителей, но не вручную, а с использованием специального средства. Тем самым гарантируются самая широкая кросс-браузерная совместимость, а также избавление от добавления уже ненужных префиксов. На момент написания книги в качестве наилучшего инструмента такого плана мне виделось средство Autoprefixer (https://github.com/postcss/autoprefixer), упоминаемое мною в большинстве глав.

Создание теней для текста средствами CSS3

Одним из наиболее широко реализованных средств CSS3 является `text-shadow`. У него, как и у `@font-face`, была своя предыстория, прервавшаяся в CSS 2.1. К счастью, оно вернулось и получило весьма широкую поддержку во всех современных браузерах, начиная с Internet Explorer 9. Рассмотрим базовый синтаксис:

```
.element {
    text-shadow: 1px 1px 1px #ccc;
}
```

Не забудьте, что значения в краткой форме правил всегда идут сначала слева направо, а затем сверху вниз (можно также, если вам так легче, представлять себе направление по часовой стрелке). Следовательно, первое значение будет задавать объем тени справа, второе — объем тени снизу, третье — протяженность размытия (расстояние, проходимое тенью до ее полного исчезновения), четвертое — цвет.

Тени, направленные влево и вверх, можно получить, задав отрицательные значения, например:

```
.text {
    text-shadow: -4px -4px 0px #dad7d7;
}
```

Цветовое значение не обязательно определять в HEX-виде (то есть задавать в виде шестнадцатеричных чисел цветовых составляющих), с тем же успехом оно может быть определено в формате HSL(A) или RGB(A):

```
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

И тем не менее следует иметь в виду, что тогда для вывода эффекта на экран браузер также должен поддерживать не только тени для текста, но и режимы задания цвета HSL/RGB.

Значения, задающие тень, можно указывать и в любых других приемлемых для CSS единицах длины, таких как em, rem, ch и т. д. Лично я при задании значений для свойства `text-shadow` редко пользуюсь единицами длины em или rem. Поскольку значения всегда небольшие, то при всех величинах окон просмотра обычно неплохо выглядят тени, для задания которых используется значение `1px` или `2px`.

Благодаря использованию медиазапросов мы можем также без особого труда удалять текстовые тени при различных окнах просмотра. Ключевым в данном случае выступает значение `none`:

```
.text {
    text-shadow: .0625rem .0625rem 0 #bfbfbf;
}

@media (min-width: 30rem) {
    .text {
        text-shadow: none;
    }
}
```

СОВЕТ

Кстати, вам стоит знать, что в CSS там, где значение начинается с нуля, например 0.14s, лидирующий нуль можно не ставить: .14s будет работать точно так же, как и предыдущее указание значения.

Если размытие не нужно, его значение можно опустить

Если добавлять размытие к значениям свойства `text-shadow` не нужно, то его можно опустить, например:

```
.text {
    text-shadow: -4px -4px #dad7d7;
}
```

Это значение будет вполне допустимым. Браузер поймет, что если не задано третье значение, то первые два значения будут относиться к смещению тени.

Получение нескольких теней для текста

Если разделить задания теней запятыми, то к тексту можно добавить сразу несколько теней, например:

```
.multiple {
    text-shadow: 0px 1px #fff,4px 4px 0px #dad7d7;
}
```

Кроме того, поскольку в CSS на пробельные символы не обращается никакого внимания, подобные значения можно разметить следующим образом, если, конечно, так их будет легче прочитать:

```
.text {
    font-size: calc(100vmax / 40);
    text-shadow:
    3px 3px #bbb, /* справа и внизу */
    -3px -3px #999; /* слева вверху */
}
```

СОВЕТ

W3C-спецификацию, касающуюся свойства text-shadow, можно найти по адресу http://www.w3.org/TR/css3-text/.

Создание теней для блоков

Свойство `box-shadow` позволяет создавать прямоугольные тени как снаружи, так и внутри того элемента, к которому они применяются. После того как вы разобрались с тенями для текста, понять суть теней для блоков будет намного легче. В принципе,

и те и другие следуют одному и тому же синтаксису: смещение по горизонтали, смещение по вертикали, размытие, протяженность (о которой мы вскоре узнаем) и цвет.

Из четырех возможных значений длины обязательными являются только два (в отсутствие последних двух цветовым значением определяется цвет тени, а для радиуса размытия используется нулевое значение). Рассмотрим простой пример:

```
.shadow {
    box-shadow: 0px 3px 5px #444;
}
```

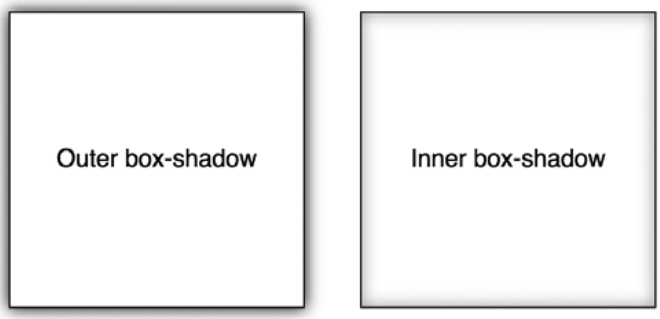
Изначально свойство `box-shadow` настроено на создание тени снаружи элемента. Хотя дополнительное ключевое слово `inset` позволяет применить `box-shadow` для создания тени внутри элемента.

Тень внутри элемента

Свойство `box-shadow` может также использоваться для создания внутренней тени. Синтаксис при этом идентичен тому, который используется для задания обычной тени, за исключением того, что значения начинаются с ключевого слова `inset`:

```
.inset {
    box-shadow: inset 0 0 40px #000;
}
```

Все работает точно так же, как и раньше, но `inset`-составляющая объявления предписывает браузеру применить эффект к внутренней части элемента. Примеры обоих типов можно увидеть, запустив код из каталога example_06-01.



Создание нескольких теней

Как и `text-shadow`, свойство `box-shadow` позволяет применить сразу несколько теней. Разделите значения, задаваемые свойству `box-shadow` запятой, и они будут применены, начиная с нижнего набора значений и заканчивая верхним набором (от последнего к первому). Усвойте мнемоническое правило, согласно которому самое близкое к вершине правило (в коде) проявится как самое верхнее в порядке вывода браузером на экран. Как и с `text-shadow`, вы можете извлечь пользу от примене-

ния пробельных символов для выстраивания визуального столбца из различных наборов значений для свойства box-shadow:

```
box-shadow: inset 0 0 30px hsl(0, 0%, 0%),
            inset 0 0 70px hsla(0, 97%, 53%, 1);
```

СОВЕТ

Выстраивая в коде в столбец нескольких длинных значений по принципу одно под другим, вы получаете дополнительные преимущества при использовании систем управления версиями, поскольку упрощаете обнаружение различий, когда сравниваете две версии файла в режиме их поиска. Именно поэтому я также выстраиваю в столбцы один под другим и селекторы в группах селекторов.

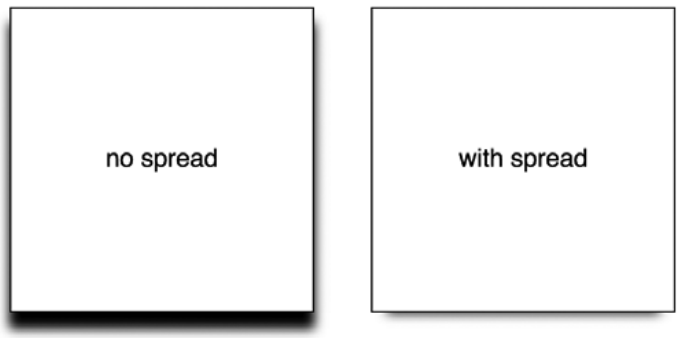
Понятие протяженности

Буду честен: буквально несколько лет назад я никак не мог взять в толк, что на самом деле для свойства box-shadow означает параметр spread. Не думаю, что название spread («распространение») отражает его истинный смысл. Лучше думать о нем как о протяженности. Позвольте пояснить.

Посмотрите на блок, появляющийся слева на экране при запуске кода из каталога example_06-02. К нему применены стандартные значения свойства box-shadow. А к блоку справа применено отрицательное значение spread, заданное четвертым по счету. Вот как выглядит соответствующий код:

```
.no-spread {
    box-shadow: 0 1 0px 10px;
}

.spread {
    box-shadow: 0 1 0px 10px -10px;
}
```

А вот эффект от каждой настройки на экране (элемент с установленным значением spread расположен справа).



Значение spread позволяет за счет его указания расширить или сократить тень по всем направлениям. В этом примере отрицательное значение «задвинуло» тень

по всем направлениям. Получилось, что мы видим только тень внизу, вместо того чтобы увидеть, как размытие растекается во все стороны (поскольку в противовес размытию работает отрицательное значение параметра `spread`).

ПРИМЕЧАНИЕ

W3C-спецификацию, касающуюся свойства box-shadow, можно найти по адресу http://www.w3.org/TR/css3-background/.

Градиентные фоны

В прежние времена, чтобы получить для элемента градиентный фон, нужно было выложить плиткой тонкие графические фрагменты градиента. С точки зрения расхода графических ресурсов это был весьма экономный компромисс. Изображение шириной всего лишь в один или два пиксела не наносило существенного ущерба пропускной способности сети и могло на одном и том же сайте использоваться сразу для нескольких элементов.

Но чтобы настроить градиент, приходилось по-прежнему возвращаться к использованию графического редактора. Кроме того, иногда содержимое могло выплескиваться за пределы градиентного фона, выходя за рамки ограничений фиксированного размера изображений. В адаптируемой конструкции эта проблема усложнялась, поскольку разделы страницы в различных окнах просмотра могли увеличиваться.

Но с применением CSS-градиента `background-image` все стало намного гибче. Являясь частью спецификации *CSS Image Values and Replaced Content Module Level 3*, CSS позволяет создавать линейные и радиальные градиентные фоны. Посмотрим, как их можно определить.

СОВЕТ

Спецификацию CSS Image Values and Replaced Content Module Level 3 можно найти по адресу http://www.w3.org/TR/css3-images/.

Запись линейного градиента

Запись линейного градиента `linear-gradient` в наипростейшей форме выглядит следующим образом:

```
.linear-gradient {
    background: linear-gradient(red, blue);
}
```

Она приведет к созданию линейного градиента, начинающегося с красного (градиент по умолчанию начинается сверху) и постепенно переходящего в синий цвет.

Указание направления градиента

Теперь, если нужно указать направление градиента, можно воспользоваться двумя способами. Начинаться градиент всегда будет с направления, противоположного

указанному. Но когда направление не указано, он всегда будет распространяться сверху вниз, например:

```
.linear-gradient {
    background: linear-gradient(to top right, red, blue);
}
```

В данном примере градиент направляется в правый верхний угол. Он начинается с красного цвета в левом нижнем углу и постепенно переходит в синий цвет в правом верхнем углу.

Если вам ближе математические представления, то можете поверить, что градиент допустимо записать в следующей форме:

```
.linear-gradient {
    background: linear-gradient(45deg, red, blue);
}
```

Но имейте в виду, что в прямоугольном блоке градиент, направляющийся в правый верхний угол, `'to top right'` (что всегда относится к правому верхнему углу того элемента, к которому он применяется), будет заканчиваться немного в другой позиции, чем при указании `45deg` (градиент, заданный под углом 45° от его стартовой позиции).

Полезно будет также знать, что стартовую точку градиентов можно назначать еще до того, как они приобретут видимость внутри блока, например:

```
.linear-gradient {
    background: linear-gradient(red -50%, blue);
}
```

При такой настройке градиент как таковой начинается до обретения видимости внутри блока.

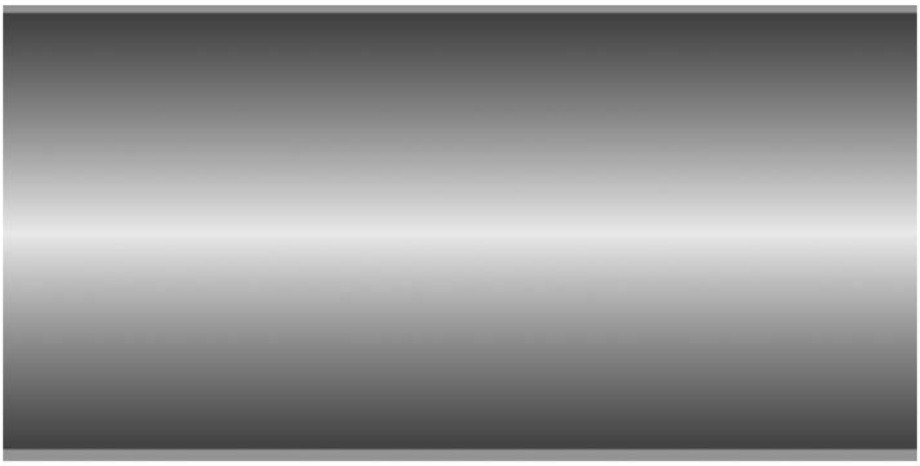
В последнем примере мы также воспользовались цветовой опорной точкой, чтобы определить то место, где цвет должен начинаться и заканчиваться, поэтому разберемся в этом подробнее.

Цветовые опорные точки

Возможно, самыми удобными средствами задания градиентных фонов являются цветовые опорные точки. Они позволяют устанавливать, какой цвет нужно использовать и в какой точке градиента. С помощью цветовых опорных точек можно задать установки любой сложности. Рассмотрим следующий пример:

```
.linear-gradient {
    margin: 1rem;
    width: 400px;
    height: 200px;
    background: linear-gradient(#f90 0, #f90 2%, #555 2%, #eee 50%, #555 98%,
                                #f90 98%, #f90 100%);
}
```

Вот каким будет вывод на экран при таком задании значений для свойства `linear-gradient`..



В данном примере (код которого находится в каталоге example_06-03) направление не указывалось, поэтому было применено исходное направление сверху вниз.

Цветовые опорные точки внутри градиента записаны через запятые и определены путем указания сначала цвета, а затем позиции остановки. Обычно советуют не смешивать единицы измерения в одной и той же записи, но делать это никто не запрещает. Цветовых опорных точек может быть сколько угодно, а цвета могут быть заданы ключевыми словами, а также значениями в форматах HEX, RGBA или HSLA.

СОВЕТ

Следует иметь в виду, что с годами сложилось множество различных синтаксисов градиентных фонов, поэтому данная область относится к тем, в которых невероятно трудно записывать альтернативные варианты вручную. Чтобы не рисковать получить эффект испорченной пластинки (если молодежь не знает, что такое пластинка, нужно спросить об этом у родителей), упростите свою жизнь путем использования средства Autoprefixer. Оно позволит записывать код, соответствующий текущему стандарту синтаксиса W3C (как указывалось ранее), и автоматически создавать для вас предыдущие версии.

Спецификацию W3C, относящуюся к линейным градиентным фонам, можно найти по адресу http://www.w3.org/TR/css3-images/.

Добавление альтернативных вариантов для устаревших браузеров

В качестве альтернативного варианта для устаревших браузеров, не поддерживающих градиентные фоны, нужно просто сначала определить сплошной фоновый цвет. Тогда устаревшие браузеры, если они не понимают градиентов, определяемых после этой установки, будут как минимум выводить на экран сплошной фон, например:

```
.thing {
    background: red;
    background: linear-gradient(45deg, red, blue);
}
```

Радиальные градиентные фоны

Создавать радиальные градиенты в CSS ничуть не сложнее. Как правило, они начинаются в центральной точке и постепенно распространяются в форме эллипса или круга.

Радиальный градиентный фон имеет следующий синтаксис (испытать его на практике можно, запустив код из каталога example_06-04):

```
.radial-gradient {
    margin: 1rem;
    width: 400px;
    height: 200px;
    background: radial-gradient(12rem circle at bottom, yellow, orange, red);
}
```

Анализ синтаксиса radial-gradient. После указания свойства (`background:`) начинается запись `radial-gradient`. Для начала перед первой запятой определяются форма или размер градиента и его позиция.

Для формы и размера в показанном ранее примере мы использовали круг размером 12 rem, но давайте теперь рассмотрим несколько других примеров значений.

- `5em` задаст круг размером 5 em. Если задан только размер, то ту часть, которая определяет форму (`'circle'`), можно опустить.
- `circle` задаст круг на весь размер контейнера (изначально размер радиального градиента, если он не указан, задается до самого дальнего угла, а ключевые слова для задания размера будут рассмотрены позже).
- `40px 30px` определит форму эллипса, как бы прорисованного внутри прямоугольника шириной 40 пикселов и высотой 30 пикселов.
- `ellipse` задаст форму эллипса, помещаемого внутрь элемента.

После размера и/или формы определяется позиция. Позицией по умолчанию является центр, но рассмотрим ряд других возможностей и способы их определения.

- `at top right` задает начало распространения радиального градиента из правого верхнего угла.
- `at right 100px top 20px` устанавливает начало распространения радиального градиента на 100 пикселов от правого края объекта и на 20 пикселов от верхнего края.
- `at center left` задает начало распространения градиента на половине пути вниз по левой стороне элемента.

Наши параметры размеров, формы и позиции завершаются запятой, после чего определяются любые цветовые опорные точки, работающие точно так же, как и в `linear-gradient`.

Упростим описание записи: размер, форма и позиция указываются перед первой запятой, после которой указывается любое количество цветовых опорных точек (эти точки разделяются запятыми).

Удобные ключевые слова распространения для задания размеров адаптивных конструкций

Чтобы добиться адаптивности, может оказаться полезно задавать размеры градиентов в пропорциях, а не в фиксированных пиксельных размерах. Тогда вы будете знать, что защищены (как буквально, так и фигурально) от изменения размеров элементов. К градиентам можно применять ряд удобных ключевых слов, задающих размеры. Их вместо любых значений размеров можно записывать следующим образом:

```
background: radial-gradient(closest-side circle at center, #333, blue);
```

А вот то, к чему приводит применение каждого из них.

- `closest-side` — фигура распространяется до ближайшей к центру стороны блока (в случае задания круглой формы) или соответствует расстояниям до ближайших к центру горизонтальной и вертикальной сторон (в случае задания эллиптической формы).
- `closest-corner` — фигура распространяется от центра строго до ближайшего угла блока.
- `farthest-side` — в противоположность `closest-side` вместо распространения фигуры до ближайшей стороны она распространяется из центра до самой дальней стороны (или, в случае задания эллипса, до самых дальних горизонтальной и вертикальной сторон).
- `farthest-corner` — фигура распространяется из центра до самого дальнего угла блока.
- `cover` — результат идентичен применению `farthest-corner`.
- `contain` — результат идентичен применению `closest-side`.

Спецификацию W3C, относящуюся к радиальным градиентным фонам можно найти по адресу http://www.w3.org/TR/css3-images/.

ХИТРЫЙ СПОСОБ ПОЛУЧЕНИЯ ОТЛИЧНЫХ ЛИНЕЙНЫХ И РАДИАЛЬНЫХ ГРАДИЕНТОВ CSS3

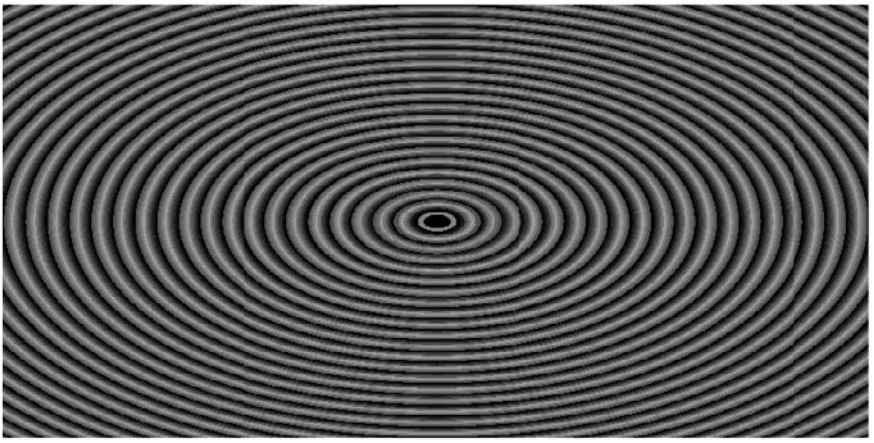
Если задание градиентов вручную представляется весьма трудоемкой работой, существует несколько интерактивных генераторов градиентов. Я предпочитаю пользоваться средством, которое можно найти по адресу http://www.colorzilla.com/gradient-editor/. В его графическом интерфейсе используется стиль графического редактора, позволяющий выбирать цвета, цветовые опорные точки, стиль градиента (поддерживаются как линейные, так и радиальные градиенты) и даже цветовое пространство (HEX, RGB(A), HSL(A)), в котором нужно задать получающийся в результате этого выбора градиент. Допускается также загружать в качестве отправных предопределенные градиенты. Если этого будет недостаточно, данное средство даже обеспечит вас дополнительным кодом для исправления недостатков Internet Explorer 9, чтобы показать градиент, и для создания альтернативного варианта со сплошным цветом для устаревших браузеров. Все еще сомневаетесь? А как насчет способности генерировать CSS-градиент на основе градиентных значений в существующем изображении? Возможно, именно этот аргумент станет для вас решающим.

Повторяющиеся градиенты

CSS3 также дает нам возможность создавать повторяющиеся градиентные фоны. Посмотрим, как это делается:

```
.repeating-radial-gradient {
    background: repeating-radial-gradient(black 0px, orange 5px, red 10px);
}
```

А вот как это выглядит на экране (чтобы не начало тошнить, долго на это смотреть не стоит).



Сначала нужно для `linear-gradient` или `radial-gradient` указать префикс `repeating`, после чего использовать тот же синтаксис, что и для обычного градиента. Здесь между черным, оранжевым и красным цветом (0, 5 и 10 пикселов соответственно) я использовал пиксельную дистанцию, но вы можете задать значения и в процентах. Для получения наилучших результатов рекомендуется использовать внутри градиента одинаковые единицы измерений (например, пикселы или проценты).

ПРИМЕЧАНИЕ

Информацию от W3C, касающуюся повторяющихся градиентов, можно получить по адресу http://www.w3.org/TR/css3-images/.

Хочу поделиться с вами еще одним способом использования градиентных фонов.

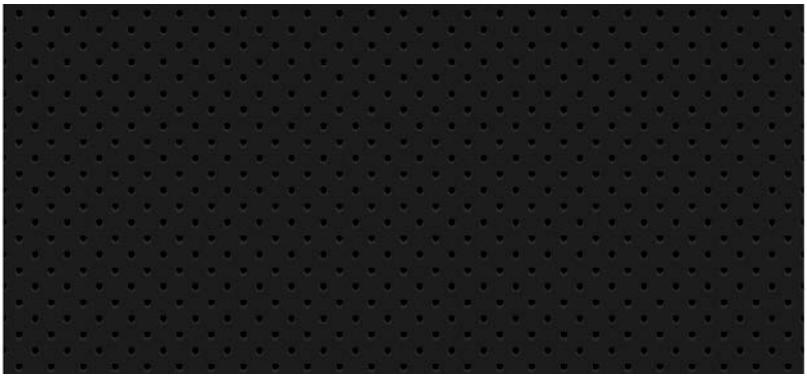
Узоры из градиентных фонов

В своих конструкциях я часто использовал едва заметные линейные градиенты и считал, что радиальные и повторяющиеся градиенты менее практичны. Но умные люди уже воспользовались эффективностью градиентов для создания узоров из градиентных фонов. Рассмотрим примеры из CSS Ninja — коллекции фоновых

узоров CSS, принадлежащей Лиа Веру (Lea Verou) и доступной на сайте http://lea.verou.me/css3patterns/:

```
.carbon-fibre {
    margin: 1rem;
    width: 400px;
    height: 200px;
    background:
    radial-gradient(black 15%, transparent 16%) 0 0,
    radial-gradient(black 15%, transparent 16%) 8px 8px,
    radial-gradient(rgba(255,255,255,.1) 15%, transparent 20%) 0 1px,
    radial-gradient(rgba(255,255,255,.1) 15%, transparent 20%) 8px 9px;
    background-color:#282828;
    background-size:16px 16px;
}
```

После запуска кода в браузере на экране будет виден эффект фона `carbon-fibre`.



Понравилось? Всего лишь несколько строк кода CSS3, и у нас есть легко редактируемый адаптивный и масштабируемый фоновый узор.

СОВЕТ

Чтобы разобраться в работе того или иного правила, попробуйте добавить в конце правила background-repeat: no-repeat.

Как всегда, благодаря медиазапросам для различных адаптивных сценариев могут использоваться разные объявления. Например, притом что в менее широких окнах просмотра вполне успешно могут работать градиентные узоры, для более крупных окон может быть лучше выводить простой фон:

```
@media (min-width: 45rem) {
    .carbon-fibre {
        background: #333;
    }
}
```

Этот пример можно увидеть, запустив файл из каталога example_06-05.

Использование нескольких фоновых изображений

Хотя теперь это немного и вышло из моды, раньше было весьма распространено требование к конструкции, заключавшееся в создании страницы, у которой верхнее фоновое изображение отличалось от нижнего. Или же в использовании различных фоновых изображений для верхнего и нижнего разделов внутри страницы. В прежние времена при использовании CSS 2.1 для получения этого эффекта обычно требовалась дополнительная разметка (один элемент нужен был для фона заголовка, а другой — для фона подвала).

Используя CSS3, в одном элементе можно накладывать друг на друга любое количество изображений.

Синтаксис имеет следующий вид:

```
.bg {
    background:
        url('../img/1.png'),
        url('../img/2.png'),
        url('../img/3.png');
}
```

Как и при накладывании друг на друга нескольких теней, изображение, стоящее в перечне первым, станет на экране браузера верхним. Если нужно, в том же самом объявлении можно добавить общий цвет фона:

```
.bg {
    background:
        url('../img/1.png'),
        url('../img/2.png'),
        url('../img/3.png') left bottom, black;
}
```

Цвет следует указать последним, и тогда он будет показан под всеми изображениями, заданными выше его.

СОВЕТ

При указании нескольких фоновых элементов вам не нужно выстраивать ссылки на различные изображения на разных строчках одну под другой, я это сделал исключительно из соображений улучшения читаемости кода.

Браузеры, не понимающие правила нескольких фонов (например, Internet Explorer 8 и более ранние версии), будут игнорировать все правило целиком, поэтому может потребоваться объявить свойство обычного фона непосредственно перед CSS3-правилом, касающимся нескольких фонов, в качестве альтернативного варианта для безнадежно устаревших браузеров.

При использовании нескольких фоновых изображений и том условии, что это PNG-файлы со свойством прозрачности, любые частично прозрачные фоновые изображения, находящиеся выше других изображений, позволят просматривать

то, что находится под ними. Но фоновые изображения не обязательно должны наслаиваться друг на друга и не обязательно должны быть одного и того же размера.

Размер фона

Чтобы установить для каждого изображения различные размеры, нужно воспользоваться свойством `background-size`. При использовании нескольких изображений синтаксис работает следующим образом:

```
.bg {
    background-size: 100% 50%, 300px 400px, auto;
}
```

Здесь в том порядке, в котором изображения перечислены в свойстве `background`, указаны значения размеров каждого изображения (сначала ширина, а затем высота), отделенные друг от друга запятыми. Как и в том примере, который приведен ранее, для каждого изображения можно использовать значения, выраженные как в процентах, так и в пикселах, и кроме этого, можно воспользоваться следующими ключевыми словами:

- `auto` — задает естественный размер элемента;
- `cover` — расширяет изображение, сохраняя соотношение его сторон, чтобы занять всю площадь элемента;
- `contain` — расширяет изображение таким образом, чтобы оно поместилось внутри элемента по своей самой длинной стороне с сохранением соотношения сторон.

Позиция фона

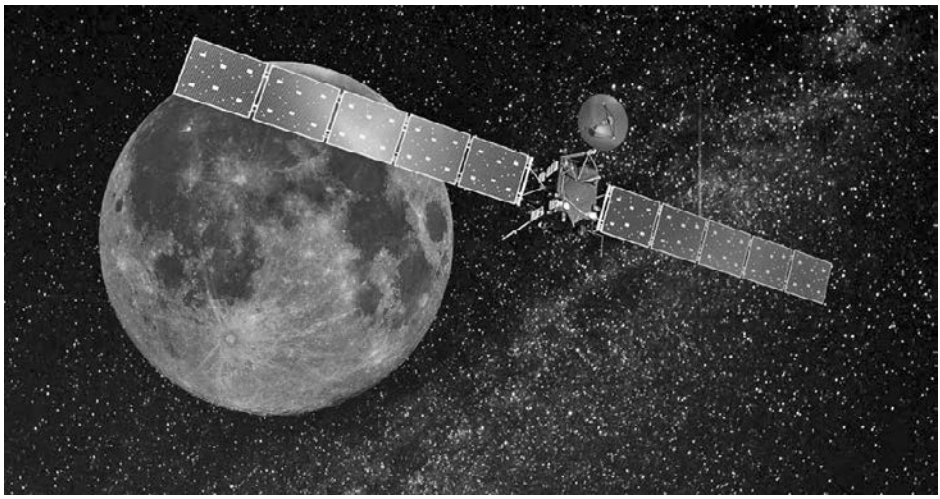
Если имеются различные фоновые изображения с отличающимися размерами, далее может потребоваться указать для них разные позиции. К счастью, с этой задачей вполне может справиться свойство `background-position`.

Объединим в одно целое все возможности указания фоновых изображений, а также воспользуемся некоторыми установками для адаптируемых конструкций, рассмотренными в предыдущих главах.

Создадим простую космическую сцену из одного элемента и трех фоновых изображений, для которых устанавливаются три различных размера и задаются совершенно разные позиции:

```
.bg-multi {
    height: 100vh;
    width: 100vw;
    background:
        url('rosetta.png'),
        url('moon.png'),
        url('stars.jpg');
    background-size: 75vmax, 50vw, cover;
    background-position: top 50px right 80px, 40px 40px, top center;
    background-repeat: no-repeat;
}
```

В браузере мы увидим картинку, похожую на эту.



В нижнем слое будет звездное небо, поверх него — луна, и, наконец, в самом верхнем слое — космический зонд «Розетта». Самостоятельно вывести картинку на экран можно, запустив файл из каталога example_06-06. Обратите внимание на то, что при изменении размеров окна браузера адаптивные единицы измерения (vmax, vh и vw) срабатывают довольно четко, сохраняя пропорции, чего нельзя сказать о тех изображениях, для которых в качестве единицы измерения использовались пикселы.

ПРИМЕЧАНИЕ

Если значения для свойства background-position не указывались, то по умолчанию применяется позиция в левом верхнем углу.

Краткий метод записи фона

Существует краткий метод записи, объединяющий различные свойства настройки фона. Соответствующую спецификацию можно найти по адресу http://www.w3.org/TR/css3-background/. Но пока что мой опыт мне подсказывает, что данный метод выдает ошибочные результаты. Поэтому я рекомендую пользоваться развернутым методом записи и сначала объявлять несколько изображений, затем размеры, а за ними — позиции.

ПРИМЕЧАНИЕ

Документацию W3C, относящуюся к нескольким фоновым элементам, можно найти по адресу http://www.w3.org/TR/css3-background/.

Фоновые изображения с высоким разрешением

Благодаря медиазапросам у нас есть возможность загружать различные фоновые изображения не только для различных размеров окон просмотра, но и для различных разрешений этих окон. Например, вот как выглядит официально утвержденный способ указания фонового изображения для нормального экрана и для экрана с высоким показателем dpi. Этот код можно найти в файле каталога example_06-07:

```
.bg {
    background-image: url('bg.jpg');
}
@media (min-resolution: 1.5dppx) {
    .bg {
        background-image: url('bg@1_5x.jpg');
    }
}
```

Медиазапрос составляется точно так же, как и раньше, с проверкой ширины, высоты или любой другой характеристики. В данном примере определен минимальный показатель разрешения, при котором должно использоваться изображение `bg@1_5x.jpg`, составляющее `1.5dppx` (количество пикселов устройства, приходящихся на один пиксел CSS). При желании можно также использовать такие единицы измерения, как dpi (количество точек на дюйм) или dpcm (количество точек на сантиметр). Я считаю, что проще всего осмыслить единицу измерения dppx, невзирая на ее более слабую поддержку, поскольку, к примеру, `2dppx` означает удвоенное разрешение, а `3dppx` — утроенное. Разобраться в разрешении, выраженном в dpi, намного сложнее. Стандартным считается разрешение `96dpi`, удвоенное разрешение выражается значением `192dpi` и т. д.

Пока поддержка единицы измерения dppx не слишком широка (ее наличие в тех браузерах, на которые рассчитано ваше устройство, можно проверить на сайте http://caniuse.com/), поэтому, чтобы все это работало повсеместно с приемлемым результатом, требуется создать несколько версий медиазапросов в отношении разрешения или положиться на средство, расставляющее нужные префиксы производителей.

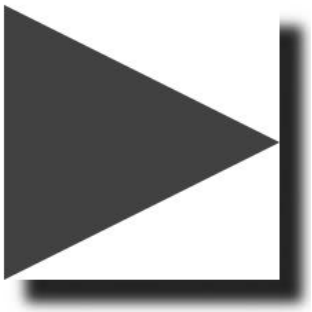
КРАТКОЕ ЗАМЕЧАНИЕ ПО ПОВОДУ ПРОИЗВОДИТЕЛЬНОСТИ

Следует помнить, что большие изображения потенциально могут снизить наблюдаемую скорость работы сайта, что приведет к ухудшению пользовательского восприятия. Хотя фоновые изображения не заблокируют вывод страницы на экран (ожидая загрузки, вы по-прежнему будете наблюдать на экране прорисовку всего остального сайта), они поспособствуют существенному общему утяжелению страницы, что немаловажно, если пользователям приходится платить за принятый объем данных.

CSS-фильтры

У свойства `box-shadow` есть одна вполне очевидная проблема. В соответствии с названием его работа ограничена принятой в CSS прямоугольной формой блока,

к которому оно применяется. Вот как выглядит копия экрана треугольника, полученного с помощью CSS, с примененной к нему тенью блока (код можно найти в каталоге example_06-08).



Я же надеялся получить другой результат. К счастью, этот недочет можно обойти с помощью CSS-фильтров, являющихся частью спецификации Filter Effects Module Level 1 (http://www.w3.org/TR/filter-effects/). Они не пользуются такой же широкой поддержкой, как box-shadow, но отлично работают при использовании подхода, предусматривающего постепенное улучшение качества вывода на экран. Если браузер не понимает, что ему нужно делать с фильтром, он его просто проигнорирует. А для тех браузеров, которые его поддерживают, на экран выводятся просто фантастические эффекты.

Вот тот же самый элемент с CSS-фильтром drop-shadow, примененным вместо box-shadow.

Формат CSS-фильтров имеет следующий вид:

```
.filter-drop-shadow {
    filter: drop-shadow(8px 8px 6px #333);
}
```

После свойства filter указывается тот фильтр, который нужно применить, в данном примере это drop-shadow, а затем передаются аргументы для фильтра. Для drop-shadow используется такой же синтаксис, как и для box-shadow, поэтому разобраться в нем довольно просто: смещение по осям *X* и *Y*, размытие, затем радиус протяженности (оба этих параметра являются необязательными) и, наконец, цвет (указывать его также не обязательно, хотя в целях совместимости я все же рекомендую от этого не отказываться).

СОВЕТ

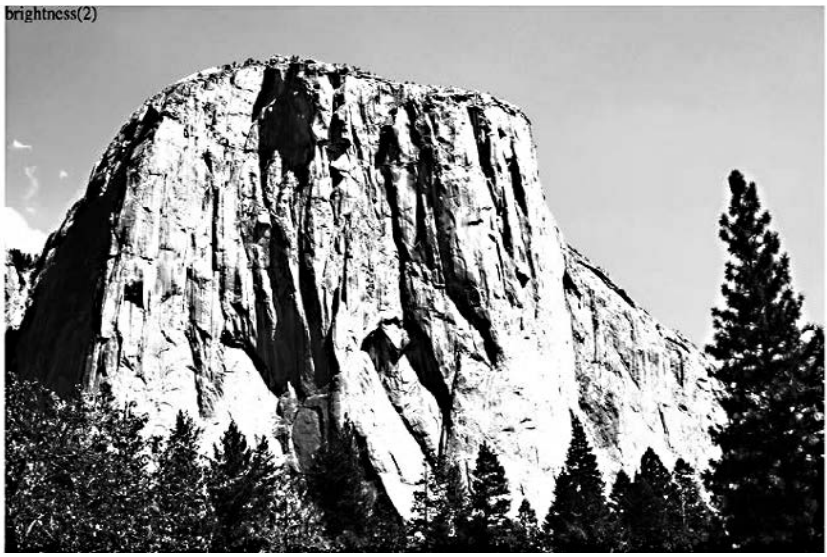
Фактически CSS-фильтры основаны на SVG-фильтрах, обладающих более широкой поддержкой. Их эквиваленты на основе SVG рассматриваются в главе 7.

Доступные CSS-фильтры

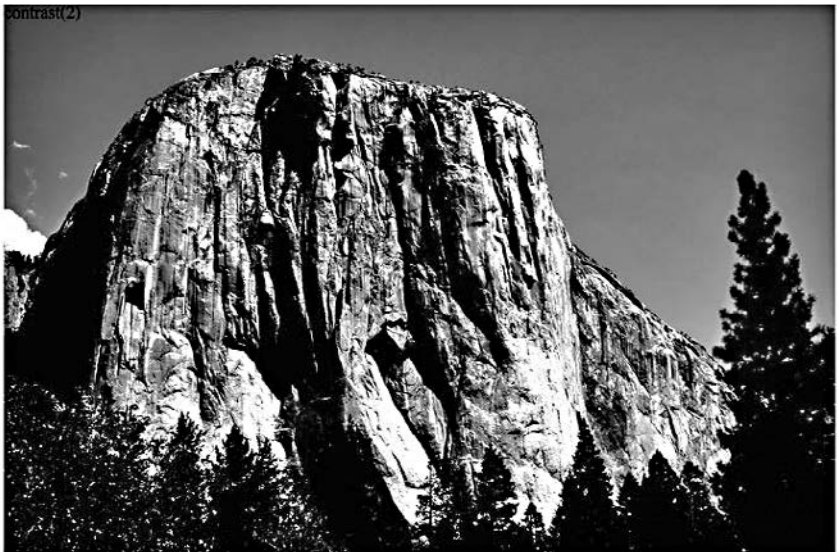
Выбирать мы можем из нескольких фильтров. Рассмотрим каждый из них. Хотя далее будут показаны изображения после применения большинства фильтров, читателям бумажной версии книги с черно-белыми картинками заметить разницу будет нелегко. Оказавшись в подобной ситуации, следует помнить, что работу различных фильтров можно по-прежнему увидеть в браузере, открыв файл из каталога example_06-08. Теперь я собираюсь перечислить все эти фильтры с указанными для них подходящими значениями. Нетрудно представить, что чем больше значение, тем выше степень применения фильтра. Там, где используются изображения, они следуют сразу же после соответствующего кода.

- `filter: url ('./img/filters.svg#filterRed')` — позволяет указать используемый SVG-фильтр.
- `filter: blur(3px)` — используется единственное значение протяженности размытия (но только не в процентном выражении).

- `filter: brightness(2)` — используются значения в диапазоне от 0 до 1 или от 0 до 100 %. Значения `0` или `0%` приводят к полному затемнению изображения, `1` или `100%` обеспечивают нормальную яркость, а более высокие значения вызывают осветление элемента.

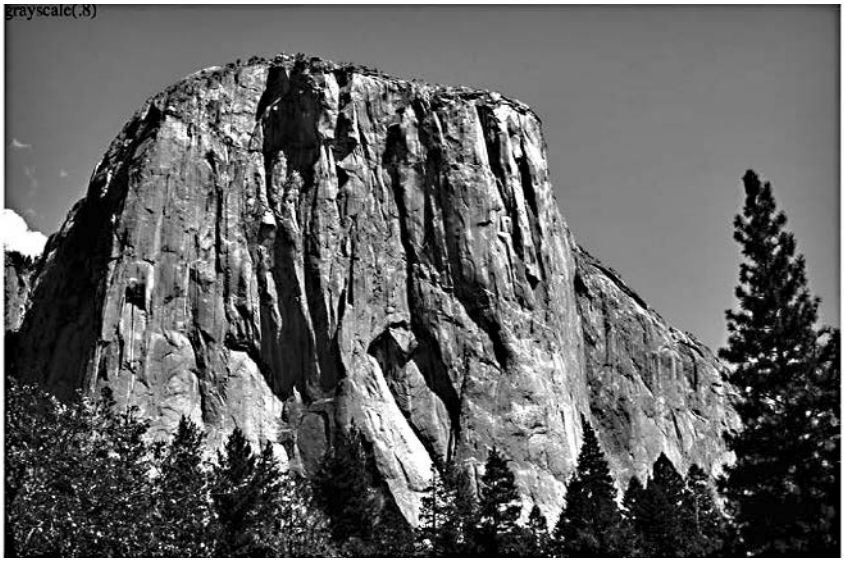


- `filter: contrast(2)` — применяются значения в диапазоне от 0 до 1 или от 0 до 100 %. Значения `0` или `0%` приводят к полному затемнению изображения, 1 или 100 % обеспечивают нормальную контрастность, а более высокие значения вызывают повышение цветовой контрастности.

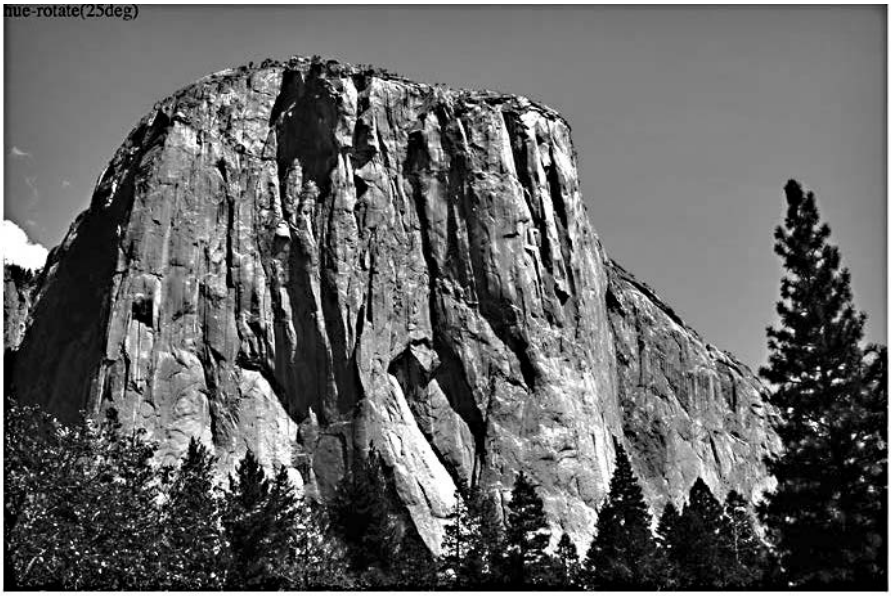


- `filter: drop-shadow(4px 4px 6px #333)` — фильтр `drop-shadow` был подробно рассмотрен ранее.
- `filter: grayscale(.8)` — используются значения от 0 до 1 или от 0 до 100 %, приводящие к различным степеням обесцвечивания элемента. Значение `0` оставля-

ет изображение полноцветным, а 1 его полностью обесцвечивает, превращая цвета в оттенки серого.

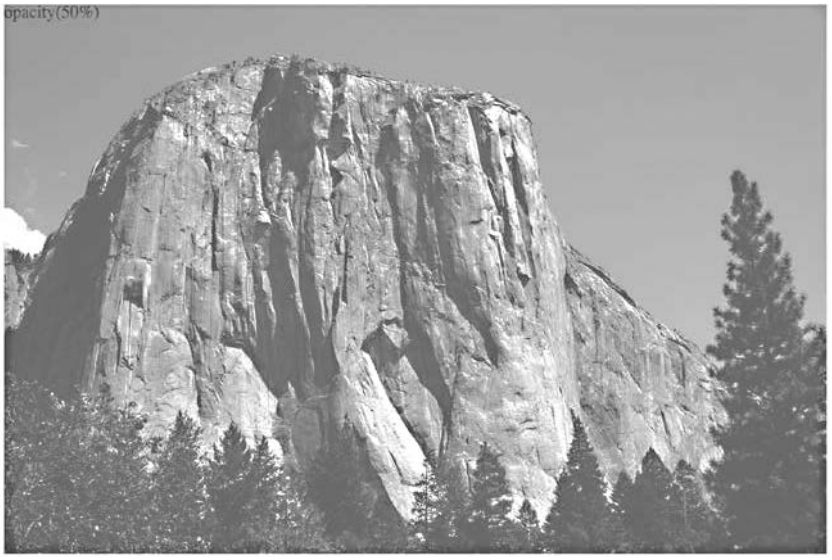


- `filter: hue-rotate(25deg)` — используются значения от 0 до 360°, вызывающие коррекцию цвета по цветовому кругу.

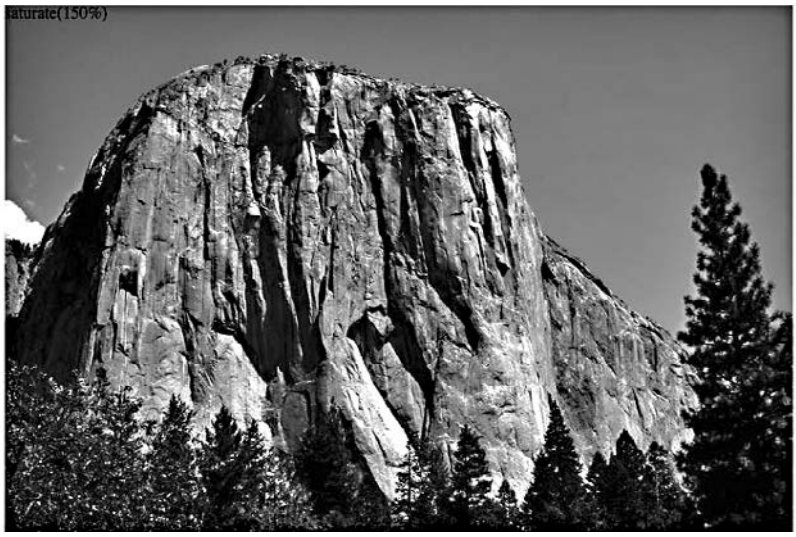


- `filter: invert(75%)` — используются значения от 0 до 1 или от 0 до 100 %, приводящие к различным степеням цветового инвертирования элемента.

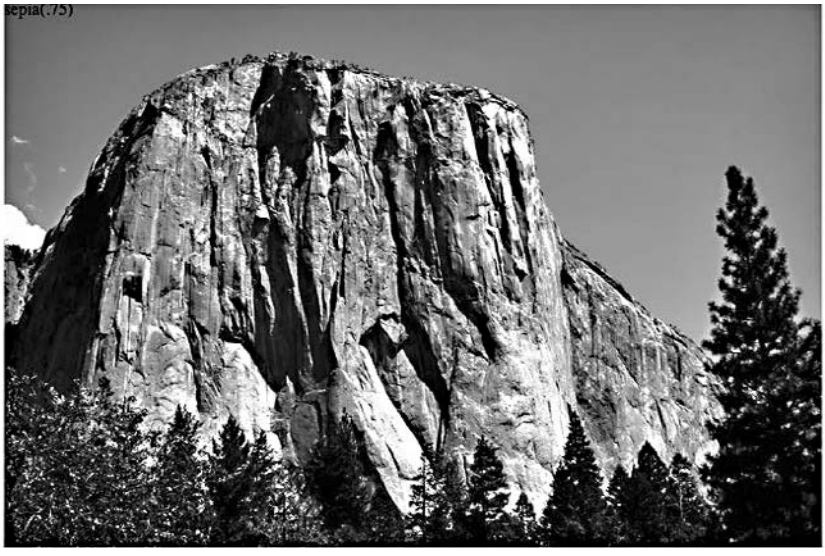
- `filter: opacity(50%)` — используются значения от 0 до 1 или от 0 до 100 %, приводящие к изменению прозрачности элемента. Этот фильтр по своему действию аналогичен уже знакомому вам свойству `opacity`. Но фильтры, в чем мы вскоре сможем убедиться, могут объединяться, позволяя применять прозрачность одновременно с другими фильтрами.



- `filter: saturate(15%)` — используются значения от 0 до 1 или от 0 до 100 %, приводящие к снижению насыщенности изображения, и значения, превышающие 1 или 100 %, вызывающие ее повышение.



- `filter: sepia(.75)` — используются значения от 0 до 1 или от 0 до 100 %, приводящие к появлению элемента в коричневых оттенках. Значение `0` или `0%` сохраняет внешний вид элемента неизменным, а все, что больше, придает ему все больше коричневых оттенков вплоть до максимума при значении `1` или `100%`.



Объединение CSS-фильтров

Фильтры можно легко объединять, просто разделяя их пробелами. Например, одновременного применения фильтров `opacity`, `blur` и `sepia` можно добиться с помощью следующего кода:

```
.MultipleFilters {
    filter: opacity(10%) blur(2px) sepia(35%);
}
```

ПРИМЕЧАНИЕ

Отрицательные значения для фильтров, за исключением hue-rotate, применять не разрешается.

Полагаю, вы не станете возражать, когда я скажу, что CSS-фильтры предлагают нам довольно мощные эффекты. Существуют также эффекты, которые от ситуации к ситуации можно подвергать превращениям и преобразованиям. Как это делается, будет показано в главе 8.

Но пока вы не стали забавляться с этими новыми игрушками, мы должны серьезно поговорить о производительности.

Предупреждения, касающиеся CSS-производительности

Когда речь заходит о CSS-производительности, мне хочется напомнить о следующем: *«Архитектура находится за скобками, а производительность — внутри»* (Бен Фрэйн).

Позвольте раскрыть смысл этого небольшого изречения. Насколько я могу судить, разговор о том, насколько быстро или медленно работает селектор CSS (являющийся той частью, которая находится за скобками), не имеет смысла. Доказательства я поместил в статье по адресу http://benfrain.com/css-performance-revisited-selectors-bloat-expensive-styles/.

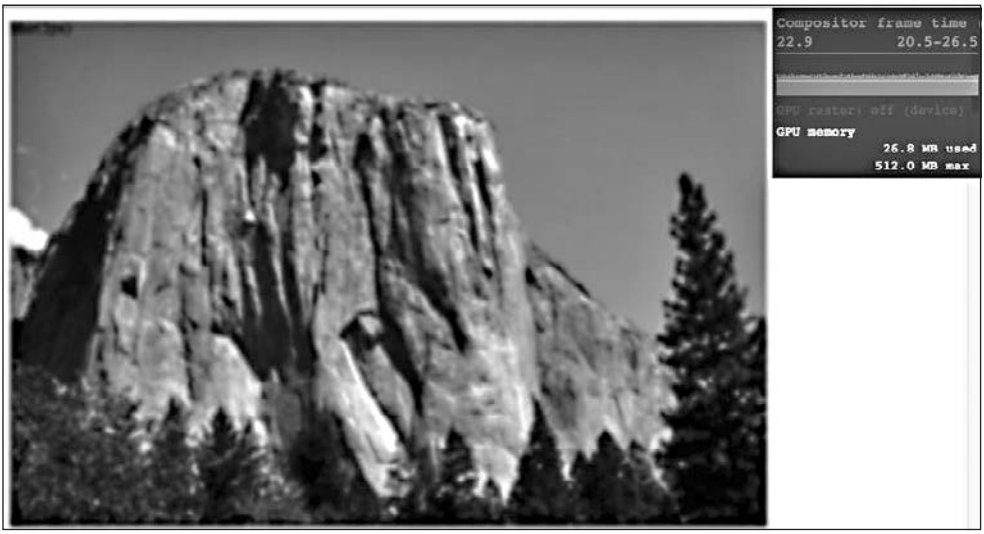
Но есть то, что действительно может довести страницу до остановки, — это использование в CSS затратных свойств (той части, которая находится внутри скобок). Когда в отношении конкретных стилей используется понятие «затратные», под этим подразумевается, что они заставляют браузер претерпевать массу издержек. Это то, что браузер считает чрезмерно обременительным для выполнения.

Чтобы догадаться о том, что, скорее всего, вызовет перегрузку браузера, можно обратиться к здравому смыслу. В основном это все, что заставит его производить предварительные вычисления, прежде чем заняться прорисовкой на экране. Сравним, к примеру, стандартный `div`-контейнер со сплошным однотонным фоном с полупрозрачным изображением в самом верхнем слое фона, составленного из нескольких градиентов, имеющего скругленные углы и отбрасываемую тень, полученную с применением фильтра `drop-shadow`. Последний из сравниваемых вариантов будет намного затратнее, вызовет существенно больший объем вычислительной работы для браузера и, соответственно, станет причиной более серьезных издержек.

Поэтому при применении таких эффектов, как фильтры, нужно проявлять расчетливость и по возможности проверять, где падает скорость работы страницы на самых маломощных устройствах, для которых вы надеетесь сохранить поддержку.

В качестве крайней меры включите такие свойства средств разработки, как постоянная перерисовка страницы в Chrome, и выключите те эффекты, которые, по-вашему, могут вызвать проблемы. Вы получите данные в миллисекундах, позволяющие судить о продолжительности прорисовки в текущем окне просмотра и принять более взвешенное решение о том, какие эффекты применять. Чем ниже показания на графике, тем быстрее будет выполняться страница (но при этом следует учитывать широкое разнообразие браузеров и платформ, поэтому, как всегда, по возможности проводите тестирование на реальных устройствах).

Для углубленного изучения данного вопроса я рекомендую следующий ресурс: https://developers.google.com/web/fundamentals/performance/rendering/.



Замечание по поводу применения в CSS масок и обрезки. В ближайшем будущем CSS в качестве части спецификации CSS Masking Module Level 1 сможет предложить маски и обрезку. Эти свойства позволят обрезать изображения с применением формы или произвольной траектории, указанной посредством SVG или ряда точек многоугольника. К сожалению, несмотря на то, что спецификация находится на наиболее проработанной стадии CR, на момент написания этих строк в браузерных реализациях было слишком много ошибок, чтобы можно было рекомендовать применение этих свойств. Но все течет, все изменяется, поэтому есть все основания предполагать, что ко времени прочтения книги появятся вполне работоспособные реализации. Всех любопытных я отсылаю к спецификации, находящейся по адресу http://www.w3.org/TR/css-masking/.

Я также считаю, что Крис Койер (Chris Coyier) проделал большую работу, разъясняя разумность поддержки этих свойств, для чего написал следующую статью: http://css-tricks.com/clipping-masking-css/.

И в завершение, неплохой обзор и объяснения будущих возможностей предлагаются в статье Сары Сьюайден (Sara Soueidan) по адресу http://alistapart.com/article/css-shapes-101.

Резюме

В этой главе была рассмотрена подборка CSS-свойств, наиболее полезных для создания эстетически привлекательных элементов в адаптивных веб-конструкциях. Мы узнали, как можно преодолеть зависимость от изображений для получения фоновых эффектов с помощью градиентных фонов CSS3. Было рассмотрено даже их использование для создания бесконечно повторяющихся фоновых узоров. Мы также изучили порядок применения текстовых теней для простого повышения привлекательности текста, а также теней для блоков, которые можно добавлять снаружи и внутри элементов. Были рассмотрены CSS-фильтры, позволяющие получить еще более впечатляющие визуальные эффекты с применением только кода CSS и допускающие объединение для получения воистину ошеломляющих результатов.

В следующей главе я собираюсь привлечь внимание к созданию и использованию масштабируемой векторной графики, или, проще говоря, SVG-графики. Несмотря на то что это весьма развитая технология с большой предысторией, она приспособлена лишь к современной обстановке высокопроизводительных адаптивных сайтов, фактически вступающих в свое совершеннолетие.

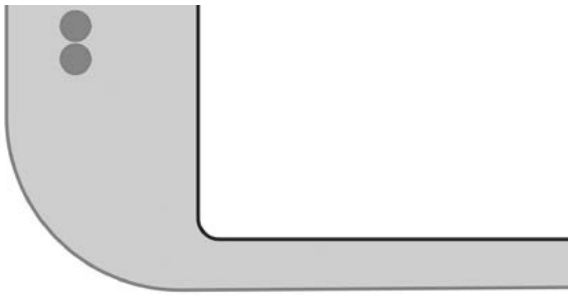
7 Использование SVG для достижения независимости от разрешения

Об SVG (сокращение от «масштабируемая векторная графика») написаны, пишутся и будут написаны целые книги. Важность технологии SVG для адаптивного веб-дизайна заключается в том, что она предоставляет опробованные графические ресурсы с четкой поточечной прорисовкой для экранов всех разрешений.

У используемых в Интернете изображений форматов JPEG, GIF или PNG визуальные данные сохраняются в виде набора пикселов. Если сохранить графику в любом из этих форматов с установкой ширины и высоты и увеличить изображение так, чтобы его исходные размеры были превышены в два и более раза, то тут же проявятся присущие этим форматам ограничения.

Копия экрана с фрагментом изображения в формате PNG, увеличенного на экране браузера, имеет следующий вид.

Заметили, что в изображении явно присутствует мозаичность? А вот в точности такая же картинка, сохраненная как векторное изображение в SVG-формате и выведенная на экран с той же степенью увеличения.



Надеюсь, разница очевидна.

Если исключить наименьшие из возможных графические ресурсы, то при использовании SVG вместо JPEG, GIF или PNG мы получим графику, независимую от разрешения и, в отличие от растровых изображений, требующую файлов намного меньших размеров.

В данной главе мы коснемся множества аспектов, присущих SVG-графике, но основное внимание будет уделено способам интеграции ее использования в ваш рабочий процесс. Кроме того, будут рассмотрены возможности, предоставляемые SVG-технологией.

Нам предстоит рассмотреть следующие вопросы:

- SVG, краткая история и анатомия простого SVG-документа;
- создание SVG-графики с помощью популярных пакетов и сервисов редактирования изображений;
- вставка объектов SVG-графики в страницу с помощью тегов `img` и `object`;
- вставка объектов SVG-графики в качестве фоновых изображений;
- вставка объектов SVG-графики непосредственно в код HTML в качестве линейных объектов;
- повторное использование SVG-символов;
- ссылка на внешние SVG-символы;
- возможности, предоставляемые каждым из методов вставки;
- анимация объектов SVG-графики с помощью SMIL;
- придание стилевого оформления объектам SVG-графики с помощью внешней таблицы стилей;
- придание стилевого оформления объектам SVG-графики с помощью внутренних стилей;
- изменение и анимация объектов SVG-графики с помощью CSS;
- медиазапросы и SVG-графика;
- оптимизация объектов SVG-графики;
- использование объектов SVG-графики с целью определения фильтров для CSS;
- манипуляция объектами SVG-графики с помощью кода JavaScript и JavaScript-библиотек;

- советы по внедрению SVG-технологии;
- ресурсы для дальнейшего изучения.

SVG-технология — предмет непростой. Какие разделы данной главы в большей степени отвечают вашим потребностям, будет зависеть от того, что конкретно вам нужно получить от SVG. Надеюсь, я сразу же смогу предложить ряд кратчайших способов ее использования.

Если вам просто нужно заменить статичные графические ресурсы на сайте их SVG-версиями с целью повышения резкости изображений и/или использования файлов меньших размеров, тогда обратитесь к менее объемным разделам по использованию SVG в качестве фоновых изображений и внутри `img`-тегов.

Если вы интересуетесь тем, какие приложения и сервисы могут помочь создать SVG-ресурсы и управлять ими, сразу переходите к разделу «Создание SVG-графики с помощью популярных пакетов и сервисов редактирования изображений», где будет дан ряд полезных ссылок и указателей.

А если желаете разобраться с SVG поглубже или управлять этой графикой, в том числе с применением анимации, то вам лучше усесться поудобнее и запастись двойной порцией любимого напитка, поскольку разговор будет долгим.

Приступая к освоению этой технологии, для начала заглянем в 2001 год.

Краткая история SVG

Первый выпуск SVG состоялся в 2001 году. И это не опечатка. SVG как технология существует с 2001 года. С тех пор она постепенно набирала обороты, но широкий интерес к ней и практическое применение связаны с появлением устройств с экранами высокого разрешения. Введение в SVG-технологию изложено в ее спецификации версии 1.1 (http://www.w3.org/TR/SVG11/intro.html) следующим образом:

> «SVG — это язык описания двухмерной графики в XML [XML10]. SVG позволяет создавать три типа графических объектов: фигуры векторной графики (например, пути, состоящие из прямых и кривых линий), изображения и текст».

Судя по названию технологии (масштабируемая векторная графика), SVG позволяет дать в коде описание двухмерных изображений в виде векторных опорных точек. Это открывает для нее широкие перспективы по созданию значков, прорисовке линий и схем.

Поскольку векторы задаются с помощью относительных описаний, их можно масштабировать без потери качества до любых размеров. Более того, так как данные SVG-графики представлены в виде описаний векторных опорных точек, их объем получается мизерным по сравнению с данными файлов JPEG, GIF или PNG, содержащими изображения сопоставимых размеров.

Кроме того, в настоящее время SVG-графика пользуется широкой поддержкой браузеров. Ее в числе прочих поддерживают Android 2.3 и следующие версии, а также версии Internet Explorer, начиная с 9-й (http://caniuse.com/#search=svg).

Графика, представляющая собой документ

Обычно при попытке просмотра кода графического файла в текстовом редакторе понять что-либо абсолютно невозможно.

А вот SVG-графика описывается языком разметки. SVG записывается на расширяемом языке разметки (Extensible Markup Language (XML)), являющемся близким родственником HTML. Возможно, вы не в курсе, но XML повсеместно используется в Интернете. Вам приходилось пользоваться RSS-каналом? А ведь в нем также используется XML. Именно в этом языке заключается содержимое RSS-канала, обеспечивая его доступность для разнообразных средств и сервисов. Следовательно, читать и понимать SVG-графику могут не только машины, но и мы с вами.

Позвольте привести пример. Посмотрите на изображение звезды.

Данное SVG-изображение можно найти в файле Star.svg, который находится в каталоге example_07-01. Файл можно открыть как в браузере, на экране которого он появится в виде звезды, так и в текстовом редакторе, где можно будет увидеть код, формирующий эту звезду. Рассмотрим код:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg width="198px" height="188px" viewBox="0 0 198 188" version="1.1"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.
w3.org/1999/xlink" xmlns:sketch="http://www.bohemiancoding.com/sketch/ns">
    <!-- Generator: Sketch 3.2.2 (9983) - http://www.bohemiancoding.
    com/sketch -->
    <title>Star 1</title>
    <desc>Created with Sketch.</desc>
    <defs></defs>
<g id="Page-1" stroke="none" stroke-width="1" fill="none" fill-
rule="evenodd" sketch:type="MSPage">
        <polygon id="Star-1" stroke="#979797" stroke-width="3"
fill="#F8E81C" sketch:type="MSShapeGroup" points="99 154 40.2214748 184.901699 51.44
1742 119.450853.894 348 3773.098 300 669.610 737 463.549 150 399 4128.389 263 63.549 150 3194
.105 652 73.098 300 6146.552 826 119.450 85 157.778 525 184.901699 "></polygon>
    </g>
</svg>
```

И это все, что нужно для создания звезды в качестве SVG-графики.

Если раньше вам никогда не приходилось видеть код SVG-графики, то у вас может возникнуть вопрос: а зачем это нужно? Если требуется всего лишь получить

отображение векторной графики на веб-странице, просматривать этот код действительно не имеет смысла. Нужно найти графическое приложение, способное сохранять результаты вашего векторного творчества в формате SVG, и вы получите то, что хотели. Перечень таких приложений будет дан чуть позже.

Это, конечно же, лишь начальный подход, позволяющий работать с SVG-графикой из графических редакторов, а вот понимание взаимодействия SVG-компонентов и освоение способов вмешательства в их совместную работу, несомненно, пригодится в том случае, когда появятся потребности в манипуляции SVG-графикой и получения на ее основе эффектов анимации.

Поэтому разберемся в SVG-разметке и постараемся понять, что происходит при ее выводе на экран. Я хочу привлечь ваше внимание к ряду ключевых моментов.

Корневой элемент SVG

У корневого SVG-элемента имеются атрибуты для ширины (`width`), высоты (`height`) и области просмотра (`viewbox`):

```
<svg width="198px" height="188px" viewBox="0 0 198 188"
```

Все это при отображении SVG на экране играет весьма важную роль.

Надеюсь, к этому времени вы уже хорошо разбираетесь в том, что такое окно просмотра. Оно использовалось в большинстве глав книги для описания той области экрана устройства, на которой просматривается содержимое. Например, у мобильного устройства может быть окно просмотра размером 320 × 480 пикселов, у настольного компьютера — 1920 × 1080 пикселов.

Фактически SVG-атрибуты `width` и `height` создают окно просмотра. Через определенное таким образом окно мы можем смотреть на фигуры, определенные внутри SVG. Как и при просмотре веб-страницы, содержимое SVG может быть больше окна просмотра, но это не означает отсутствия всего остального, оно просто не войдет целиком в текущее окно. В то же время область просмотра определяет систему координат, в которой управляют всеми фигурами SVG.

`viewbox`-значения `0 0 198 188` следует рассматривать в качестве описания левой верхней и правой нижней точек прямоугольной области. Два первых значения, известные в техническом смысле как `min-x` и `min-y`, описывают левый верхний угол, а два вторых значения, известные в техническом смысле как ширина и высота, описывают правый нижний угол.

Наличие атрибута `viewbox` позволяет увеличивать или уменьшать изображение. Например, если в нем уменьшить наполовину параметры ширины и высоты:

```
<svg width="198px" height="188px" viewBox="0 0 99 94"
```

фигура уменьшится, чтобы поместиться в размерах SVG-параметров ширины и высоты.

СОВЕТ

Чтобы подробнее разобраться с атрибутом viewbox и системой координат SVG, а также с предоставляемыми ими возможностями, я рекомендую изучить следующую статью Сары Сьюайден: http://sarasoueidan.com/blog/svg-coordinate-systems/, а также статью Якоба Женкова (Jakob Jenkov): http://tutorials.jenkov.com/svg/svg-viewport-view-box.html.

Пространство имен

У рассматриваемого кода SVG имеется дополнительное пространство имен, определенное для сгенерировавшей его графической программы Sketch (для пространства имен XML используется аббревиатура `xmlns`):

```
xmlns:sketch="http://www.bohemiancoding.com/sketch/ns"
```

Ссылки на пространство имен предназначены исключительно для использования той программой, которая генерировала SVG, поэтому, когда SVG-графика привязывается для использования в веб-приложениях, эти ссылки зачастую оказываются не нужны и поэтому удаляются в ходе оптимизации, проводимой с целью уменьшения размера SVG-графики.

Теги title и desc

Теги `title` и `desc` повышают доступность SVG-документа:

```
<title>Star 1</title>
    <desc>Created with Sketch.</desc>
```

Эти теги могут использоваться для описания содержимого графики, когда его невозможно просмотреть. Но когда SVG-графика используется в качестве фоновой, эти теги можно убрать, чтобы уменьшить размер файла.

Тег defs

В нашем примере кода имеется пустой тег `defs`:

```
<defs></defs>
```

Хотя в данном примере этот элемент остался пустым, он играет весьма важную роль и используется для хранения определений любого повторно используемого содержимого, например градиентов, символов, путей и многого другого.

Элемент g

Элемент `g` используется для создания групп других элементов. Например, при создании SVG-изображения автомобиля тег `g` использован, чтобы собрать в группу все фигуры, изображающие колесо:

```
<g id="Page-1" stroke="none" stroke-width="1" fill="none" fill-
rule="evenodd" sketch:type="MSPage">
```

В нашем `g`-теге можно увидеть повторное использование ранее определенного пространства имен редактора sketch. Оно поможет данному графическому приложению открыть эту графику еще раз, но при какой-либо привязке этого изображения никакой роли не играет.

Фигуры SVG

Самый внутренний узел в рассматриваемом примере представлен тегом многоугольника (polygon):

```
<polygon id="Star-1" stroke="#979797" stroke-width="3" fill="#F8E81C"
sketch:type="MSShapeGroup" points="99 154 40.2214748 184.901699 51.4471742 119.45085
3.89434837 73.0983006 69.6107374 63.5491503994 128.389263 63.5491503 194.105652
73.0983006 146.552826 119.45085 157.778525 184.901699 "></polygon>
```

В SVG имеется ряд уже готовых к использованию фигур (`path`, `rect`, `circle`, `ellipse`, `line`, `polyline` и `polygon`).

SVG-пути

SVG-пути отличаются от других фигур SVG, так как составляются из любого количества соединяемых точек, предоставляя вам свободу создания любой нужной фигуры.

Это важные составляющие SVG-файла, и, я надеюсь, вы хорошо себе представляете, о чем идет речь. Хотя некоторые и рады воспользоваться возможностью создания или редактирования кода SVG-файлов, все же основная масса людей склонна к созданию SVG-графики с помощью графических пакетов. Рассмотрим наиболее популярные из них.

Создание SVG-графики с помощью популярных пакетов и сервисов редактирования изображений

Хотя SVG-документы могут открываться, редактироваться и создаваться в текстовом редакторе, существует множество приложений, предлагающих графический интерфейс пользователя (Graphical User Interface (GUI)), облегчающий создание сложной SVG-графики при наличии соответствующего опыта работы по ее редактированию. Наверное, наиболее известным вариантом является использование приложения Illustrator (PC/Mac) от компании Adobe. Но для обычных пользователей у него слишком высокая цена, поэтому лично я предпочитаю приложение Sketch от Bohemian Coding (версию, предназначенную только для компьютеров Mac, можно найти по адресу http://bohemiancoding.com/sketch/). Оно само по себе недешевое (на данный момент продается по цене 99 долларов), но все же, если вы работаете на компьютере Mac, я хочу порекомендовать именно его.

Если вы пользуетесь Windows или Linux или же подыскиваете менее дорогой вариант, присмотритесь к бесплатному приложению с открытым кодом Inkscape (https://inkscape.org/en/). Конечно, это далеко не самый приятный для работы инструмент, но в нем имеются весьма впечатляющие функциональные возможности (в качестве доказательства можете просмотреть галерею Inkscape, находящуюся по адресу https://inkscape.org/en/community/gallery/).

И наконец, можно воспользоваться онлайн-редакторами. У Google есть редактор SVG-edit (http://svg-edit.googlecode.com/svn/branches/stable/editor/svg-editor.html). Есть также редакторы Draw SVG (http://www.drawsvg.org) и Method Draw, последний из которых, возможно, является наиболее интересным средством SVG-редактирования (http://editor.method.ac/).

Экономия времени с использованием сервисов SVG-значков. Все ранее упомянутые приложения позволяют создавать SVG-графику с нуля. Но если вы интересуетесь значками, то загрузка их SVG-версий с соответствующего веб-сервиса позволит сэкономить массу времени (и, полагаю, исходя из ваших талантов, приведет к получению более качественных результатов). Лично я предпочитаю http://icomoon.io/.

Если хотите получить краткое представление о преимуществах использования веб-сервиса значков, то загруженное приложение icomoon.io предоставит вам соответствующую библиотеку с возможностью поиска (часть значков будет в свободном доступе, а за остальные придется платить).

Следует выбрать нужный значок и нажать кнопку загрузки. Получаемый в результате файл содержит значки в форматах SVG, PNG, а также SVG-символы для помещения в элемент `defs` (вспомним, что это контейнер элементов, предназначенных для многократного использования).

Чтобы увидеть все это своими глазами, откройте каталог example_07-02, в котором находятся файлы, полученные в результате загрузки по окончании выбора значков с сайта http://icomoon.io/.

Вставка SVG-графики в веб-страницы

В зависимости от используемого браузера с SVG-изображениями можно делать то, чего нельзя сделать с изображениями обычных форматов (JPEG, GIF, PNG). Диа-

пазон возможностей сильно зависит от способа вставки SVG в страницу. Поэтому, прежде чем мы доберемся до возможностей SVG-графики, будут рассмотрены способы помещения ее на страницу.

Использование тега img

Наиболее простой способ использования SVG-графики совпадает со вставкой в HTML-документ любой графики. Для этого нужно воспользоваться хорошо известным тегом `img`:

```
<img src="mySconeVector.svg" alt="Amazing line art of a scone" />
```

При этом SVG-изображение ведет себя практически так же, как и любое другое изображение. И больше здесь сказать нечего.

Использование тега object

Тег `object` является контейнером, рекомендованным W3C для содержимого веб-страницы, не имеющего отношения к HTML (спецификацию на `object` можно найти по адресу http://www.w3.org/TR/html5/embedded-content-0.html). Мы можем воспользоваться им для вставки SVG-графики на нашу страницу:

```
<object data="img/svgfile.svg" type="image/svg+xml">
    <span class="fallback-info">Your browser doesn't support SVG</span>
</object>
```

Этому тегу требуется либо атрибут `data`, либо атрибут `type`, хотя я всегда буду рекомендовать добавление обоих атрибутов. С помощью атрибута `data` внешняя ссылка на SVG-ресурс осуществляется точно так же, как ссылка на любой другой ресурс. С помощью атрибута `type` дается описание MIME-типа (медиатипа Интернета), соответствующего содержимому. В данном случае `image/svg+xml` является MIME-типом, обозначающим данные в формате SVG. Если нужно ограничить размер SVG в пределах контейнера, можно также добавить атрибуты `width` и `height`.

SVG-графика, вставленная на страницу с помощью тега `object`, доступна также коду JavaScript, что является одним из аргументов в пользу этого способа вставки. Но дополнительным преимуществом использования тега `object` является предоставление им простого механизма действия в случае, если браузер не понимает типа данных. Например, если бы предыдущий элемент `object` просматривался в Internet Explorer 8 (в котором отсутствует поддержка SVG), то на экране просто было бы показано сообщение о том, что браузер не поддерживает SVG. Пространство, в котором обозначено сообщение, можно использовать для предоставления альтернативного изображения в теге `img`. Но при этом следует иметь в виду, что, по результатам моего экспресс-тестирования, браузер всегда будет загружать альтернативное изображение независимо от практической надобности в нем. Поэтому, если вы желаете, чтобы сайт загружался как можно быстрее (а я не сомневаюсь, что вы именно этого и хотите), данный вариант, наверное, будет не самым лучшим выходом из положения.

ПРИМЕЧАНИЕ

Если нужно работать с SVG-графикой, вставленной с помощью тега object, с использованием jQuery, то вам следует воспользоваться имеющимся в JavaScript свойством .contentDocument. Затем для внесения изменений, например для заполнения, можно будет воспользоваться jQuery-методом .attr.

Альтернативным вариантом может стать добавление фонового изображения с помощью CSS-свойства `background-image`. Например, в показанном ранее коде у нашего альтернативного `span`-контейнера имеется класс `.fallback-info`. Мы можем этим воспользоваться в CSS для ссылки на подходящее фоновое изображение. Тогда это изображение будет загружаться только в случае реальной надобности.

Вставка SVG-графики в качестве фонового изображения

SVG-графика может использоваться в качестве фонового изображения в CSS примерно так же, как и изображение любого другого формата (PNG, JPG, GIF). В способе ссылки на него нет ничего необычного:

```
.item {
    background-image: url('image.svg');
}
```

Для устаревших браузеров, не поддерживающих SVG, может понадобиться включить альтернативное изображение в более широко поддерживаемом формате (обычно PNG). Одно из решений для Internet Explorer 8 и Android 2 (IE8 не поддерживает SVG или `background-size`, а Android 2.3 не поддерживает SVG и требует для `background-size` префикса производителя) выглядит следующим образом:

```
.item {
    background: url('image.png') no-repeat;
    background: url('image.svg') left top / auto auto no-repeat;
}
```

Когда в коде CSS применяются два одинаковых свойства, нижестоящим свойством всегда переписывается то свойство, что стоит выше его. В правиле CSS браузер всегда будет игнорировать ту пару «свойство — значение», в которой он не может разобраться. В данном случае устаревшие браузеры получат изображение в формате PNG, поскольку они не могут воспользоваться SVG или разобраться в лишенном префикса свойстве `background-size`, а более новые браузеры, которые могут воспользоваться и тем и другим, возьмут самое нижнее свойство, так как им отменяется первое свойство.

Альтернативные варианты можно предоставить с помощью JavaScript-средства Modernizr, предназначенного для тестирования функциональных возможностей браузера (более подробно Modernizr рассматривается в главе 5). В Modernizr содержатся отдельные тесты для различных методов вставки SVG-графики, а в следующей версии Modernizr (которая на момент написания книги еще не вышла)

может содержаться что-нибудь более специфичное для SVG в CSS. Но пока вы можете сделать следующее:

```
.item {
    background-image: url('image.png');
}
.svg .item {
    background-image: url('image.svg');
}
```

Или инвертировать логику, если вам так удобнее:

```
.item {
    background-image: url('image.svg');
}
.no-svg .item {
    background-image: url('image.png');
}
```

Когда запросы возможностей (feature queries) получат более полную поддержку, можно будет также сделать следующее:

```
.item {
    background-image: url('image.png');
}

@supports (fill: black) {
    .item {
        background-image: url('image.svg');
    }
}
```

Правило `@supports` здесь сработает, поскольку `fill` является свойством SVG, и если браузер это понимает, то он предпочтет не верхнее, а нижнее правило.

Если ваши потребности в SVG-графике сводятся в основном к статичным фоновым изображениям и, возможно, значкам и т. п., я настоятельно рекомендую внедрять SVG-графику в качестве фоновых изображений. Для этого существует множество инструментальных средств, автоматически создающих спрайты изображений или ссылки на ресурсы в таблицах стилей (что означает включение SVG-графики в виде URI-идентификатора данных), альтернативные PNG-ресурсы и необходимые таблицы стилей из отдельных создаваемых вами SVG-изображений. Этот способ использования SVG-графики пользуется весьма широкой поддержкой, сами изображения неплохо кэшируются (следовательно, их использование приносит немалую выгоду с точки зрения производительности), и все это довольно просто реализовать.

Краткое отступление по поводу URI-идентификаторов данных

Если при чтении предыдущего раздела у вас возник вопрос о том, что такое данные, на которые указывает унифицированный идентификатор ресурса (Uniform Resource Identifier (URI)), то по отношению к CSS он означает включение того, что обычно

должно быть внешним ресурсом, например изображения, в сам CSS-файл. Поэтому мы можем сделать это, сославшись на внешний файл изображения:

```
.external {
    background-image: url('Star.svg');
}
```

Можно просто включить изображение в состав нашей таблицы стилей, воспользовавшись URI-идентификатором данных:

```
.data-uri {
    background-image: url(data:image/svg+xml,%3C%3Fxml%20
version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standalone%3D%22
no%22%3F%3E%0A%3Csvg%20width%3D%22198px%22%20height%3D%22188px-
%22%20viewBox%3D%220%200%20198%20188%22%20version%3D%221.1%22%20
xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%20xmlns%3Axlink
%3D%22http%3A%2F%2Fwww.w3.org%2F1999%2Fxlink%22%20xmlns%3Asketch%3
D%22http%3A%2F%2Fwww.bohemiancoding.com%2Fsketch%2Fns%22%3E%0A%20
%20%20%20%3C%21--%20Generator%3A%20Sketch%203.2.2%20%289983%29%20
-%20http%3A%2F%2Fwww.bohemiancoding.com%2Fsketch%20--%3E%0A%20
%20%20%20%3Ctitle%3EStar%201%3C%2Ftitle%3E%0A%20%20%20%20
%3Cdesc%3ECreated%20with%20Sketch.%3C%2Fdesc%3E%0A%20%20%20%20-
%3Cdefs%3E%3C%2Fdefs%3E%0A%20%20%20%20%3Cg%20id%3D%22Page-1%22%20
stroke%3D%22none%22%20stroke-width%3D%221%22%20fill%3D%22none%22%20
fill-rule%3D%22evenodd%22%20sketch%3Atype%3D%22MSPage%22%3E%
0A%20%20%20%20%20%20%20%20%3Cpolygon%20id%3D%22Star-1%22%20
stroke%3D%22%23979797%22%20stroke-width%3D%223%22%20
fill%3D%22%23F8E81C%22%20sketch%3Atype%3D%22MSShapeGroup%22%20
points%3D%2299%20154%2040.2214748%20184.901699%2051.4471742%20119.45085%20
3.89434837%2073.0983006%2069.6107374%2063.5491503%2099%204%20128.389263%20
63.5491503%20194.105652%2073.0983006%20146.552826%20119.45085%20157.778525%20
184.901699%20%22%3E%3C%2Fpolygon%3E%0A%20%20
%20%20%3C%2Fg%3E%0A%3C%2Fsvg%3E);
}
```

Выглядит все это не очень привлекательно, но зато обеспечивает способ избавления от отдельного запроса по сети. Существуют различные методы кодирования URI-идентификаторов данных, и имеется множество средств, доступных для создания таких идентификаторов из ваших ресурсов.

Если кодировать SVG-графику таким вот образом, то я посоветую не пользоваться методом base64, поскольку SVG-содержимое не сжимается им так же хорошо, как текст.

Создание спрайтов изображений

Лично я рекомендую для создания спрайтов изображений или ресурсов, состоящих из URI-идентификаторов данных, использовать средство Iconizr (http://iconizr.com/). Оно обеспечивает полный контроль над внешним видом получаемого в конечном итоге SVG- и альтернативного PNG-ресурса. Можно вывести SVG-графику и альтернативные PNG-файлы в виде URI-идентификаторов данных или спрайтов изо-

бражений и даже включить туда фрагмент кода JavaScript для загрузки подходящего ресурса, если выбор будет сделан в пользу URI-идентификаторов данных. Настоятельно советую воспользоваться именно этим средством.

Если вы колеблетесь, не зная, что выбрать для своих проектов — URI-идентификаторы данных или спрайты изображений, то вас могут заинтересовать дополнительные исследования, которые я провел, чтобы взвесить все «за» и «против» обоих вариантов: http://benfrain.com/image-sprites-data-uris-icon-fonts-v-svgs/.

Лично я сторонник использования SVG-графики в качестве фоновых изображений, но если вам потребуется получить ее динамическую анимацию или ввести в нее с помощью JavaScript какие-либо значения, лучше будет выбрать вставку SVG-данных непосредственно в код HTML.

Непосредственная вставка SVG

Поскольку SVG-графика — это просто XML-документ, ее можно вставить непосредственно в код HTML, например:

```
<div>
    <h3>Inserted 'inline':</h3>
    <span class="inlineSVG">
      <svg id="svgInline" width="198" height="188" viewBox="00198188" xmlns="http://
www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
        <title>Star 1</title>
            <g class="star_Wrapper" fill="none" fill-rule="evenodd">
                <path id="star_Path" stroke="#979797" stroke-
width="3" fill="#F8E81C" d="M991541-58.7830.90211.227-65.45L3.89473.097165.717-
9.55L994129.3959.5565.7169.548-47.55346.35311.22665.452z" />
            </g>
        </svg>
    </span>
</div>
```

Элемент не нуждается в каком-либо особом контейнере, SVG-разметка буквально вставляется в разметку HTML. Полезно будет также знать, что при удалении из элемента `svg` любых атрибутов `width` и `height` SVG-графика будет подогнана под размер того элемента, в котором она содержится.

Вставка SVG-графики в ваш документ — наверное, самый универсальный способ реализации возможностей технологии SVG.

Повторное использование графических объектов из символов

Ранее в главе я уже упомянул о том, что выбрал и загрузил ряд значков с сайта http://icomoon.io. Это были значки, изображающие жестикуляцию, используемую при работе на сенсорных устройствах: скольжение, движение двумя пальцами в разные стороны и т. д. Предположим, на создаваемом вами сайте нужно использовать эти

значки не один раз. Помните, я говорил, что была версия этих значков в виде определений SVG-символов? Именно она нам сейчас и пригодится.

В код, который находится в файле каталога example_07-09, мы будем вставлять определения различных символов в имеющийся на странице `defs`-элемент SVG-графики. Следует заметить, что в отношении SVG-элемента использовано встроенное в код стилевое оформление `display:none`, а для атрибутов `height` и `width` установлены нулевые значения (эти же стилевые настройки при желании могут быть установлены и в таблице CSS). По этой причине SVG-графика не занимает никакого пространства. Этот SVG-элемент используется только для размещения в нем символов графических объектов, которые мы собираемся использовать в других местах.

Итак, наша разметка начинается со следующего фрагмента кода:

```
<body>
    <svg display="none" width="0" height="0" version="1.1"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.
w3.org/1999/xlink">
    <defs>
    <symbol id="icon-drag-left-right" viewBox="0013441024">
        <title>drag-left-right</title>
        <path class="path1" d="M256192v-160l-224224224224v-160h256v-128z"></path>
```

Заметили, что внутри элемента `defs` находится элемент `symbol`? Он предназначен для использования в том случае, если нам понадобится определить фигуру для последующего повторного применения.

После того как SVG определяет все необходимые для нашей работы символы, у нас имеется вся обычная HTML-разметка. Затем, когда нужно будет воспользоваться одним из таких символов, можно сделать следующее:

```
<svg class="icon-drag-left-right">
    <use xlink:href="#icon-drag-left-right"></use>
</svg>
```

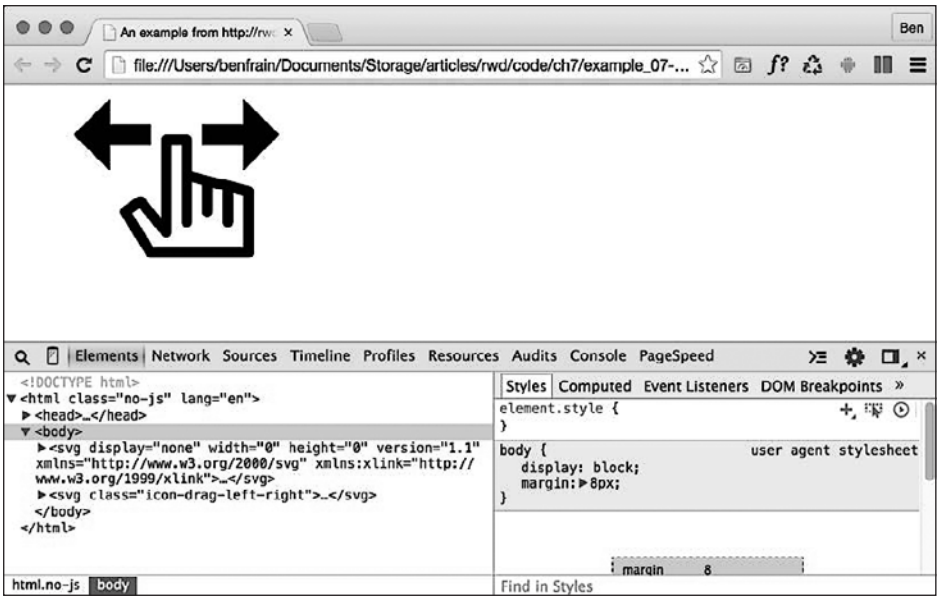
В результате появится значок перемещения влево-вправо.

Вся магия заключается в элементе `use`. Исходя из его имени, нетрудно догадаться, что он применяется для использования существующих графических объектов, которые уже были где-то определены. Механизм конкретной ссылки заложен в атрибуте `xlink`, который в данном случае ссылается на идентификатор символа значка перемещения влево-вправо (`#icon-drag-left-right`), вставленного в код в самом начале разметки.

При повторном использовании символа, пока размер не будет указан явно (с помощью либо атрибутов самого элемента, либо CSS), для элемента `use` будут установлены ширина и высота, равные 100 %. Следовательно, для изменения размеров значка можно сделать следующее:

```
.icon-drag-left-right {
    width: 2.5rem;
    height: 2.5rem;
}
```

Элемент `use` может применяться для повторного использования любого SVG-содержимого: градиентов, фигур, символов и т. д.



Встраиваемая в код SVG-графика позволяет задавать различные цвета в разных контекстах

Используя встраиваемые непосредственно в код страницы SVG-данные, можно добиться и других полезных результатов, например изменить цвет на основе контекста, что особенно пригодится при необходимости иметь несколько версий одного и того же значка в различных цветовых решениях:

```
.icon-drag-left-right {
    fill: #f90;
}

.different-context .icon-drag-left-right {
    fill: #ddd;
}
```

Создание двухтональных значков, наследующих цвет от своих родителей. Со встроенной в код SVG-графикой также можно поразвлечься и создать двухтональные эффекты из одноцветного значка (поскольку SVG-значки создаются более чем из одного пути), используя для этого `currentColor` — одну из самых старых переменных CSS. Для этого нужно внутри SVG-символа установить для атрибута заливки (`fill`) того пути, для которого предусматривается неопределенный цвет, значение `currentColor`. Затем использовать цветовое значение из CSS-таблицы для заливки элемента. Значение заливки в SVG-символе получат те пути, у которых не имеется атрибута заливки со значением `currentColor`. В качестве иллюстрации:

```
.icon-drag-left-right {
    width: 2.5rem;
```

```
    height: 2.5rem;
    fill: #f90;
    color: #ccc; /* будет применено к тому пути символа,
    у которого для атрибута fill установлено
    значение currentColor */
}
```

Вот как выглядит один и тот же символ, использованный три раза, всякий раз с другими цветовыми настройками и размерами.

Не забудьте, что вы можете покопаться в коде, найдя его в файле каталога example_07-09. Не помешает также знать, что присваивать цвет самому элементу совсем не обязательно, он может задаваться в любом родительском элементе, а `currentColor` унаследует значение из верхнего элемента DOM-дерева, являющегося ближайшим прародителем с установленным значением цвета.

Этот способ использования SVG-графики имеет массу преимуществ. Единственным недостатком является то, что одни и те же SVG-данные следует включать в каждую страницу, на которой нужно использовать значки. К сожалению, это отрицательно влияет на производительность, поскольку ресурсы (SVG-данные) трудно поддаются кэшированию. Но есть и другой вариант (если вы согласитесь добавить сценарий для поддержки Internet Explorer).

Повторное использование графических объектов из внешних источников

Вместо вставки неимоверного количества SVG-символов в каждую страницу можно, воспользовавшись тем же элементом `use`, создать ссылку на внешние SVG-файлы и получить часть документа, которую нужно использовать. Посмотрите на файл из каталога example_07-10 и обратите внимание на то, как те же три значка, которые были у нас в каталоге example_07-09, помещены на страницу:

```
<svg class="icon-drag-left-right">
    <use xlink:href="defs.svg#icon-drag-left-right"></use>
</svg>
```

Самой важной частью, с которой следует разобраться, является значение `href`. Мы ссылаемся на внешний SVG-файл (обозначен в значении как `defs.svg`), а затем внутри этого файла указываем на идентификатор символа, которым собираемся воспользоваться (обозначен в значении как `#icon-drag-left-right`).

Преимущества такого подхода состоят в том, что ресурс кэшируется браузером (как это должно или может произойти с любым другим внешним изображением), а наша разметка не засоряется полными определениями SVG-символов. Недоста-

ток состоит в том, что, в отличие от вставки `defs` в код страницы, любые динамические изменения, вносимые в defs.svg (например, при манипулировании каким-либо путем с помощью кода JavaScript), в тегах `use` обновляться не будут.

К сожалению, Internet Explorer не позволяет пользоваться ссылками на символы из внешних ресурсов. Но есть сценарий-полифилл для IE9-11 под названием SVG For Everybody, который все равно позволит нам воспользоваться данной технологией. Более подробную информацию можно найти по адресу https://github.com/jonathantneal/svg4everybody.

При использовании этого кода JavaScript вы сможете вполне успешно ссылаться на внешние ресурсы, и полифилл для Internet Explorer вставит данные непосредственно в тело документа.

Что можно делать с любым методом вставки SVG-данных

Как уже упоминалось, SVG-ресурсы отличаются от других графических ресурсов. Они могут вести себя по-разному в зависимости от способа их вставки в страницу. Как мы уже видели, есть четыре способа помещения SVG-графики на страницу:

- внутри тега `img`;
- внутри тега `object`;
- в качестве фонового изображения;
- путем вставки непосредственно в код.

И в зависимости от метода вставки вам будут доступны или недоступны конкретные возможности.

Проще, наверное, будет разобраться с возможностями, предоставляемыми при использовании каждого из методов вставки, изучая следующую таблицу.

Свойство	**img**	**object**	**Вставка непосредственно в код страницы**	**background-image**
SMIL	Да	Да	Да	Да
Внешняя CSS-таблица	Нет	См. примечание 1	Да	Нет
Внутренний CSS-код	Да	Да	Да	Да
Доступ к коду JavaScript	Нет	Да	Да	Нет
Кэшируемость	Да	Да	См. примечание 2	Да
Медиазапрос в SVG	Да	Да	См. примечание 3	Да
Повторное использование	Нет	Да	Да	Нет

Примечания

1. При использовании SVG внутри объекта внешнюю таблицу стилей для стилевого оформления SVG использовать можно, но ссылку на эту таблицу нужно сделать из кода SVG.

2. У вас есть возможность использования SVG-графики, находящейся на внешнем ресурсе (при этом она подлежит кэшированию), но изначально ссылка на нее в Internet Explorer не работает.
3. Медиазапрос внутри раздела стилей SVG-графики, вставленной непосредственно в код страницы, работает в отношении размера того документа, в котором он находится, а не в отношении размера самой SVG-графики.

Особенности браузеров. Следует знать, что браузерная реализация SVG-графики также не может похвастаться однообразием. Поэтому только то, что те или иные свойства согласно приведенной ранее таблице должны быть доступны, еще не означает, что они реализованы в каждом браузере или что все они работают одинаково!

Например, результаты, показанные в предыдущей таблице, основаны на тестировании страницы, которая находится в файле каталога example_07-03.

Поведение тестовой страницы в самых последних версиях Firefox, Chrome и Safari сравнительно одинаковое. А вот Internet Explorer порой работает несколько иначе.

Например, во всех версиях Internet Explorer, совместимых с SVG-графикой (на данный момент это 9, 10 и 11-я версии), как мы уже видели, отсутствует возможность ссылки на внешние SVG-источники. Далее, Internet Explorer применяет стили из внешних таблиц стилей к SVG-объектам вне зависимости от того способа, с помощью которого они были вставлены (все остальные браузеры применяют стили из внешних таблиц стилей, только если SVG-объекты были вставлены посредством тега `object` или же непосредственно в код страницы). Internet Explorer также не позволяет применять к SVG анимационные эффекты посредством кода CSS, поэтому анимацию SVG в Internet Explorer приходится реализовывать с применением кода JavaScript. И еще раз для тех, кто в танке: вы не сможете анимировать SVG-графику в Internet Explorer ничем другим, кроме кода JavaScript.

Дополнительные возможности и особенности технологии SVG

Давайте пока оставим в покое недостатки браузеров и рассмотрим некоторые из перечисленных в таблице и фактически доступных свойств, а также выясним, зачем их применять, или причины, по которым от их применения следует отказаться.

SVG-графика всегда будет выводиться на экран с максимально допустимой для устройства резкостью независимо от способа ее вставки на страницу. Обычно в большинстве практических ситуаций вполне достаточным основанием для применения SVG-графики является независимость ее качества от разрешения экрана. Остается только решить вопрос, какой именно метод ее вставки на страницу наиболее приемлем для вашего рабочего процесса и решаемой задачи.

Но есть и другие возможности и особенности, о которых стоит узнать, например SMIL-анимация, иные способы ссылок на внешние таблицы стилей, пометка внутренних стилей с помощью разделителей символьных данных, усовершенствование SVG-технологии с помощью JavaScript и использование медиазапросов внутри SVG-документов. Приступим к их изучению.

SMIL-анимация

SMIL-анимация (http://www.w3.org/TR/smil-animation/) является способом определения анимации для SVG-графики внутри самого SVG-документа.

SMIL (если интересно, произносится как smile, то есть «смайл», что созвучно со словом, означающим улыбку) расшифровывается как язык разметки для создания интерактивных мультимедийных презентаций (Synchronized Multimedia Integration Language), он был разработан в качестве способа определения анимации внутри XML-документа (вспомним, что в SVG-документах используется XML).

Рассмотрим пример определения анимации на основе SMIL:

```
<g class="star_Wrapper" fill="none" fill-rule="evenodd">
    <animate xlink:href="#star_Path" attributeName="fill"
attributeType="XML" begin="0s" dur="2s" fill="freeze" from="#F8E81C"
to="#14805e" />

    <path id="star_Path" stroke="#979797" stroke-width="3"
fill="#F8E81C"  d="M99154l-58.7830.90211.227-65.45L3.89473.097l65.717-
9.55L994l29.3959.5565.7169.548-47.55346.35311.22665.452z" />
</g>
```

Я воспользовался разделом из ранее рассмотренного нами SVG-документа. Тег `g` обозначает в SVG группирующий элемент, а в данный элемент включается как фигура звезды (путевой элемент с `id="star_Path"`), так и SMIL-анимация внутри элемента `animate`. Эта простая анимация меняет цвет заполнения звезды с желтого на зеленый (проводит операцию под названием tweening) в течение двух секунд. Кроме того, это делается, когда SVG-графика вставляется в страницу с помощью тегов `img`, `object`, указания в свойстве `background-image` или непосредственно в сам код страницы (если честно, то пример в файле каталога example_07-03 можно посмотреть в любом современном браузере, кроме Internet Explorer).

TWEENING

Если вы еще не знаете, что такое tweening (я вот не знал), то это термин, являющийся простым сокращением от inbetweening (переходное состояние между двумя позициями), обозначающим все находящееся в промежуточном положении между стадиями от одной точки анимации к другой.

Правда, здорово получилось? Да, вот так все и могло бы быть. Но, несмотря на то что данная технология некоторое время даже рассматривалась как стандарт, дни ее, похоже, сочтены.

Конец улыбки SMIL. У SMIL нет поддержки в Internet Explorer. Никакой. Ни малейшей. И все. Пшик. Я могу продолжить сокрушаться, но, полагаю, вы поняли, что на данный момент SMIL не имеет никакой поддержки в Internet Explorer.

А что еще хуже, в Microsoft даже не планируют когда-либо ее вводить. Состоянием дел можно поинтересоваться на сайте https://status.modern.ie/svgsmilanimation?term=SMIL.

Вдобавок в Chrome стали просматриваться тенденции к запрещению SMIL: https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/5o0yiO440LM.

Занавес.

ПРИМЕЧАНИЕ

Если вы все же испытываете потребность в использовании SMIL, то у Сары Сьюайден (Sara Soueidan) есть превосходная и очень подробная статья о SMIL-анимации по адресу http://css-tricks.com/guide-svg-animations-smil/.

К счастью, существует масса других способов применения эффектов анимации к SVG-графике, которые мы вскоре рассмотрим. Поэтому, если вам приходится поддерживать Internet Explorer, положитесь именно на них.

Задание стилей SVG с помощью внешней таблицы стилей

Придать стиль SVG-документу можно с помощью кода CSS. Это может быть CSS-код, заключенный в сам SVG-документ, или же таблицы стилей CSS, куда можно записать весь ваш обычный код CSS.

Теперь, если вернуться к ранее показанной таблице свойств, вы увидите, что стилевое оформление SVG с помощью внешней CSS-таблицы невозможно, когда SVG-документ включен с использованием тега `img` или в качестве значения свойства `background-image` (за исключением Internet Explorer). Такая возможность появляется только тогда, когда SVG-графика вставляется с помощью тега `object` или непосредственно в код страницы.

Для создания ссылки на внешнюю таблицу стилей из SVG-документа существует два варианта синтаксиса. Самый простой способ заключается в применении следующего кода (который обычно добавляется в раздел `defs`):

```
<link href="styles.css" type="text/css" rel="stylesheet"/>
```

Это похоже на то, как мы использовали ссылку на таблицу стилей до выхода HTML5. (Обратите, к примеру, внимание на атрибут `type`, который в HTML5 уже не нужен.) Но, несмотря на то что этот способ работает во многих браузерах, он не упоминается в спецификации, определяющей способы ссылки SVG-документа на внешние таблицы стилей (http://www.w3.org/TR/SVG/styling.html).

А вот как выглядит правильный или официальный способ, фактически определенный для XML в далеком 1999 году (http://www.w3.org/1999/06/REC-xml-stylesheet-19990629/):

```
<?xml-stylesheet href="styles.css" type="text/css"?>
```

Эта строка кода должна быть в вашем файле выше открывающего SVG-элемента, например:

```
<?xml-stylesheet href="styles.css" type="text/css"?>
<svg width="198" height="188" viewBox="00198188" xmlns="http://www.
w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
```

Интересно, что последний вариант синтаксиса является единственным работающим в Internet Explorer. Поэтому, если нужна ссылка на внешнюю таблицу стилей из вашего SVG-документа, то для более широкой поддержки я рекомендую использовать второй вариант синтаксиса.

Использовать внешнюю таблицу стилей совсем не обязательно, если хотите, можете воспользоваться стилями, непосредственно встроенными в сам SVG.

Задание стилей SVG с помощью внутренних стилей

Стили для SVG можно поместить в сам SVG-документ в элементе `defs`. Поскольку структура SVG-документа основана на XML, надежнее будет включить маркер Character Data (CDATA). Этот маркер просто сообщает браузеру, что информация, находящаяся внутри ограниченного раздела символьных данных, может быть истолкована как XML-разметка, но не должна толковаться в этом качестве. Используемый для этого синтаксис выглядит следующим образом:

```
<defs>
    <style type="text/css">
        <![CDATA[
            #star_Path {
                stroke: red;
            }
        ]]>
    </style>
</defs>
```

Свойства и значения SVG внутри CSS. Обратите внимание на свойство `stroke` в предыдущем блоке кода. Это свойство не CSS, а SVG. Существует довольно много специализированных свойств SVG, которые можно использовать в стилях (неважно, где они объявлены, во встроенном коде или во внешней таблице стилей). Например, для SVG указывается не `background-color`, а `fill`. Вместо указания `border` указывается `stroke-width`. Полный перечень специализированных свойств SVG можно найти в спецификации по адресу http://www.w3.org/TR/SVG/styling.html.

Как во встроенном, так и во внешнем коде CSS допустимо все, чего можно ожидать от обычного CSS: изменение места появления элементов, анимация, преобразование элементов и т. д.

Анимация SVG-графики с помощью CSS

Рассмотрим небольшой пример добавления в SVG CSS-анимации (следует помнить, что эти стили могут так же легко оказаться и во внешней таблице стилей).

Возьмем пример со звездой, рассматриваемый во многих местах этой главы, и заставим звезду вращаться. Окончательный вид примера можно найти в файле каталога example_07-07:

```
<div class="wrapper">
    <svg width="198" height="188" viewBox="00220200" xmlns="http://
www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
        <title>Star 1</title>
        <defs>
            <style type="text/css">
                <![CDATA[
```

```
                @keyframes spin {
                    0% {
                        transform: rotate(0deg);
                    }
                    100% {
                        transform: rotate(360deg);
                    }
                }
                .star_Wrapper {
                        animation: spin 2s 1s;
                        transform-origin: 50% 50%;
                }
                .wrapper {
                        padding: 2rem;
                        margin: 2rem;
                }
                ]]>
            </style>
            <g id="shape">
                <path fill="#14805e" d="M50 50h50v50H50z"/>
                <circle fill="#ebebeb" cx="50" cy="50" r="50"/>
            </g>
        </defs>
        <g class="star_Wrapper" fill="none" fill-rule="evenodd">
            <path id="star_Path" stroke="#333" stroke-width="3"
fill="#F8E81C"  d="M99154l-58.7830.90211.227-65.45L3.89473.097165.717-
9.55L994129.3959.5565.7169.548-47.55346.35311.22665.453z"/>
        </g>
    </svg>
</div>
```

Если загрузить этот пример в браузер, то после задержки в одну секунду звезда начнет вращаться, обойдя полный круг в течение двух секунд.

СОВЕТ

Заметили, как SVG-элементу была назначена исходная точка преобразования 50 % 50 %? Дело в том, что, в отличие от CSS, по умолчанию исходная точка преобразования в SVG не устанавливается как 50 % 50 % (по центру обеих осей), а имеет значение 0 0 (в левом верхнем углу). Если не установить это свойство, звезда будет вращаться вокруг точки в левом верхнем углу.

Используя одну лишь CSS-анимацию, в анимации SVG-графики можно зайти довольно далеко (если предположить, что насчет Internet Explorer волноваться не придется). Но если нужно добавить интерактивности, поддержку Internet Explorer или синхронизировать ряд событий, лучше будет положиться на JavaScript. Хорошей новостью в этом плане является наличие большого количества библиотек, существенно облегчающих анимацию SVG-графики. А теперь рассмотрим примеры.

Анимация SVG-графики с помощью JavaScript

Когда SVG-графика вставлена в страницу посредством тега `object` или непосредственно в код самой страницы, появляется возможность работать с SVG-графикой напрямую или опосредованно с помощью JavaScript.

Под опосредованностью подразумевается возможность средствами JavaScript изменить класс самого SVG-элемента или элемента, в который заключена SVG-графика, что должно инициировать запуск CSS-анимации, например:

```
svg {
    /* нет анимации */
}

.added-with-js svg {
    /* есть анимация */
}
```

Но также возможно применить эффект анимации к SVG-графике с помощью JavaScript непосредственным образом.

Если анимации подвергаются один или два элемента независимо друг от друга, то, возможно, написанный вручную код JavaScript будет рациональнее, что существенно облегчит задачу. Но если нужно подвергнуть эффектам анимации большое количество элементов или синхронизировать анимацию элементов, как с применением шкалы времени, то реальную помощь в этом могут оказать библиотеки JavaScript. В конечном счете вам решать, насколько оправданным окажется утяжеление страницы за счет включения в нее библиотеки для достижения цели.

Для придания эффектов анимации SVG-графике посредством JavaScript хочу порекомендовать анимационную платформу GreenSock (http://greensock.com), Velocity.js (http://julian.com/research/velocity/) или Snap.svg (http://snapsvg.io/). Далее мы рассмотрим очень простой пример использования GreenSock.

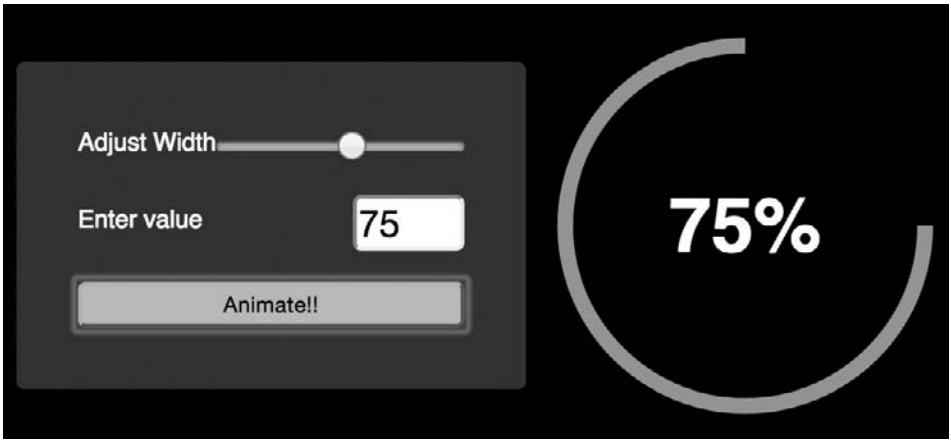
Простой пример анимации SVG-графики с помощью GreenSock. Предположим, что нам нужно создать интерфейс с круглой шкалой, подвергаемой анимации по щелчку на кнопке от нулевой позиции до позиции, задаваемой вводимым значением. Нам нужно не только подвергнуть анимации продвижение по круглой шкале как по длине, так и по цвету, но и вывести число от нуля до вводимого значения. Полный код реализации можно найти в файле каталога example_07-08.

Если будет введено значение `75` и нажата кнопка запуска анимации, круглая шкала будет заполнена и возникнет следующая картина.

Вместо демонстрации распечатки всего файла JavaScript, в котором имеется много комментариев, чтобы стимулировать его отдельное изучение, для краткости мы рассмотрим только ряд ключевых моментов.

Основной замысел состоит в том, что окружность создана в виде SVG-тега `<path>`, а не элемента `<circle>`. То, что используется путь, означает, что его можно подвергнуть анимации, как будто он прорисовывается с использованием технологии `stroke-dashoffset`. Краткое изложение этой технологии представлено на следующей

вкладке. Мы используем JavaScript для измерения длины пути, а затем применим атрибут `stroke-dasharray` для указания длины выводимой на экран части окружности и длины пустого промежутка. Затем `stroke-dashoffset` используется для изменения места старта этого `dasharray`. Это означает, что можно фактически запустить обвод по контуру вне пути и анимировать этот обвод. Это создает иллюзию, что путь прорисовывается.



Если бы значение для анимации `dasharray` было известным статичным значением, такого же эффекта можно было бы относительно легко достичь с помощью CSS-анимации и некоторого количества проб и ошибок (более подробно CSS-анимация будет рассмотрена в следующей главе).

Но вдобавок к динамическому значению одновременно с прорисовкой линии нам нужны постепенное повышение яркости цвета от одного значения к другому и визуальный подсчет до введенного значения, выполняемый в текстовом узле. Эта анимация сродни одновременному похлопыванию по голове, поглаживанию живота и обратному отсчету от 10 000. Средство GreenSock существенно облегчает задачу — в анимационной части оно не будет похлопывать вас по голове или поглаживать по животу, хотя может при необходимости выполнить обратный отсчет от 10 000. Вот строки кода JavaScript, необходимые, чтобы заставить GreenSock выполнить все три задачи:

```
// анимация прорисовки линии и изменения цвета
TweenLite.to(circlePath, 1.5, {'stroke-dashoffset': "-"+amount,
stroke: strokeEndColour});
// установка счетчика в нуль и его анимация до введенного значения
var counter = { var: 0 };
TweenLite.to(counter, 1.5, {
    var: inputValue,
    onUpdate: function () {
        text.textContent = Math.ceil(counter.var) + "%";
    },
    ease:Circ.easeOut
});
```

По сути, функции `TweenLite.to()` передаются: то, что нужно подвергнуть анимации, время, за которое эта анимация должна осуществиться, а затем значения, которые нужно изменить (и то, что в них нужно изменить).

Сайт GreenSock содержит отличную документацию и поддерживает форумы, и если вам понадобится одновременно синхронизировать целый ряд анимаций, выкроите денек в своем рабочем расписании и ознакомьтесь с GreenSock.

СОВЕТ

Если вам еще не приходилось сталкиваться с SVG-технологией прорисовки линий, она была изложена в доступной форме журналом Polygon, когда Vox Media анимировала несколько прорисовок линий в игровых консолях Xbox One и Playstation 4. Исходную статью можно прочитать по адресу http://product.voxmedia.com/2013/11/25/5426880/polygon-feature-design-svg-animations-for-fun-and-profit.

Есть еще великолепное и более основательное объяснение этой технологии, изложенное Джейком Арчибальдом (Jake Archibald) по адресу http://jakearchibald.com/2013/animated-line-drawing-svg/.

Оптимизация SVG

Как добросовестные разработчики, мы хотим обеспечить наименьший возможный объем своих ресурсов. Проще всего это сделать с SVG-графикой, воспользовавшись средствами анимации, способными оптимизировать различные реквизиты SVG-документов. Кроме очевидной экономии, такой как удаление элементов (например, удаление элементов заголовков и описаний), можно также выполнить целую кучу микрооптимизаций, которые в сумме делают SVG-активы намного компактнее.

В настоящее время для решения данной задачи можно порекомендовать средство SVGO (https://github.com/svg/svgo). Если вам еще не приходилось пользоваться SVGO, рекомендую начать с SVGOMG (https://jakearchibald.github.io/svgomg/). Это версия SVGO, основанная на применении браузера и позволяющая переключать различные дополнительные модули оптимизации, получая немедленную реакцию в виде сохраненного файла.

Помните пример SVG-разметки для получения звезды, приведенный в начале главы? В исходном виде этот простой SVG-документ занимает 489 байт. Пропустив его через SVGO, можно уменьшить размер до 218 байт, оставив на месте `viewBox`. Тем самым будет сэкономлено 55,42 % объема. Если используется множество SVG-изображений, такая экономия может вылиться в весьма солидный результат. Оптимизированная SVG-разметка имеет следующий вид:

```
<svg width="198" height="188" viewBox="00198188" xmlns="http://
www.w3.org/2000/svg"><path stroke="#979797" stroke-width="3"
fill="#F8E81C"  d="M991541-58.7830.90211.227-65.45L3.89473.097165.717-
9.55L994129.3959.5565.7169.548-47.55346.35311.22665.454z"/></svg>
```

Прежде чем потратить уйму времени на освоение SVGO, следует узнать, что благодаря популярности SVGO этим средством пользуются и другие SVG-средства. Например, ранее упомянутое средство Iconizr (http://iconizr.com/) перед созданием ваших ресурсов по умолчанию прогоняет SVG-файлы через SVGO, избавляя вас от ненужной двойной оптимизации.

Использование SVG в качестве фильтров

В главе 6 рассматривались эффекты, создаваемые фильтрами CSS. Но в настоящее время в Internet Explorer 10 или 11 они не поддерживаются. Если вы собирались воспользоваться в этих браузерах эффектами фильтров, то ничего, кроме досады, не испытаете. К счастью, с помощью SVG можно создавать фильтры, работающие и в Internet Explorer 10 и 11, но, как всегда, все не настолько просто, как можно было бы себе представить. Например, в файле каталога example_07-05 имеется страница, в теле которой содержится следующая разметка:

```
<img class="HRH" src="queen@2x-1024x747.png"/>
```

Это фотография королевы Великобритании. Изначально она имеет следующий вид.

В этом же каталоге примера содержится SVG-документ с фильтром, определенным в элементах `defs`. Разметка SVG имеет следующий вид:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
    <defs>
        <filter id="myfilter" x="0" y="0">
            <feColorMatrix in="SourceGraphic" type="hueRotate"
            values="90" result="A"/>
            <feGaussianBlur in="A" stdDeviation="6"/>
```

```
        </filter>
    </defs>
</svg>
```

Внутри фильтра сначала задается вращение тона на 90° с помощью `feColorMatrix`, а затем этот эффект через атрибут `result` передается следующему фильтру (`feGaussianBlur`) со значением размытия, равным 6. Сразу предупреждаю, что задал слишком жесткие параметры в ущерб эстетическому восприятию намеренно, чтобы у нас не осталось никаких сомнений в том, что эффект работает!

Теперь вместо добавления этой SVG-разметки к HTML мы можем оставить ее там, где она находится, и сослаться на нее, используя тот же синтаксис CSS-фильтра, который уже видели в предыдущей главе:

```
.HRH {
    filter: url('filter.svg#myfilter');
}
```

В самых популярных браузерах (Chrome, Safari, Firefox) будет получен следующий эффект.

К сожалению, этот метод не работает в Internet Explorer 10 или 11. Но есть и другой способ достижения нашей цели, заключающийся в использовании собственного тега `image` SVG-документа, предназначенного для включения изображения в этот документ. В файле каталога example_07-06 содержится следующая разметка:

```
<svg height="747px" width="1024px" viewbox="0 0 1024 747"
xmlns="http://www.w3.org/2000/svg" version="1.1">
    <defs>
        <filter id="myfilter" x="0" y="0">
            <feColorMatrix in="SourceGraphic" type="hueRotate"
            values="90" result="A"/>
            <feGaussianBlur in="A" stdDeviation="6"/>
        </filter>
    </defs>
    <image x="0" y="0" height="747px" width="1024px"
xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="queen@2x-
1024x747.png" filter="url(#myfilter)"></image>
</svg>
```

Здесь SVG-разметка очень похожа на внешний фильтр `filter.svg`, который использовался в предыдущем примере, но с добавлением атрибутов `height`, `width` и `viewbox`. Кроме того, изображение, к которому мы хотим применить фильтр, является единственным содержимым SVG-документа за пределами элемента `defs`. Для ссылки на фильтр используется атрибут `filter` и передается идентификатор того фильтра, который мы хотим использовать. (В данном случае из внутреннего содержимого расположенного выше элемента `defs`.)

Хотя над этим подходом придется поработать немного больше, он предназначен для того, чтобы дать вам множество разнообразных фильтрующих эффектов, предоставляемых SVG-технологией, даже в версиях 10 и 11 браузера Internet Explorer.

Заметки по медиазапросам внутри SVG

Все браузеры, понимающие SVG-документы, должны уважать определяемые внутри этих документов медиазапросы CSS. Но при работе с медиазапросами внутри SVG-документов следует помнить о некоторых особенностях.

Предположим, к примеру, что вы вставляете медиазапрос в SVG-документ следующим образом:

```
<style type="text/css"><![CDATA[
    #star_Path {
        stroke: red;
    }
    @media (min-width: 800px) {
        #star_Path {
            stroke: violet;
        }
    }
]]></style>
```

И этот документ выводит на странице SVG-графику шириной 200 пикселов, в то время как окно просмотра имеет ширину 1200 пикселов.

Предполагается, что обвод звезды по контуру будет лиловым (violet), когда ширина экрана не меньше 800 пикселов. Все-таки именно на это настроен наш медиазапрос. Но когда SVG-графика помещается на страницу посредством тега

`img`, или в качестве фонового изображения, или внутри тега `object`, то, какой при этом окружающий HTML-документ, в расчет не берется. Следовательно, в данной ситуации `min-width` означает минимальную ширину самой SVG-графики. Поэтому, хотя сама SVG-фигура отображается на странице шириной не менее 800 пикселов, обвод по контуру лиловым не будет.

В противовес этому, когда SVG-документ вставляется непосредственно в код страницы, он, образно говоря, сливается с окружающим HTML-документом. И здесь уже медиазапрос с `min-width` для того, чтобы принять решение о своем соответствии ситуации, смотрит на параметры окна просмотра (точно так же, как в HTML).

Для решения данной конкретной проблемы, чтобы заставить одни и те же медиазапросы вести себя одинаково, можно придать нашему медиазапросу следующий вид:

```
@media (min-device-width: 800px) {
    #star_Path {
        stroke: violet;
    }
}
```

Тогда независимо от размера SVG-графики или способа ее вставки на страницу в расчет будет браться ширина экрана устройства (а по сути, окна просмотра).

Советы по внедрению

Мы почти добрались до конца главы, хотя касательно технологии SVG можно было бы еще о многом поговорить. Поэтому сейчас я перечислю ряд не связанных друг с другом соображений. Им не обязательно давать развернутое толкование, но я перечислю их здесь в форме заметок, чтобы вы сэкономили час-другой на поиске в Google.

- Если вам не требуется придавать SVG-графике эффект анимации, остановите свой выбор на спрайте изображения ваших ресурсов или на таблице стилей с указанием на URI-идентификатор данных. Вам будет намного проще предоставить альтернативные ресурсы, и они практически всегда выгодно отличаются от всего остального с точки зрения поддержания высокой производительности.
- Автоматизируйте как можно больше шагов по созданию ресурса, тем самым вы уменьшите количество допускаемых вами ошибок и предсказуемый результат будет получен гораздо быстрее.
- Для вставки в проект статичных SVG-объектов постарайтесь выбрать единый механизм доставки и придерживаться только его (спрайт изображения, URI-идентификатор данных или встраивание непосредственно в код страницы). Получение одних активов одним способом, а других — другим может стать обременительным занятием, равно как и дальнейшая поддержка различных реализаций.
- Единого варианта выбора SVG-анимации на все случаи жизни не существует. Для получения простых, применяемых от случая к случаю эффектов анимации следует воспользоваться CSS. Для сложных интерактивных эффектов анимации или таких эффектов, ход которых меняется по шкале времени и которые работают и в Internet Explorer, освойте какую-нибудь опробованную библиотеку, например GreenSock, Velocity.js или Snap.svg.

Дополнительные ресурсы

В начале главы уже говорилось, что у меня нет ни места, ни достаточного объема знаний, чтобы поведать буквально обо всем, что касается SVG. Поэтому я лучше поставлю вас в известность о существовании следующих великолепных ресурсов, предоставляющих более глубокий и более широкий взгляд на данный предмет изучения:

- *SVG Essentials*, второе издание, Дж. Дэвида Эйсенберга (J. David Eisenberg) и Амелии Беллами-Роудс (Amelia Bellamy-Royds) (http://shop.oreilly.com/product/063 69200 32335.do);
- *A Guide to SVG Animations (SMIL)* Сары Сьюайден (Sara Soueidan) (http://css-tricks.com/guide-svg-animations-smil/);
- *Media Queries inside SVGs Test* Жереми Патонье (Jeremie Patonnier) (http://jeremie.patonnier.net/experiences/svg/media-queries/test.html).
- *An SVG Primer for Today's Browsers* (http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html);
- *Understanding SVG Coordinate Systems and Transformations (Part 1)* Сары Сьюайден (Sara Soueidan) (http://sarasoueidan.com/blog/svg-coordinate-systems/);
- *Hands On: SVG Filter Effects* (http://ie.microsoft.com/testdrive/graphics/hands-on-css3/hands-on_svg-filter-effects.htm);
- *Full set of SVG tutorials* Якоба Женкова (Jakob Jenkov) (http://tutorials.jenkov.com/svg/index.html).

Резюме

В данной главе рассмотрен большой объем важной информации, необходимой для понимания SVG-графики и ее внедрения в проекты создания адаптивных веб-приложений. Были изучены различные графические приложения и онлайн-решения, доступные для создания SVG-ресурсов, затем рассмотрены доступные методы их вставки и возможности, предоставляемые каждым из этих методов, а также особенности различных браузеров, которые следует брать в расчет.

Мы также узнали, как ссылаться на внешние таблицы стилей и повторно использовать SVG-символы как на одной и той же странице, так и при внешней ссылке на них. Мы даже посмотрели, как создаются фильтры, использующие SVG, на которые можно сослаться и которые можно использовать в CSS для более широкой поддержки по сравнению с фильтрами CSS.

И наконец, мы рассмотрели способы использования библиотек JavaScript с целью анимации SVG-графики, а также изучили методы оптимизации SVG-графики с помощью средства SVGO.

В следующей главе будут рассмотрены CSS-переходы, преобразования и анимация. Ее стоит изучить и в отношении SVG-графики, поскольку многие синтаксические примеры и технологии могут использоваться применительно к SVG-документам. Итак, запаситесь каким-нибудь горячим питьем (вы его заслужили) и продолжайте чтение книги.

8 Переходы, преобразования и анимация

Исторически сложилось так, что перемещение элементов по экрану или применение к ним эффектов анимации было исключительной прерогативой JavaScript. Сегодня с основной частью задач по перемещению может справиться код CSS с использованием для этого трех своих основных агентов: CSS-переходов (transitions), CSS-преобразований (transforms) и CSS-анимации (animations). Фактически к перемещению имеют непосредственное отношение только переходы и анимация, а преобразование просто позволяет изменять элементы, но, как мы вскоре увидим, оно часто встраивается и в весьма впечатляющие эффекты перемещений.

Чтобы четко разобраться в том, за что отвечает каждая из этих трех составляющих, хочу предложить, возможно, несколько упрощенное описание.

- CSS-переход следует использовать, когда уже имеются начальное и конечное состояние того элемента, к которому его нужно применить, и требуется простой способ получения промежуточных переходов от одного состояния к другому.
- CSS-преобразование используется, когда нужно обеспечить визуальное преобразование элемента, не оказывая при этом никакого влияния на разметку страницы.
- CSS-анимация используется, когда нужно применить к элементу серию изменений с различными ключевыми точками, определяемыми по времени.

Итак, мы уловили смысл и получили представление о том, как всем этим можно воспользоваться, а теперь узнаем, что в этой главе будут рассматриваться следующие вопросы:

- что такое CSS3-переходы и как ими можно воспользоваться;
- как создать код CSS3-перехода и что собой представляет его сокращенный синтаксис;
- что такое функции развития процесса CSS3-перехода во времени (`ease`, `cubic-bezier` и т. д.);
- как можно развлечься с переходами на адаптивных сайтах;
- что такое CSS3-преобразования и как ими можно воспользоваться;
- что представляют собой различные 2D-преобразования (`scale`, `rotate`, `skew`, `translate` и т. д.);

- что такое 3D-преобразования;
- как средствами CSS3 осуществлять анимацию, используя `keyframes`.

Что такое CSS3-переходы и как ими можно воспользоваться

Переходы между двумя состояниями являются простейшим способом создания с помощью CSS визуального эффекта. Рассмотрим простой пример перехода элемента из одного состояния в другое при прохождении над ним указателя мыши.

При стилевом оформлении гиперссылок в CSS сложилась практика создания особого вида элемента с использованием состояния, при котором на этот элемент наведен указатель мыши, чтобы пользователь знал, что этот элемент является ссылкой. Состояния наведения указателя, конечно, имеют весьма слабую связь с постоянно возрастающим количеством устройств с сенсорным экраном, но для тех, кто пользуется мышью, они служат весьма эффективным и простым средством взаимодействия между сайтом и пользователем. Они также пригодятся для иллюстрации переходов, чем мы сейчас и займемся.

Традиционно при использовании одного лишь кода CSS с помощью состояний, связанных с нахождением указателя мыши над элементом, CSS-правила могут только включаться или выключаться. У элемента есть один исходный набор свойств и значений, и когда указатель проходит над элементом, свойства и значения тут же изменяются. Но CSS3-переходы, как следует из их названия, позволяют обеспечивать переход от одних или нескольких свойств и значений к другим свойствам и значениям.

СОВЕТ

Сначала нужно предупредить вас о двух весьма важных обстоятельствах. Во-первых, переход от состояния display: none; невозможен. Если для элемента установлено свойство display: none;, то его прорисовка на экране отсутствует, следовательно, отсутствует и реальное состояние, с которого можно было бы осуществить переход. Чтобы создать процесс постепенного появления элемента, нужно задавать переход в отношении непрозрачности или позиции элемента. Во-вторых, не все свойства подвергаются переходам. Чтобы убедиться в том, что вы не пытаетесь добиться невозможного, обратитесь к перечню доступных для переходов свойств, который можно найти по адресу http://www.w3.org/TR/css3-transitions/.

Если открыть файл из каталога example_08-01, то в нем, в контейнере `nav`, можно увидеть несколько ссылок. Разметка имеет следующий вид:

```
<nav>
    <a href="#">link1</a>
    <a href="#">link2</a>
    <a href="#">link3</a>
    <a href="#">link4</a>
    <a href="#">link5</a>
</nav>
```

А вот соответствующий ей код CSS:

```
a {
    font-family: sans-serif;
    color: #fff;
    text-indent: 1rem;
    background-color: #ccc;
    display: inline-flex;
    flex: 1120%;
    align-self: stretch;
    align-items: center;
    text-decoration: none;
    transition: box-shadow 1s;
}

a + a {
    border-left: 1px solid #aaa;
}

a:hover {
    box-shadow: inset 0 -3px 0 #CC3232;
}
```

И два состояния: первое, исходное.

И второе, при наведении указателя мыши.

В данном примере при наведении указателя на элемент к нижней границе этого элемента добавляется красная блочная тень (мой выбор пал на блочную тень, поскольку она не влияет на разметку ссылки, как это может случиться при использовании границы элемента). Обычно проведение указателя мыши над ссылкой переводит ее из первого состояния (без красной линии) во второе (с красной линией) по принципу включения-выключения. Но вот эта строка:

```
transition: box-shadow 1s;
```

добавит к блочной тени переход от обычного состояния к состоянию наведения указателя за одну секунду.

СОВЕТ

В предыдущем примере кода CSS можно заметить использование селектора примыкающего одноуровневого элемента +. Это означает, что заключенные в него стили будут применены в том случае, если за одним выбранным тегом (в нашем случае это <a>) тут же следует другой выбранный тег (еще один тег <a>). Здесь этот селектор применяется из тех соображений, что левая граница у первого элемента нам не нужна.

Учтите, что свойство перехода применяется в CSS к исходному состоянию элемента, а не к тому состоянию, в котором он в конечном итоге окажется. Короче говоря, объявление перехода применяется не к конечному, а к начальному состоянию. Это объясняется тем, что различные состояния, такие как `:active`, также могут иметь различные стилевые установки и подвергаться такому же переходу.

Свойства перехода

При объявлении перехода можно воспользоваться четырьмя свойствами:

- `transition-property` — название CSS-свойства, подвергаемого переходу (например, `background-color`, `text-shadow` или название любого другого доступного для применения переходов свойства);
- `transition-duration` — время, за которое должен произойти переход (определенное в секундах, например `.3s`, `2s` или `1.5s`);
- `transition-timing-function` — порядок изменения скорости в процессе перехода (например, `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out` или `cubic-bezier`);
- `transition-delay` — дополнительное значение для указания задержки до начала перехода. Как вариант, может использоваться отрицательное значение для немедленного начала перехода, но уже с того момента, до которого переход дошел бы за указанное время. Определяется в секундах, например `.3s`, `1s` или `2.5s`.

Используемые по отдельности различные свойства могут быть задействованы для создания следующего перехода:

```
.style {
    /*...(дополнительные стилевые настройки)...*/
    transition-property: all;
    transition-duration: 1s;
    transition-timing-function: ease;
    transition-delay: 0s;
}
```

Краткая форма записи перехода с помощью свойства transition

Показанные ранее отдельные объявления можно свести в одну краткую форму:

```
transition: all 1s ease 0s;
```

Но при этом следует сделать одно важное замечание: при записи краткой версии первое значение, связанное со временем, всегда применяется для свойства `transition-`

duration. Второе значение, также связанное со временем, относится к свойству transition-delay. Я склоняюсь к использованию именно сокращенной формы, поскольку обычно мне нужно лишь определить продолжительность перехода и те свойства, к которым он должен быть применен.

Конечно, это мелочь, но постарайтесь указывать только то свойство или свойства, к которым действительно нужно применить переход. Безусловно, удобно установить значение all, имея в виду все свойства, но если переход нужен применительно лишь к непрозрачности, то свойству transition нужно указать только opacity. В противном случае браузер будет перегружен ненужной работой. В большинстве случаев это не создаст проблем, но если питать надежды на получение от сайта наивысшей производительности, особенно на не самых новых устройствах, то тут пригодится любая мелочь.

СОВЕТ

Переходы пользуются весьма широкой поддержкой, но, как всегда, не поленитесь установить такое средство, как Autoprefixer, чтобы добавить префиксы производителей, нужные браузерам, от которых требуется поддержка вашего сайта. Можно также зайти на сайт caniuse.com и проверить, какие возможности какими браузерами поддерживаются.

Короче говоря, переходы и 2D-преобразования работают практически везде, кроме IE9 и более ранних версий, 3D-преобразования работают везде, кроме IE9 и более ранних версий, Android 2.3 и более ранних версий и Safari 3.2 и более ранних версий.

Переходы различных свойств за разные периоды времени

Когда в правиле содержатся несколько объявленных свойств, то не обязательно реализовывать переходы их всех за одно и то же время. Рассмотрим следующее правило:

```
.style {
    /* ...( дополнительные стилевые настройки)... */
    transition-property: border, color, text-shadow;
    transition-duration: 2s, 3s, 8s;
}
```

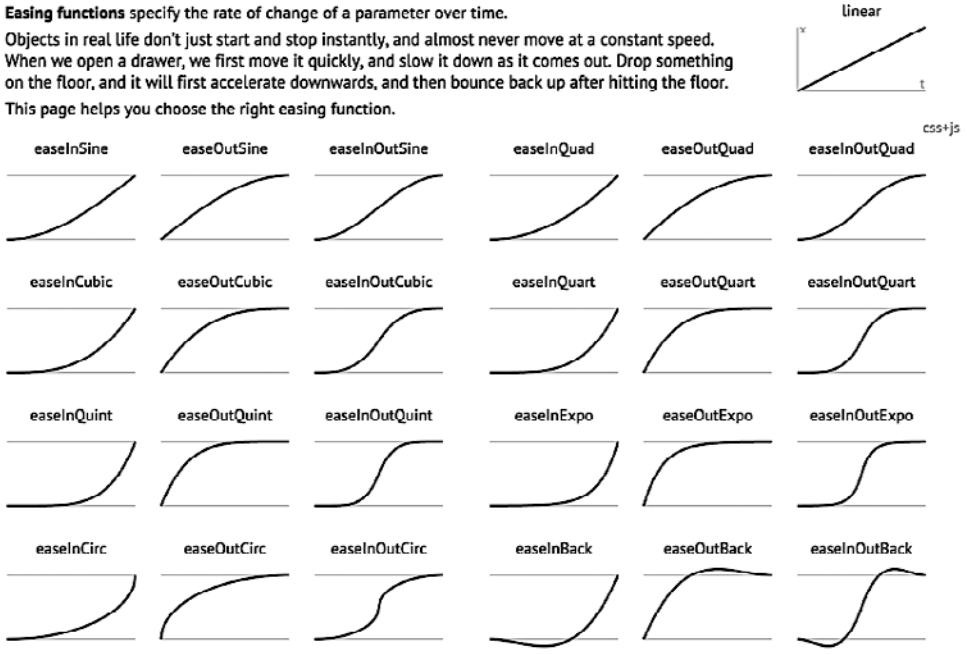
Здесь с помощью объявления transition-property указаны свойства, подвергаемые переходу: border, color и text-shadow. А затем с помощью объявления transition-duration указано, что свойство border будет подвергнуто переходу за две секунды, color — за три секунды, а text-shadow — за восемь секунд. Указания продолжительностей переходов, разделенные запятыми, соответствуют порядку указания подвергаемых переходу свойств, при перечислении которых в качестве разделителей также применялись запятые.

Основные сведения о функциях развития процесса по времени

Понять при объявлении перехода, что такое свойства, продолжительность и задержка, нетрудно. Но вот понять, что делает каждая функция развития процесса по времени, намного труднее. Чем же занимаются ease, linear, ease-in, ease-out,

ease-in-out и cubic-bezier? Каждая из них фактически является предопределенной кривой Безье третьего порядка (cubic-bezier curve) и, по сути, аналогична функции плавности. Или, проще говоря, является математическим описанием того, как должен выглядеть процесс перехода. Нагляднее будет показать эти кривые, поэтому я рекомендую вам заглянуть по адресам http://cubic-bezier.com/ и http://easings.net/.

На обоих указанных сайтах вы можете сравнить функции развития процесса по времени и определить различия, вносимые каждой из них в этот процесс. Вот копия экрана, сделанная при просмотре сайта http://easings.net, где можно навести указатель мыши на каждую линию для демонстрации функции плавности.



Но даже если вы способны записать собственную кривую Безье вслепую, скорее всего, на практике в большинстве ситуаций это мало что изменит. Причина, как и в случае со многими другими улучшениями, заключается в том, что при применении эффектов переходов следует проявлять определенную сдержанность. При реальной работе переходы, длящиеся слишком долго, создают впечатление заторможенности сайта. К примеру, навигационные ссылки с переходами по пять секунд способны скорее навредить работе, чем впечатлить ваших пользователей. Ощущение скорости играет весьма важную роль для наших пользователей, поэтому и вы и я должны сконцентрироваться на создании сайтов и приложений, скорость работы которых ощущается как можно более высокой.

Поэтому, если для иного нет веских причин, зачастую наилучшим выбором будет использование исходного развития процесса перехода (ease) за короткий промежуток времени, который, на мой взгляд, не должен быть более одной секунды.

Развлечение с переходами на адаптивных сайтах

Бывало ли у вас в детстве такое, что один из родителей куда-то уезжал на целый день, а другой, чтобы завоевать ваше расположение, говорил что-нибудь вроде: «Пока мамы (папы) нет дома, мы будем сыпать сахар во все сухие завтраки, но ты должен пообещать, что не скажешь об этом, когда она (он) вернется»? Каюсь, я тоже проделывал нечто подобное со своими малышами. В этом-то все и дело: когда никто не видит, можно немного порезвиться. Я не собираюсь советовать внедрение этого в серьезные приложения, но попробуйте следующий код в своем проекте по разработке адаптивного веб-приложения:

```
* {
    transition: all 1s;
}
```

Здесь используется универсальный CSS-селектор *, предназначенный для выбора абсолютно всего, а затем устанавливается переход всех свойств за одну секунду (`1s`). Поскольку функция развития процесса по времени не указана, то по умолчанию станет использоваться `ease`, а задержки не будет, так как если дополнительно ее значение не указано, то по умолчанию принимается значение `0`. Каков будет эффект? А вы попробуйте изменить размер окна вашего браузера, и большинство того, что в нем находится (ссылки, состояния прохода указателя мыши и т. п.), будет вести себя вполне предсказуемо. Но, поскольку под переходы подпадает абсолютно все, сюда включаются и любые правила внутри медиазапросов, поэтому при изменении размеров окна элементы некоторым образом переходят из одного состояния в другое. Важно ли это? Конечно, нет! Забавно наблюдать за этим и развлекаться? Конечно, да! А теперь уберите это правило, пока мама не увидела!

CSS3 2D-преобразования

Несмотря на схожесть названий, CSS-преобразования (transforms) не имеют ничего общего с CSS-переходами (transitions). Их нужно рассматривать в следующем ключе: переходы сглаживают изменения, происходящие при перетекании одного состояния в другое, а преобразования определяют, во что элемент превратится. Лично я запоминаю разницу абсолютно по-детски: представляю себе робот-трансформер, например, Optimus Prime. Когда он превращается в грузовик, это преобразование. Но фаза между роботом и грузовиком является переходом (он переходит из одного состояния в другое).

Разумеется, если вам невдомек, кто или что такое Optimus Prime, можете мысленно отбросить несколько последних предложений. Надеюсь, вскоре все станет ясно и понятно.

Доступны две группы CSS3-преобразований: 2D и 3D. 2D-варианты реализованы намного шире, браузеры в них неплохо разбираются, и их, конечно же, легче

записывать, поэтому с них и начнем. Модуль CSS3 2D Transforms позволяет воспользоваться следующими преобразованиями:

- `scale` — используется для масштабирования элемента (его увеличения или уменьшения);
- `translate` — перемещает элемент по экрану (вверх, вниз, влево и вправо);
- `rotate` — вращает элемент на определенную величину, указанную в градусах;
- `skew` — используется для наклона элемента по его координатам *x* и *y*;
- `matrix` — позволяет выполнять перемещения и придавать форму преобразованиям с точностью до пиксела.

СОВЕТ

Важно запомнить, что преобразования осуществляются вне процесса формирования документа. Любой элемент, подвергаемый преобразованию, не влияет на позицию соседнего элемента, не подвергающегося преобразованию.

Попробуем применить различные 2D-преобразования, которые можно посмотреть в действии, открыв в браузере файл каталога example_08-02. Здесь, чтобы вы могли лучше разобраться со всем, что происходит, ко всем преобразованиям применяются переходы.

Масштабирование (scale)

Для масштабирования, `scale`, используется следующий синтаксис:

```
.scale:hover {
    transform: scale(1.4);
}
```

Проход указателя мыши над ссылкой, имеющей класс `scale`, произведет следующий эффект.

Браузеру указывается, что нам нужно, чтобы при наведении на элемент указателя мыши этот элемент увеличивался в 1,4 раза от исходного размера.

Кроме тех значений, которые уже использовались для увеличения элементов, использование значений меньше единицы приведет к сжатию элементов. Следующий код приведет к сжатию элемента до половины его размера:

```
transform: scale(0.5);
```

Перемещение (translate)

Для перемещения, translate, используется следующий синтаксис:

```
.translate:hover {
    transform: translate(-20px, -20px);
}
```

В нашем примере это правило приведет к следующему эффекту.

Свойство translate указывает браузеру на необходимость перемещения элемента на расстояние, определяемое либо в пикселах, либо в процентах. Первое значение относится к перемещению по оси *X*, второе — по оси *Y*. Положительные значения, заданные в скобках, приводят к перемещению элемента вправо и вниз, а отрицательные соответственно влево и вверх.

Если передается только одно значение, то оно применяется к оси *X*. Если нужно указать для перемещения элемента значение только для одной оси, можно также воспользоваться объявлениями translateX или translateY.

Использование перемещения для центрирования элементов с абсолютным позиционированием. Перемещение обеспечивает весьма полезный способ центрирования элементов с абсолютным позиционированием внутри контейнера с относительным позиционированием. Пример кода можно найти в файле каталога example_08-03.

Рассмотрим разметку:

```
<div class="outer">
    <div class="inner"></div>
</div>
```

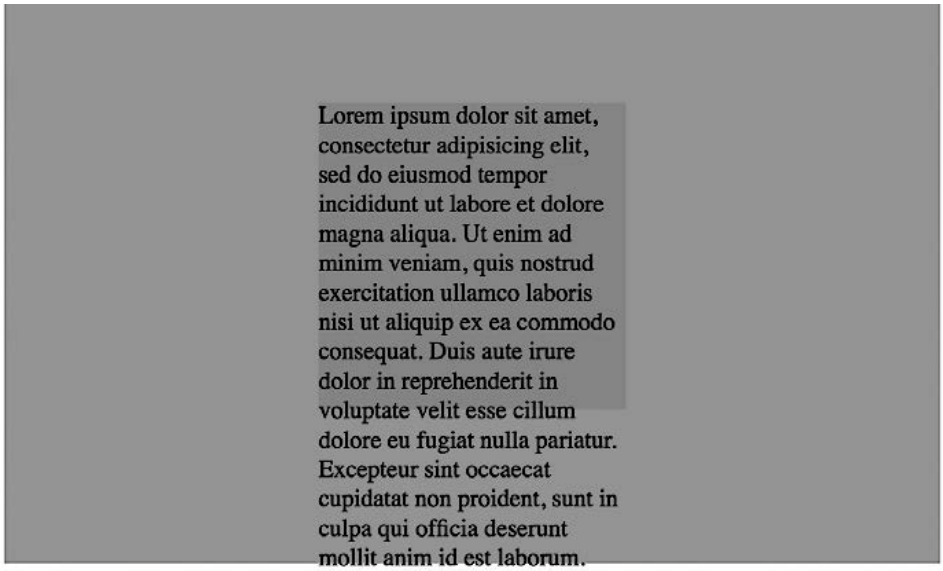
И этот код CSS:

```
.outer {
    position: relative;
    height: 400px;
    background-color: #f90;
}

.inner {
    position: absolute;
    height: 200px;
    width: 200px;
    margin-top: -100px;
    margin-left: -100px;
```

```
    top: 50%;
    left: 50%;
}
```

Возможно, вам самим приходилось делать нечто подобное. Когда известны размеры элемента с абсолютным позиционированием (в данном случае это 200 × 200 пикселов), для «подталкивания» элемента обратно в центр можно воспользоваться отступами с отрицательными значениями. А как быть, если вам понадобится включить содержимое и будет совершенно непонятно, насколько высоким оно окажется? На помощь придет перемещение.

Добавим к внутреннему блоку какое-нибудь произвольное содержимое.

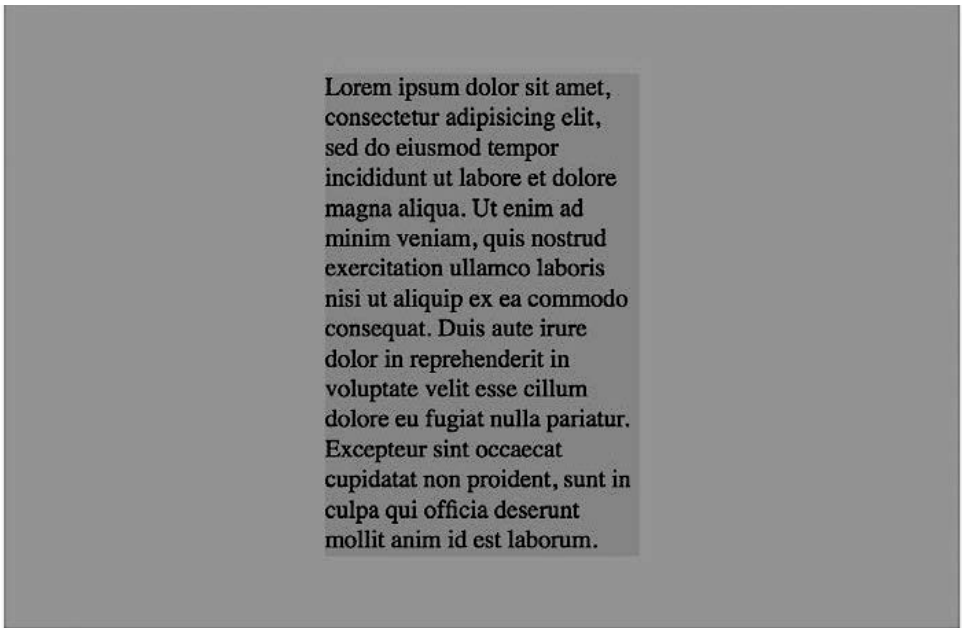


Очевидно, у нас возникла проблема! Хорошо, разберемся с этим беспорядком с помощью перемещения:

```
.inner {
    position: absolute;
    width: 200px;
    background-color: #999;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

И вот результат.

Здесь `top` и `left` позиционируют внутренний блок внутри его контейнера таким образом, что вначале левый верхний угол внутреннего блока находится в точке 50 % длины и 50 % высоты внешнего контейнера. Объявление `transform` работает в отношении внутреннего элемента и позиционирует его в обратную сторону по этим осям на половину (`-50%`) его собственной ширины и высоты. Превосходно!



Вращение (rotate)

Преобразование `rotate` позволяет вращать элемент. Для него используется следующий синтаксис:

```
.rotate:hover {
    transform: rotate(30deg);
}
```

А в окне браузера произойдет следующее.

Значение в скобках всегда задается в градусах (например, `90deg`). Положительные значения задают вращение по часовой стрелке, а отрицательные — против часовой стрелки. Можно также сойти с ума и задать элементу вращение, указав следующее значение:

```
transform: rotate(3600deg);
```

В результате элемент десять раз пройдет по полному кругу. Примеры практического применения данного конкретного значения можно пересчитать по пальцам, но вы знаете, что при создании сайтов для мукомольной компании вам это может пригодиться.

Наклон (skew)

Если вам приходилось работать в программе Photoshop, то чем занимается наклон, вы уже представляете. Он позволяет наклонить элемент по какой-либо из его осей или по двум осям сразу. Вот код нашего примера:

```
.skew:hover {
    transform: skew(40deg, 12deg);
}
```

Установка этого правила для ссылки с псевдоклассом `hover` приведет при наведении указателя мыши на элемент к следующему эффекту.

Первое значение задает наклон применительно к оси *X* (в нашем примере это `40deg`), а второе значение (`12deg`) предназначено для задания наклона по оси *Y*. Если опустить второе значение, то единственное имеющееся значение будет просто применено к оси *X* (горизонтальной оси), например:

```
transform: skew(10deg);
```

Матрица (matrix)

Кто-нибудь вспомнил одноименный и весьма переоцененный фильм? Нет? Что, простите? Вам хочется узнать о CSS3-матрице, а не о фильме? Ну что ж, хорошо.

Я не собираюсь вас обманывать. Думаю, что синтаксис матричного преобразования выглядит несколько страшновато. Рассмотрим пример кода:

```
.matrix:hover {
    transform: matrix(1.678, -0.256, 1.522, 2.333, -51.533, -1.989);
}
```

По сути, матрица позволяет объединять в одно объявление сразу несколько видов преобразований (масштабирование, вращение, наклон и т. д.). Предыдущее объявление реализуется в окне браузера в эффект, показанный на следующей странице.

Теперь мне нравятся сложные задачи, но я уверен, что нельзя не согласиться с тем, что этот синтаксис требует исследований. Лично я расстроился, глядя на

спецификацию и понимая, что математические знания, требующиеся для полного осмысления, выходят за рамки моего рудиментарного уровня: http://www.w3.org/TR/css3-2d-transforms/.

СОВЕТ

Если вам удается работать с анимацией в JavaScript, не прибегая к помощи анимационных библиотек, то, наверное, стоит поближе познакомиться и с матрицей. В ее синтаксисе вычисляются все остальные преобразования, поэтому, если вам удается осмысливать текущие представления о создании анимации с помощью JavaScript, нужно проверить на деле, сможете ли вы так же легко разобраться со значениями матрицы.

Матричные преобразования для ловкачей и остолопов. Представить меня математиком можно только с очень большой натяжкой, поэтому, сталкиваясь с необходимостью создания преобразований на основе матрицы, я немного жульничаю. Если ваши математические навыки также оставляют желать лучшего, рекомендую обратиться по адресу http://www.useragentman.com/matrix/.

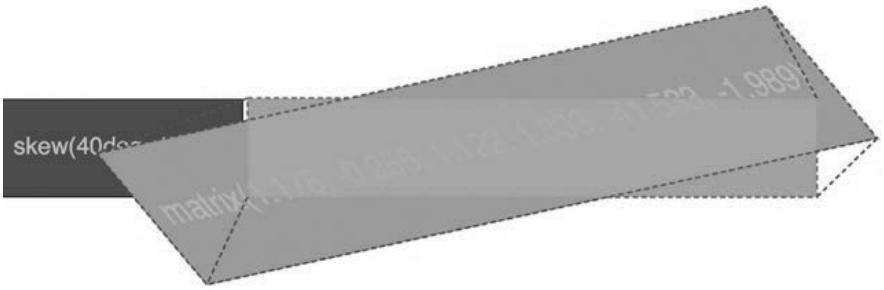
Сайт Matrix Construction Set позволяет перетаскивать элемент, придавая ему в точности тот вид, который вас устраивает, после чего по старой доброй традиции забирать код с уже включенными в него префиксами производителей для вставки в свой файл CSS.

Свойство transform-origin

Обратите внимание на то, что при использовании CSS исходная точка преобразования, которую браузер использует в качестве центра, находится посередине — в 50 % протяженности элемента по оси *X* и в 50 % его протяженности по оси *Y*. Здесь прослеживаются существенные отличия от SVG-технологии, в которой такая точка по умолчанию находится в левом верхнем углу с координатами (0; 0).

Используя свойство `transform-origin`, можно изменить точку, в которой будет начинаться преобразование.

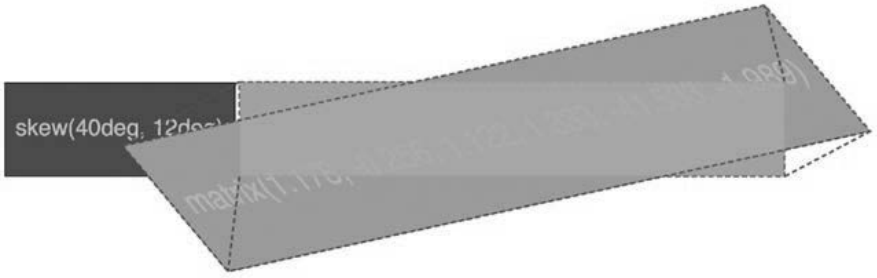
Рассмотрим наше прежнее матричное преобразование. По умолчанию точка начала преобразования `transform-origin` устанавливается в позицию `'50% 50%'` (в центре элемента). Средства разработчика Firefox показывают, как это преобразование применяется.



Теперь, если перенастроить значение transform-origin таким образом:

```
.matrix:hover {
    transform: matrix(1.678, -0.256, 1.522, 2.333, -51.533, -1.989);
    transform-origin: 270px 20px;
}
```

то можно увидеть следующий эффект.



Первое значение задает смещение по горизонтали, а второе — по вертикали. Можно использовать ключевые слова. Например, left задает 0 % по горизонтали, right — 100 % по горизонтали, top — 0 % по вертикали и bottom — 100 % по вертикали. Вместо этого можно воспользоваться длиной, задавая для нее любые единицы измерения, используемые в CSS.

Если для значений свойства transform-origin используются проценты, то горизонтальное и вертикальное смещение задаются относительно высоты и ширины контейнера, содержащего элементы.

Если используется длина, то значения отмеряются от левого верхнего угла контейнера, содержащего элементы.

Полная информация о свойстве transform-origin может быть найдена на сайте http://www.w3.org/TR/css3-2d-transforms/.

Мы рассмотрели основы 2D-преобразований. Они намного более широко реализованы, чем их 3D-собратья, и предоставляют более эффективные средства для перемещения элементов по экрану, чем прежние методы, например абсолютное позиционирование.

Полная спецификация CSS3 2D Transforms Module Level 3 находится по адресу http://www.w3.org/TR/css3-2d-transforms/.

СОВЕТ

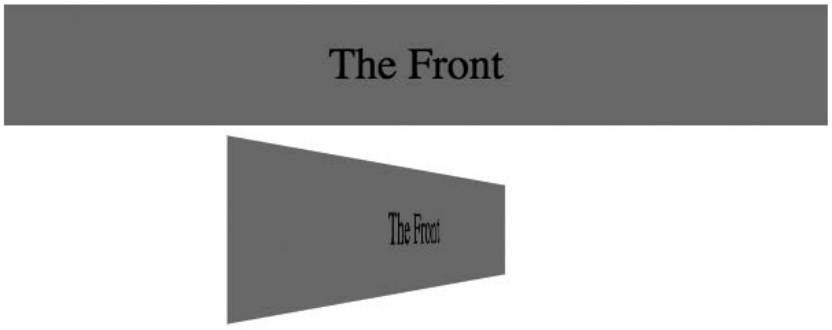
Более полно преимущества перемещения элементов с помощью преобразований рассмотрены в замечательной статье Пола Айриша (Paul Irish) (http://www.paulirish.com/2012/why-moving-elements-with-translate-is-better-than-posabs-topleft/), в которой представлен ряд весьма ценных данных.

А в качестве просто фантастического обзора того, как браузеры справляются с переходами и анимацией, и объяснения, почему преобразования могут быть столь эффектными, я настоятельно рекомендую следующую публикацию в блоге: http://blogs.adobe.com/webplatform/2014/03/18/css-animations-and-transitions-performance/.

CSS3 3D-преобразования

Рассмотрим наш первый пример — элемент, который переворачивается, когда на него наводится указатель мыши. Состояние наведения указателя используется здесь для инициирования изменения просто в целях иллюстрации, но переворот может быть легко вызван изменением класса (через код JavaScript) или получением элементом фокуса.

У нас будет два таких элемента: один с горизонтальным переворотом, другой — с вертикальным. Полный пример можно найти в файле каталога example_08-04. Изображения не могут обеспечить полное восприятие сути этой технологии, для этого лучше подойдет переворот элемента со сменой лицевой стороны с зеленой на красную с полной иллюзией того, что это происходит в трехмерном пространстве с привлечением перспективы. Посмотрите на копию экрана, сделанную на промежуточной фазе перехода от зеленого к красному, которая, надеюсь, в какой-то мере передает суть эффекта.



СОВЕТ

Следует также иметь в виду, что при абсолютном позиционировании элемента с заданием значений top, left, bottom, right в пикселах промежуточные значения преобразований могут вычисляться по субпиксельным позициям.

Разметка для элемента, подвергаемого перевороту, имеет следующий вид:

```
<div class="flipper">
    <span class="flipper-object flipper-vertical">
        <span class="panel front">The Front</span>
        <span class="panel back">The Back</span>
    </span>
</div>
```

Единственным отличием в разметке горизонтального переворота является использование вместо класса `flipper-vertical` класса `flipper-horizontal`.

Поскольку большинство стилевых настроек направлено на улучшение эстетического восприятия, мы рассмотрим только самые важные ингредиенты в наших стилях, с помощью которых осуществляется переворот. Стили, задающие эстетическое восприятие, можно увидеть в полной таблице стилей данного примера.

Прежде всего нам нужно задать объекту с классом `.flipper`, внутри которого будет осуществляться переворот, определенную перспективу. Воспользуемся для этого свойством `perspective`. Ему задается длина, предназначенная для имитации расстояния от экрана просмотра до края 3D-пространства элементов.

Если для перспективы задать небольшое значение, например 20 пикселов, 3D-пространство элемента будет простираться только на эти 20 пикселов от плоскости экрана и в результате получится слишком явно выраженный 3D-эффект. А чересчур большое значение будет означать, что край воображаемого 3D-пространства находится намного дальше, и получится менее выраженный 3D-эффект:

```
.flipper {
    perspective: 400px;
    position: relative;
}
```

Внешний элемент имеет относительное позиционирование, которое предназначено для того, чтобы создать содержимое для `flipper-object`, позиционируемое внутри него:

```
.flipper-object {
    position: absolute;
    transition: transform 1s;
    transform-style: preserve-3d;
}
```

Кроме абсолютного позиционирования `.flipper-object` в левом верхнем углу его ближайшего родительского элемента с относительным позиционированием (что является позицией по умолчанию для абсолютно позиционированных элементов), у нас имеется установка перехода для преобразования. В данном случае ключевым объявлением, касающимся 3D-эффекта, является `transform-style: preserve-3d`. Тем самым браузеру сообщается, что при преобразовании этого элемента мы хотим, чтобы 3D-эффект сохранялся для всех дочерних элементов.

Если для `.flipper-object` не указать `preserve-3d`, обратную сторону (красную часть) переворачиваемого элемента мы никогда не увидим. Спецификацию для этого свойства можно найти по адресу http://www.w3.org/TR/2009/WD-css3-3d-transforms-200 90320/.

Каждая панель переворачиваемого элемента должна получить позиционирование в верхней части своего контейнера, но мы также хотим, чтобы при вращении обратная сторона элемента была не видна (в противном случае мы никогда не увидим зеленую панель, поскольку она располагается «позади» красной). Для этого используется свойство `backface-visibility`. Его значение устанавливается в `hidden`, чтобы, как вы догадались, скрыть обратную сторону элемента:

```
.panel {
    top: 0;
    position: absolute;
    backface-visibility: hidden;
}
```

СОВЕТ

Обнаружилось, что в некоторых браузерах у свойства backface-visibility имеется ряд неожиданных побочных эффектов. Особенно полезно будет повысить эффективность работы элементов с фиксированной позицией на не самых новых Android-устройствах. Более подробные сведения и описание причин происходящего можно найти в публикациях по адресам http://benfrain.com/easy-css-fix-fixed-positioning-android-2-2-2-3/ и http://benfrain.com/improving-css-performance-fixed-position-elements/.

Затем нам нужно заставить заднюю панель быть в исходном состоянии перевернутой, чтобы при переворачивании всего элемента она оказалась в правильном положении. Для этого применяется преобразование вращения:

```
.flipper-vertical .back {
    transform: rotateX(180deg);
}
.flipper-horizontal .back {
    transform: rotateY(180deg);
}
```

Теперь все на своих местах и все, что мы хотим сделать, — это перевернуть весь внутренний элемент, когда на внешний элемент наводится указатель мыши:

```
.flipper:hover .flipper-vertical {
    transform: rotateX(180deg);
}
.flipper:hover .flipper-horizontal {
    transform: rotateY(180deg);
}
```

Как вы можете представить, существует несметное количество (кстати, это преувеличение, я проверял) способов применения главных компонентов этой конструкции. Чтобы поинтересоваться с точки зрения перспективы, как выглядит забавный навигационный эффект или меню, которое оказывается за пределами основного холста, я рекомендую посетить сайт Codrops: http://tympanus.net/Development/PerspectivePageViewNavigation/index.html.

СОВЕТ

Самые последние наработки консорциума W3C в отношении CSS Transforms Module Level 1 можно найти по адресу http://dev.w3.org/csswg/css-transforms/.

Свойство transform3d

Вдобавок к использованию перспективы я выявил немалую пользу от применения значения `transform3d`. Только одно это свойство и его значение позволяют перемещать элемент по осям *X* (влево-вправо), *Y* (вверх-вниз) и *Z* (вперед-назад). Внесем коррективы в последний пример и воспользуемся преобразованием `translate3d`. Код этого примера можно найти в файле каталога example_08-06.

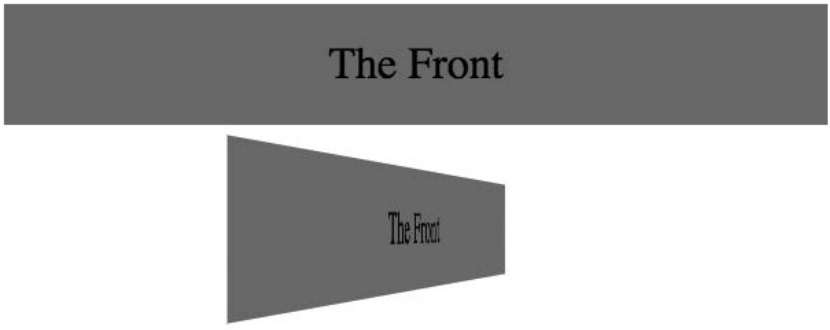
Кроме установки элементов с небольшим отступом, изменения по сравнению с предыдущим примером можно найти только здесь:

```
.flipper:hover .flipper-vertical {
    transform: rotateX(180deg) translate3d(0, 0, -120px);
    animation: pulse 1s 1s infinite alternate both;
}

.flipper:hover .flipper-horizontal {
    transform: rotateY(180deg) translate3d(0, 0, 120px);
    animation: pulse 1s 1s infinite alternate both;
}
```

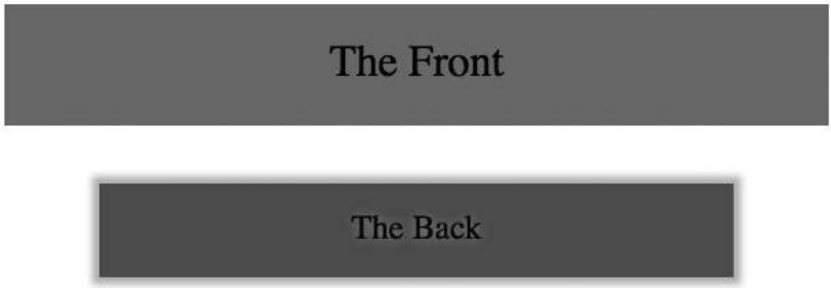
Мы по-прежнему применяем преобразование, но на этот раз добавили к вращению `translate3d`. Синтаксис включает в себя помещаемые внутрь объявления `translate3d` аргументы, отделенные друг от друга запятыми и задающие перемещение по осям *X*, *Y* и *Z*.

В наших двух примерах я не перемещал элемент по осям *X* или *Y* (слева направо или вверх и вниз), зато с вашей точки наблюдения перемещал его по направлениям к вам и от вас.

Если вы посмотрите на верхний пример, то увидите, что переворот элемента происходит за нижней кнопкой и завершается в позиции, находящейся на 120 пикселов ближе к плоскости экрана (отрицательные значения приведут к отдалению элемента от вас).



В то же время нижняя кнопка переворачивается по горизонтали и в итоге оказывается отстоящей от вас на 120 пикселов.



СОВЕТ

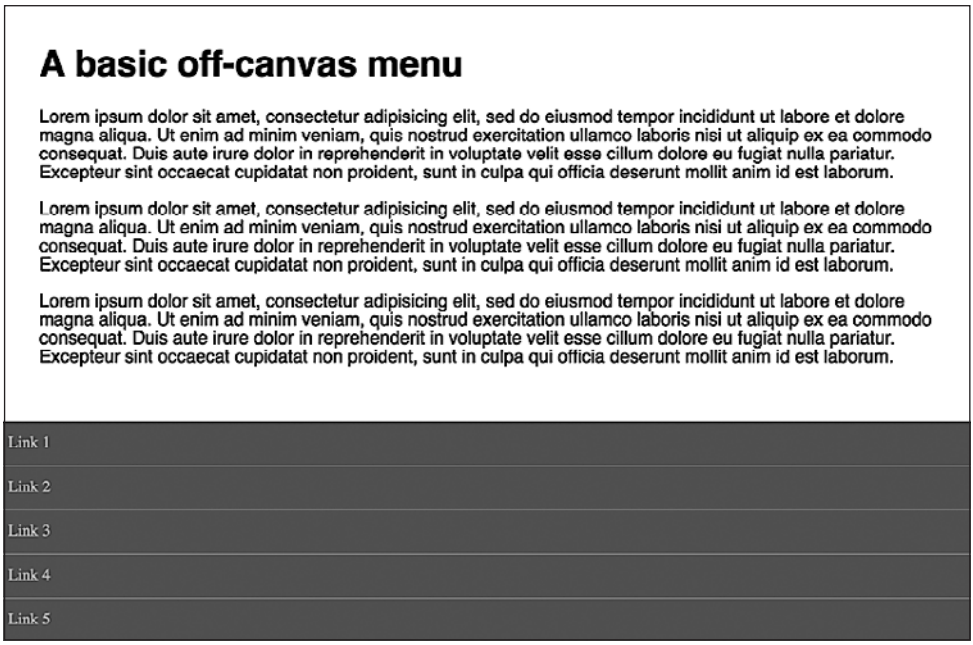
Спецификацию, касающуюся translate3d, можно найти по адресу http://www.w3.org/TR/css3-3d-transforms/.

Использование преобразований при постепенном усложнении

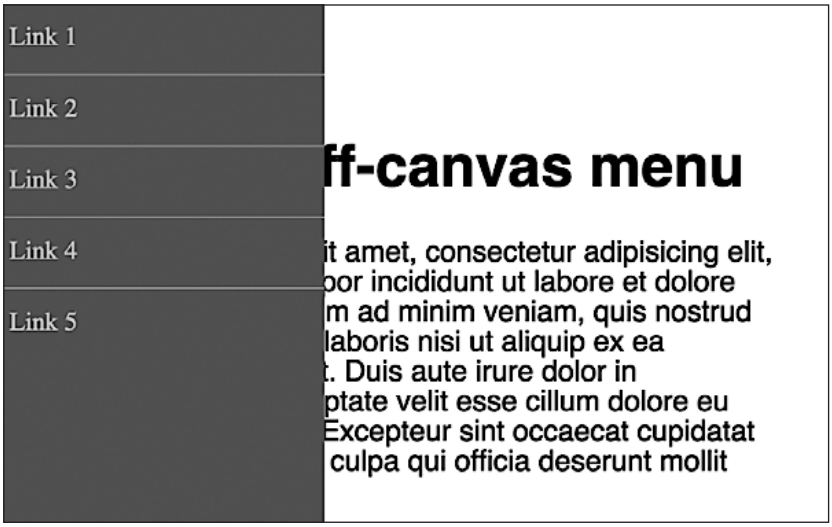
Область, в которой, на мой взгляд, свойство `transform3d` проявило себя с наилучшей стороны, — это выдвигаемые на экран и задвигаемые за его пределы панели, в частности модели, располагающие элементы навигации за пределами холста. Если вы откроете файл, находящийся в каталоге example_08-07, то увидите, что мною создана основная постепенно усложняемая модель за пределами холста.

При создании системы взаимодействия с помощью JavaScript и с использованием современных свойств CSS, подобных преобразованиям, появляется смысл испытывать компоненты, начиная с требующих поддержки устройств с самыми скромными возможностями. А что делать в расчете на тех, у кого нет JavaScript (да, попадаются еще и такие), или на тех, кто испытывает проблемы с загрузкой или выполнением кода JavaScript? А что делать, если чьи-нибудь устройства не поддерживают преобразования (устройства, использующие, к примеру, Opera Mini)? Не волнуйтесь, вы можете без особых усилий обеспечить работоспособный интерфейс для любой возможности.

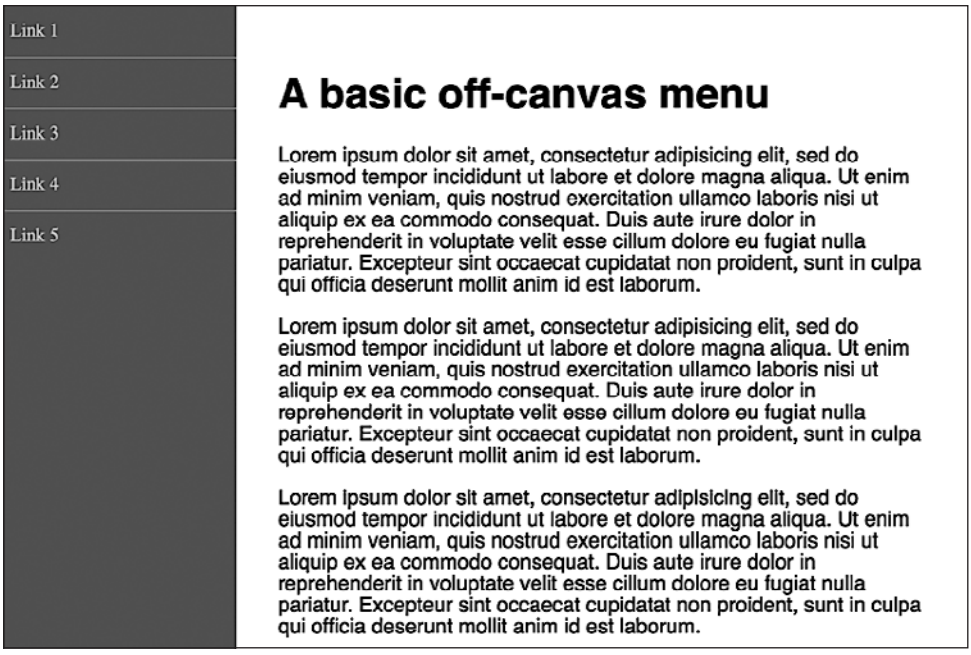
Я пришел к выводу, что при создании таких моделей интерфейсов полезнее всего начинать с самого скромного набора возможностей и выполнять усложнение, начиная с этой исходной позиции. Сначала установим, что увидят те, у кого недоступен JavaScript. Как бы там ни было, если метод для отображения меню зависит от наличия JavaScript, то вывод меню за пределы экрана не используется. В таком случае для размещения области навигации в обычном потоке элементов документа мы полагаемся на разметку. В худшем случае независимо от ширины окна просмотра пользователи смогут просто прокрутить страницу, добравшись до самого низа, и щелкнуть на ссылке.



Если JavaScript доступен, то при меньших экранах мы убираем меню влево. При щелчке на кнопке меню мы добавляем класс к тегу `body` (с помощью JavaScript) и используем его в качестве зацепки для возвращения навигационной панели в поле зрения с помощью CSS.



Для более крупных окон просмотра мы прячем кнопку меню и просто располагаем панель навигации слева, а основное содержимое перемещаем так, чтобы им было занято все остальное пространство.



Затем эффект скрытия-показа панели навигации постепенно усложняется. Здесь свое вполне заслуженное место занимает средство Modernizr: добавляя классы к HTML-тегу, мы можем использовать их как зацепки для стилевого оформления (Modernizr более подробно рассматривается в главе 5).

Сначала для браузеров, поддерживающих только преобразования, связанные с перемещениями (например, старые Android-браузеры), используется простое свойство `translateX`:

```
.js .csstransforms .navigation-menu {
    left: auto;
    transform: translateX(-200px);
}
```

Для браузеров, поддерживающих трехмерные преобразования, используется свойство `translate3d`. При этом там, где оно поддерживается, ожидается существенный прирост производительности, поскольку на большинстве устройств выполнение решаемых при его объявлении задач возлагается на графические процессоры:

```
.js .csstransforms3d .navigation-menu {
    left: auto;
    transform: translate3d(-200px, 0, 0);
}
```

Использование подхода с постепенным усложнением гарантирует положительное восприятие работоспособности ваших конструкций со стороны самой широкой аудитории. Помните о том, что ваши пользователи не нуждаются в визуальном равенстве, но могут оценить соответствие возможностям.

Создание эффектов анимации средствами CSS3

Если вам приходилось работать с приложениями Flash, Final Cut Pro или After Effects, то вы сразу же почувствуете преимущества, получаемые при работе с CSS3-анимацией. В CSS3 применяется методика анимации с использованием ключевых кадров, встречающаяся в приложениях, построенных на использовании шкалы времени.

Анимация весьма широкой реализуется в браузерах и поддерживается Firefox 5+, Chrome, Safari 4+, Android (всеми версиями), iOS (всеми версиями) и Internet Explorer 10+. Анимация в CSS3 содержит два компонента: прежде всего это объявление ключевых кадров, а затем их использование в свойстве `animation`. Посмотрим, как все это работает.

В предыдущем примере создавался простой эффект переворачивания элементов путем сочетания преобразований и переходов. Давайте соберем воедино все технологии, изученные в этой главе, и добавим к этому примеру анимацию. В следующем примере, код которого можно найти в файле каталога example_08-05, добавим анимационный эффект пульсации в процессе переворачивания элемента.

Сначала создадим правило `@keyframes`:

```
@keyframes pulse {
    100% {
        text-shadow: 0 0 5px #bbb;
        box-shadow: 0 0 3px 4px #bbb;
    }
}
```

Как видите, за написанием `@keyframes` для определения нового эт-правила `keyframes` следует имя конкретной анимации (в данном примере `pulse`).

Вообще-то лучше использовать имя, представляющее действие, производимое анимацией, а не место, к которому вы собираетесь ее применить, поскольку одно и то же правило `@keyframes` может использоваться по всему проекту необходимое количество раз.

Здесь мы использовали только один селектор ключевого кадра — `100%`. Но в правиле `keyframes` таких селекторов можно указать сколько угодно, определив их в виде процентных отметок. Эти отметки нужно представлять себе находящимися на шкале времени. Например, на 10 % следует сделать фон синим, на 30 % — лиловым, а на 60 % элемент должен стать полупрозрачным и т. д., как вы сочтете нужным. Имеется также ключевое слово `from`, эквивалентное 0 %, и `to`, эквивалентное 100 %. Их можно использовать следующим образом:

```
@keyframes pulse {
    to {
        text-shadow: 0 0 5px #bbb;
        box-shadow: 0 0 3px 4px #bbb;
    }
}
```

Но следует иметь в виду, что браузеры, основанные на движке WebKit (iOS, Safari), не всегда понимают значения `from` и `to`, предпочитая им `0%` и `100%`, поэтому я рекомендую остановить свой выбор на процентном обозначении селекторов ключевых кадров.

Как вы заметили, мы не стали определять стартовое положение. Дело в том, что оно совпадает с тем состоянием, в котором уже находится каждое из этих свойств. Ту часть спецификации, в которой все это объясняется, можно найти по адресу http://www.w3.org/TR/css3-animations/.

ПРИМЕЧАНИЕ

Если не определен ключевой кадр для 0 % или from, то агент пользователя создает ключевой кадр для 0 %, используя вычисленные значения анимируемых свойств. Если не определен ключевой кадр для 100 % или to, то агент пользователя создает ключевой кадр для 100 %, используя вычисленные значения анимируемых свойств. Если в селекторе ключевого кадра указано отрицательное процентное значение или значение, превышающее 100 %, то ключевой кадр будет проигнорирован.

В эт-правиле `keyframes` мы добавили к селектору 100 % свойства `text-shadow` и `box-shadow`, после чего ожидаем, что ключевые кадры, будут применены к элементу для анимации `text-shadow` и `box-shadow` до определенной величины. Но сколько будет длиться анимация? Как заставить ее повториться, пойти вспять и продемонстрировать другие возможности, по поводу которых я надеюсь получить ответ? Вот как происходит применение анимации на основе ключевых кадров:

```
.flipper:hover .flipper-horizontal {
    transform: rotateY(180deg);
    animation: pulse 1s 1s infinite alternate both;
}
```

Здесь для краткой формы записи ряда связанных с анимацией свойств используется свойство `animation`. В данном примере объявляются (по порядку) имя используемого объявления ключевых кадров (`pulse`), продолжительность анимации (одна секунда), задержка перед началом анимации (одна секунда, чтобы дать нашей кнопке время на первый переворот), количество запусков анимации (бесконечно — `infinite`), направление анимации (чередование — `alternate`, то есть анимация сначала идет в одном направлении, а затем его меняет), а затем нам нужно, чтобы свойство `animation-fill-mode` сохраняло значения, определенные в ключевых кадрах, когда анимация собирается идти вперед или назад (`both`).

Сокращенная запись может фактически принять все семь свойств анимации. Вдобавок к тому что было использовано в предыдущем примере, можно указать значение свойства `animation-play-state`. Для эффективного проигрывания или остановки анимации ему может быть присвоено значение состояния работы — `running` или состояния на паузе — `paused`. Разумеется, использовать сокращенную запись совсем не обязательно, иногда имеет смысл установить каждое свойство по отдельности, что поможет разобраться в коде при его последующем изучении. Далее показаны отдельные свойства и там, где это возможно, их альтернативные значения, отделенные друг от друга вертикальной чертой:

```
.animation-properties {
    animation-name: warning;
    animation-duration: 1.5s;
    animation-timing-function: ease-in-out;
    animation-iteration-count: infinite;
    animation-play-state: running | paused;
    animation-delay: 0s;
    animation-fill-mode: none | forwards | backwards | both;
    animation-direction: normal | reverse | alternate | alternate-reverse;
}
```

ПРИМЕЧАНИЕ

Полное определение каждого из этих свойств анимации можно найти по адресу http://www.w3.org/TR/css3-animations/.

Как уже упоминалось, объявленные ключевые кадры использовать в отношении других элементов и с совершенно другими установками довольно просто:

```
.flipper:hover .flipper-vertical {
    transform: rotateX(180deg);
    animation: pulse 2s 1s cubic-bezier(0.68, -0.55, 0.265, 1.55) 5
    alternate both;
}
```

Здесь `pulse`-анимация запустится на две секунды и будет использовать функцию развития процесса по времени `ease-in-out-back`, определенную в виде кривой Безье третьего порядка. Анимация будет повторяться в обе стороны пять раз. Это объявление применено в файле примера к вертикально переворачиваемому элементу.

Это всего лишь один простой пример использования CSS-анимации. Так как на ключевые кадры можно разложить практически все, перед нами открываются, по сути, безграничные возможности. О последних разработках в области CSS3-анимации можно узнать по адресу http://dev.w3.org/csswg/css3-animations/.

Свойство animation-fill-mode property. Свойство `animation-fill-mode` заслуживает специального упоминания. Рассмотрим анимацию, начинающуюся с фона желтого цвета, который переходит в фон красного цвета за три секунды. Код этого примера можно увидеть в файле каталога example_08-08.

Здесь применяется следующая анимация:

```
.background-change {
    animation: fillBg 3s;
    height: 200px;
    width: 400px;
    border: 1px solid #ccc;
}

@keyframes fillBg {
    0% {
```

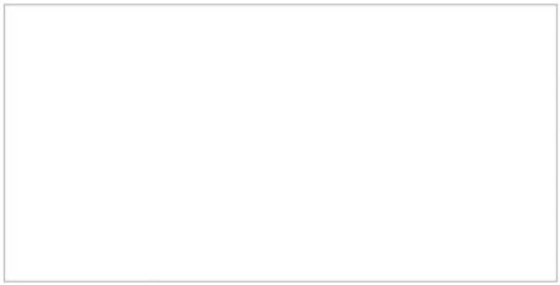
```
        background-color: yellow;
    }
    100% {
        background-color: red;
    }
}
```

Но как только анимация завершится, фон контейнера `div` вернется к бесцветному состоянию. Это потому, что по умолчанию все, что происходит за пределами анимации, остается за ее пределами! Чтобы изменить такое поведение, у нас есть свойство `animation-fill-mode`. В данном примере мы можем воспользоваться следующим кодом:

```
animation-fill-mode: forwards;
```

Тогда элемент сможет сохранить любые значения, примененные в конце анимации. В нашем случае в контейнере `div` сохранится фон красного цвета, которым заканчивается анимация. Дополнительные сведения о свойстве `animation-fill-mode` можно найти по адресу http://www.w3.org/TR/css3-animations/#animation-fill-mode-property.

Без использования свойства animation-fill-mode



С использованием свойства animation-fill-mode

Резюме

Описаниям преобразований, переходов и анимации средствами CSS можно посвятить целые книги. Но надеюсь, что даже поверхностное ознакомление в этой главе с данным вопросом позволит вам усвоить основы и приступить к практическому

использованию этих эффектов. В конечном счете освоение новейших возможностей и технологий CSS нацелено на придание адаптивной конструкции еще большей компактности и изысканности путем использования средств CSS, а не кода JavaScript и внесение в нее ряда более привлекательных эстетических усовершенствований.

Из этой главы мы узнали, что такое CSS3-преобразования и как создается код для их реализации. Мы научились пользоваться функциями развития процесса по времени `ease` и `linear`, а затем применили их для создания простых, но весьма интересных эффектов. Затем было изучено все, что касается 2D-преобразований, в том числе масштабирование и наклон, после чего рассмотрены способы их использования в связке с переходами. Вкратце были рассмотрены 3D-преобразования, а затем изучено все, что касается эффективности и относительной простоты CSS-анимации. И трудно не согласиться с тем, что ваши возможности по части использования CSS3 существенно возросли!

Но есть одна область в разработке сайтов, которую я всеми силами стараюсь обойти стороной, — это создание форм. Не знаю почему, но мне всегда казалось, что их создание является весьма нудной и зачастую не приносящей полного удовлетворения работой. Представьте себе, как я обрадовался, когда узнал, что HTML5 и CSS3 могут как никогда ранее упростить процесс создания, стилевого оформления и даже проверки правильности заполнения форм. У меня появилось желание заняться этим вопросом. Подобный прилив энтузиазма в части создания веб-форм можете испытать и вы. Этими знаниями я готов поделиться с вами в следующей главе.

9 Обуздание форм с помощью HTML5 и CSS3

До появления HTML5 для добавления в формы таких компонентов, как панели выбора даты, замещаемый текст и ползунки диапазонов, всегда требовалось применение кода JavaScript. Кроме того, не было простого способа довести до пользователя, что именно ожидается вводить в конкретные поля ввода, например, хотим ли мы от пользователя ввода телефонных номеров, адресов электронной почты или URL-адресов. Теперь же мы можем порадоваться тому, что во многом эти задачи решаются средствами HTML5.

В этой главе ставятся две основные цели: во-первых, освоить особенности создания форм в HTML5, во-вторых, упростить разметку форм для множества устройств с применением самых последних возможностей CSS.

В этой главе вы научитесь:

- легко и просто добавлять замещаемый текст в соответствующие поля ввода формы;
- отключать при необходимости автозавершение в полях формы;
- устанавливать обязательность заполнения конкретных полей перед отправкой формы;
- указывать различные типы ввода, например электронные адреса, телефонные номера и URL-адреса;
- создавать ползунки диапазонов чисел для упрощенного выбора значений;
- помещать в форму панели выбора дат и цветовых решений;
- использовать регулярные выражения для определения в формах допустимых значений;
- создавать стилевое оформление форм с помощью Flexbox.

Формы HTML5

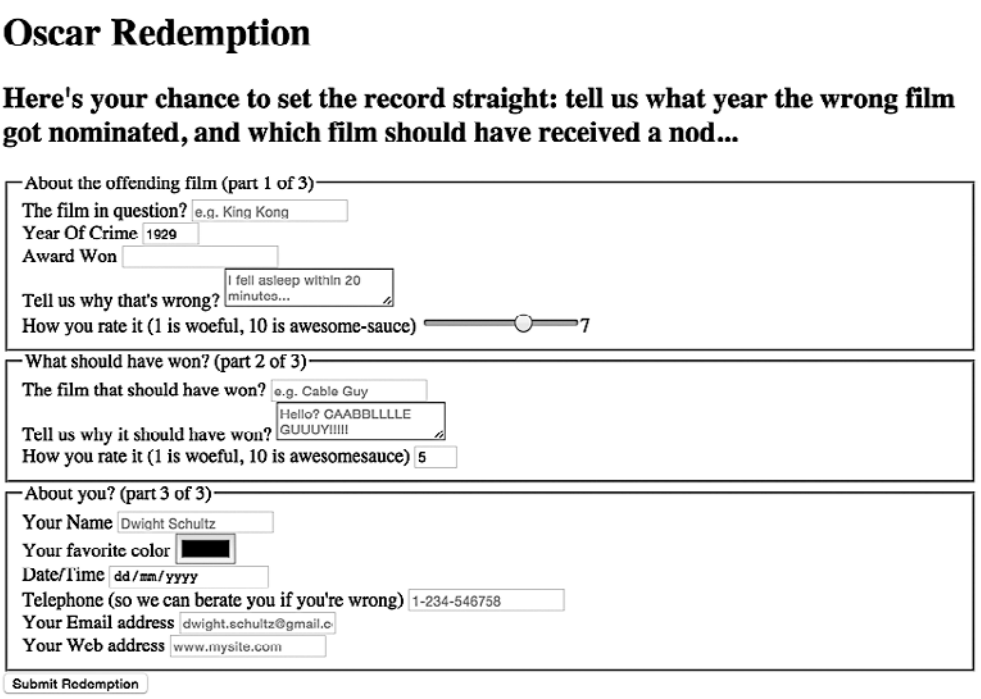
Полагаю, что легче всего освоить формы HTML5 путем проработки конкретного примера. Из лучших примеров дневных телепередач у меня есть в запасе один ранее припасенный сценарий. Здесь нужно сделать небольшое введение.

Есть два обстоятельства: во-первых, я люблю кино, во-вторых, у меня всегда собственное мнение о том, какой фильм хороший, а какой — нет.

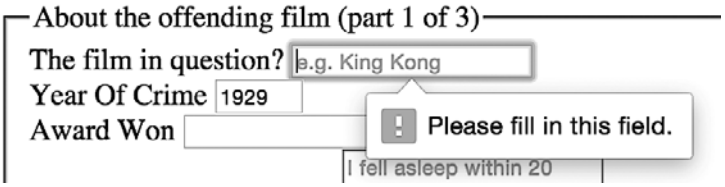
Каждый год, когда объявляют номинантов на премию «Оскар», я не могу избавиться от ощущения, что одобрение академии получают совсем не те фильмы. Поэтому мы начнем с формы HTML5, позволяющей киноманам выразить свое недовольство постоянным выдвижением недостойных фильмов в номинацию на эту премию.

Форма состоит из нескольких элементов `fieldset`, в которые мы включим основные типы ввода и атрибуты формы HTML5. Кроме стандартных полей ввода формы и областей ввода текста, у нас будут поле ввода чисел с возможностью прокрутки их последовательности, ползунок диапазона и замещаемый текст для многих полей.

Вот как эта форма выглядит в браузере Chrome без примененных к ней стилей.



Если установить фокус на первое поле и приступить к вводу текста, замещаемый текст исчезнет. Если убрать фокус и оставить поле нетронутым, ее раз щелкнув за пределами поля ввода, замещаемый текст останется в поле. Если отправить данные формы, не вводя вообще ничего, произойдет следующее.



Самое интересное, что все эти элементы пользовательского интерфейса, включая вышеупомянутые ползунок, замещаемый текст и поле с прокруткой чисел, а также функция проверки допустимости введенных данных реализуются самим браузером с применением HTML5 и без участия кода JavaScript. Пока что функция проверки правильности заполнения формы не имеет всесторонней поддержки со стороны браузеров, но вскоре этот недостаток будет устранен. Прежде всего разберемся со всеми новыми свойствами HTML5, имеющими отношение к формам и предоставляющими все эти возможности. Как только мы поймем всю механику, можно будет перейти к приданию форме стилевого оформления.

Основные сведения о компонентах формы HTML5

В нашей форме HTML5 содержится множество улучшений, поэтому разберемся со всем этим по порядку. У нас имеются три раздела формы, каждый из которых заключен в набор полей `fieldset` с соответствующей легендой `legend`:

```
<fieldset>
<legend>About the offending film (part 1 of 3)</legend>
<div>
    <label for="film">The film in question?</label>
    <input id="film" name="film" type="text" placeholder="e.g. King Kong" required>
</div>
```

В предыдущем фрагменте кода можно увидеть, что каждый элемент ввода формы также заключен в контейнер `div` с надписью, связанной с каждым полем ввода (при желании элемент ввода тоже можно заключить в элемент надписи). Пока что все как всегда. Но внутри этого первого элемента ввода нам уже встречается самая первая особенность формы, создаваемой с помощью HTML5. После обычных атрибутов `ID`, `name` и `type` появился атрибут `placeholder`.

placeholder

Атрибут `placeholder` имеет следующий вид:

```
placeholder="e.g. King Kong"
```

Текст-заместитель в полях формы стал общепринятым требованием, и создатели HTML5 решили, что он должен стать стандартным свойством HTML. Просто включите атрибут `placeholder` в поле ввода, и его значение будет по умолчанию показываться в поле до тех пор, пока в него не начнется ввод данных. Если данные в поле не введены, а фокус переведен в другое место, текст-заместитель останется в поле.

Придание стиля тексту-заместителю. Придать стиль тексту атрибута `placeholder` можно с помощью псевдоселектора `:placeholder-shown`. Следует иметь в виду, что этот селектор претерпел множество переработок, поэтому, чтобы задействовать его альтернативные варианты для уже реализованных версий, воспользуйтесь инструментальным средством, добавляющим префиксы производителей:

```
input:placeholder-shown {
    color: #333;
}
```

В предыдущем фрагменте кода следующей особенностью формы HTML5 после атрибута `placeholder` выступает атрибут `required`.

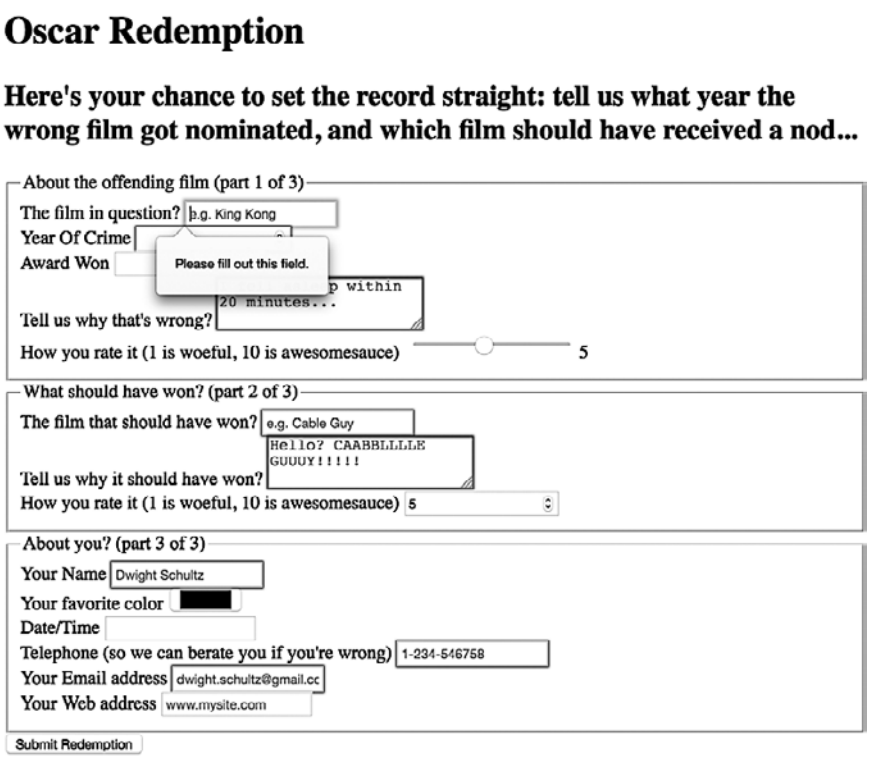
required

Атрибут `required` имеет следующий вид:

```
required
```

Добавление булева атрибута (это означает, что вы либо включаете этот атрибут, либо нет) `required` в элемент ввода в тех браузерах, которые поддерживают HTML5, показывает, что значение в поле ввода является обязательным. Если форма будет отправлена без заполненного поля, в котором должна содержаться обязательная информация, на экран будет выведено предупредительное сообщение. Каким именно будет выводимое на экран сообщение (как по текстовому содержанию, так и по оформлению), зависит как от применяемого браузера, так и от типа незаполненного поля ввода.

Мы уже видели, как выглядит сообщение, касающееся обязательного для заполнения поля, в браузере Chrome. А следующая копия экрана показывает, как такое же сообщение выглядит в браузере Firefox.



Чтобы гарантировать ввод данных, значение required можно использовать совместно с полями для ввода многих типов данных. Исключениями являются элементы ввода range, color, button и скрытые поля ввода, поскольку они практически всегда должны иметь значение по умолчанию.

autofocus

HTML5-атрибут autofocus позволяет иметь в форме поле, готовое к пользовательскому вводу, на которое уже установлен фокус. Следующий код является примером поля ввода, заключенного в контейнер div с добавленным в конце атрибутом autofocus:

```
<div>
    <label for="search">Search the site...</label>
    <input id="search" name="search" type="search" placeholder="Wyatt Earp"
        autofocus>
</div>
```

При использовании этого атрибута нужно проявлять осмотрительность. Если атрибут autofocus добавлен сразу к нескольким полям, браузеры могут прореагировать на это по-разному. Например, если autofocus добавлен к нескольким полям, то при загрузке страницы в Safari фокус получит последнее из полей с автофокусом. А вот в Firefox и Chrome все получится наоборот и фокус получит первое из таких полей.

Стоит также принять в расчет то, что для быстрого прохождения по содержимому веб-страницы после ее загрузки некоторые пользователи используют клавишу пробела. На странице с формой, у которой установлен автофокус, такой возможности не будет, вместо этого символ пробела будет добавлен в то поле ввода, у которого будет фокус. Нетрудно догадаться, что ничего, кроме раздражения, это у пользователей не вызовет.

При использовании атрибута autofocus нужно убедиться в том, что он используется в форме только один раз и вы представляете себе последствия его применения для тех, кто пользуется клавишей пробела для прокрутки страницы.

autocomplete

По умолчанию большинство браузеров помогают пользователю вводить данные с применением функции автозавершения значений в полях формы там, где это возможно. Хотя пользователь сам может отключить или включить эту функцию в браузере, теперь мы также можем показать браузеру, что не желаем применения функции автозавершения для формы или поля. Это пригодится не только при вводе важных данных (например, номеров банковских счетов), но и при желании обратить внимание пользователя и заставить его ввести что-либо вручную. Например, во многих заполняемых мною формах там, где требовался номер телефона, я вводил вымышленный номер. Я знал, что так поступаю не я один (вы ведь тоже так делали?), но путем установки в соответствующем поле ввода значения off для атрибута autocomplete я могу гарантировать, что пользователи не введут фиктивный номер, часть которого за них введет функция автозавершения. В следующем примере кода показано поле с атрибутом autocomplete, установленным в off:

```
<div>
    <label for="tel">Telephone (so we can berate you if you're wrong)</label>
    <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
        autocomplete="off" required>
</div>
```

Автозавершение можно отключить также для всей формы (но не для набора полей), воспользовавшись в объявлении формы атрибутом `autocomplete`:

```
<form id="redemption" method="post" autocomplete="off">
```

Атрибут list и связанный с ним элемент datalist

Атрибут `list` и связанный с ним элемент `datalist` позволяют ряду вариантов выбора быть представленными пользователю сразу же, как только он начнет вводить значение в поле. В следующем примере кода атрибут `list` используется вместе со связанным с ним элементом `datalist`, и все это заключено в контейнер `div`:

```
<div>
    <label for="awardWon">Award Won</label>
    <input id="awardWon" name="awardWon" type="text" list="awards">
    <datalist id="awards">
        <select>
            <option value="Best Picture"></option>
            <option value="Best Director"></option>
            <option value="Best Adapted Screenplay"></option>
            <option value="Best Original Screenplay"></option>
        </select>
    </datalist>
</div>
```

Значение, указываемое для атрибута `list` (в данном случае это `awards`), является ссылкой на идентификатор элемента `datalist`. Таким способом `datalist` привязывается к полю ввода. Хотя здесь варианты заключены в элемент `<select>`, в его использовании нет особой необходимости, но он помогает при применении полифиллов для тех браузеров, у которых это свойство не реализовано.

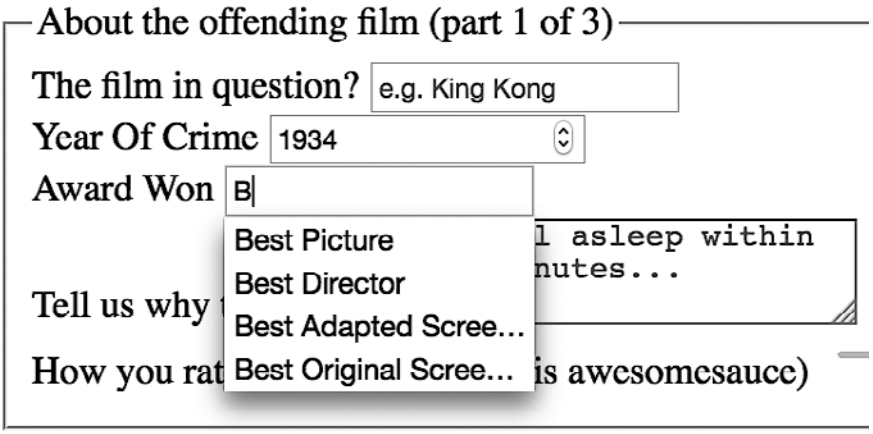
ПРИМЕЧАНИЕ

Как ни удивительно, но к середине 2015 года элемент datalist все еще не поддерживался естественным образом в iOS, Safari и Android 4.4 и более ранних версиях (см. сайт http://caniuse.com/).

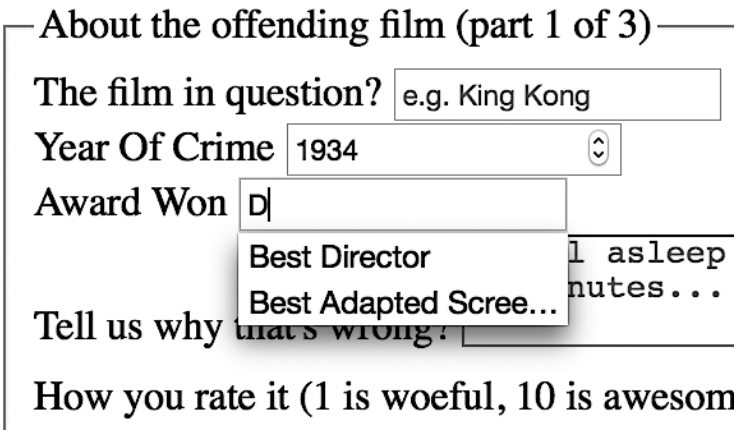
Спецификацию по datalist можно найти по адресу http://www.w3.org/TR/html5/forms.html.

Хотя это поле ввода ничем не отличается от обычного поля, предназначенного для ввода текста, при наборе в нем данных под ним появляется окно выбора (в тех браузерах, где это свойство поддерживается) с наиболее подходящими результатами из перечня, указанного в `datalist`. Работа атрибута `list` (в браузере Firefox) показана в следующей копии экрана.

В данном примере, поскольку буква «B» присутствует во всех вариантах, перечисленных в `datalist`, пользователю для выбора показываются все варианты.



Но если набрать вместо этого букву «D», то будут выведены только те варианты, которые могут подойти.



Атрибут `list` и элемент `datalist` не мешают пользователю ввести в поле какой-нибудь другой текст, но они предоставляют еще один полезный способ добавления общих функциональных возможностей и улучшения условий работы пользователя, применяя для этого исключительно разметку HTML5.

Типы вводимой информации, определяемые в HTML5

В HTML5 добавлен ряд дополнительных типов вводимой информации, что, кроме всего прочего, позволяет нам наложить ограничения на данные, вводимые пользователями, не прибегая при этом к применению внешнего кода JavaScript.

Удобнее всего в этих новых типах то, что браузеры, изначально их не поддерживающие, превращают соответствующие поля в обычные текстовые поля ввода. Кроме того, доступны отличные полифиллы, на которые мы также вскоре обратим внимание, выводящие старые браузеры на уровень их современных собратьев. А пока посмотрим на новые типы вводимой информации, определяемые в HTML5, и разберемся с предоставляемыми ими преимуществами.

email

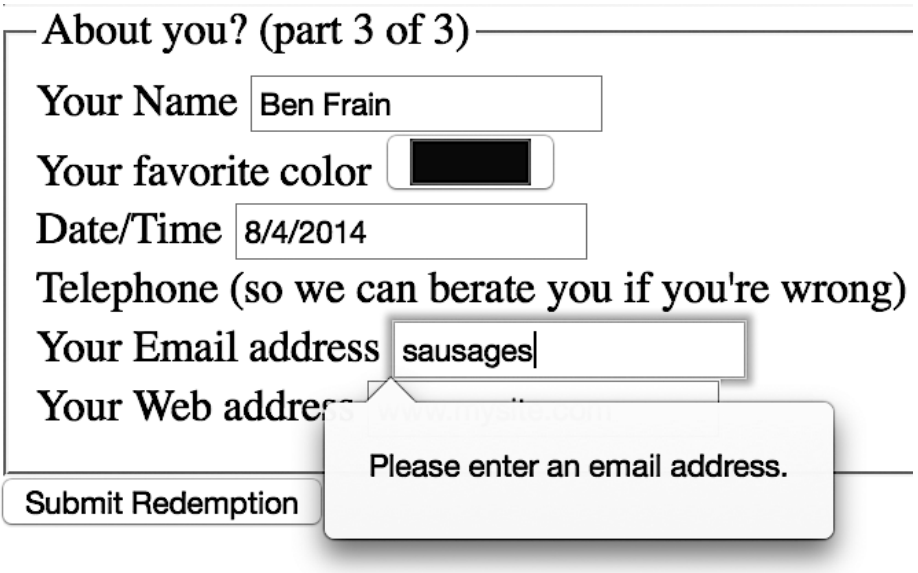
Установить для поля ввода тип данных `email` можно следующим образом:

```
type="email"
```

Поддерживающие это свойство браузеры будут ожидать пользовательского ввода, соответствующего синтаксису электронного адреса. В следующем примере кода объявление `type="email"` используется вместе с атрибутами `required` и `placeholder`:

```
<div>
    <label for="email">Your Email address</label>
    <input id="email" name="email" type="email"
        placeholder="dwight.schultz@gmail.com" required>
</div>
```

При связке с атрибутом `required` отправка несоответствующих данных вызовет появление предупреждающего сообщения.



Кроме того, многие устройства с сенсорным экраном (например, Android, iPhone и т. д.) на основе данного типа вводимой информации изменяют экран ввода. В следующей копии экрана показано, как экран с настройкой ввода `type="email"` выгля-

дит на iPad. Обратите внимание на то, что для упрощения ввода адреса электронной почты на программную клавиатуру был добавлен символ @.

number

Установить для поля ввода тип данных number можно следующим образом:

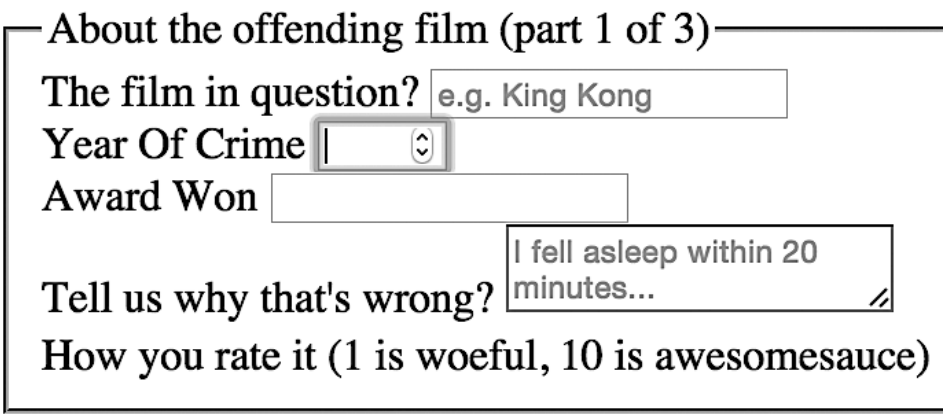
```
type="number"
```

Браузеры, поддерживающие это свойство, ожидают ввода числа. А браузеры, поддерживающие поле ввода чисел с возможностью прокрутки их последовательности, предоставляют еще и это средство выбора. Речь идет о небольшом фрагменте пользовательского интерфейса, позволяющем пользователям для изменения вводимого значения просто нажимать клавиши со стрелками на клавиатуре или щелкать указателем мыши на стрелках вверх и вниз. Рассмотрим пример кода:

```
<div>
    <label for="yearOfCrime">Year Of Crime</label>
    <input id="yearOfCrime" name="yearOfCrime" type="number"
           min="1929" max="2015" required>
</div>
```

А на следующей копии экрана показано, как это выглядит в поддерживающем данное свойство браузере Chrome.

Существуют разные варианты реакции браузеров на оставшиеся пустыми поля для числовых значений. К примеру, Chrome и Firefox ничего не делают, пока форма не отправлена, а при попытке ее отправки выводят предупреждающее сообщение о незаполненном поле. А Safari вообще ничего не делает и позволяет отправить форму. Internet Explorer 11 просто оставляет поле пустым, как только оно теряет фокус.



Диапазоны min и max

В предыдущем примере кода можно было заметить, что мы также установили диапазон допустимых минимальных и максимальных значений, использовав для этого код, похожий на следующий:

```
type="number" min="1929" max="2015"
```

Числа, выходящие за границы этого диапазона, должны проходить специальную обработку.

Сведения о том, что существует разная реализация диапазонов `min` и `max` в браузерах, вас уже вряд ли удивят. Например, Internet Explorer 11, Chrome и Firefox выдают предупреждения, а Safari не выдает.

Изменение шагов приращения

Вы можете изменить шаг приращения (степень детализации) элементов управления с прокручиваемыми числовыми значениями, воспользовавшись атрибутом `step`. Например, прокрутку с шагом десять единиц можно получить благодаря следующему объявлению:

```
<input type="number" step="10">
```

url

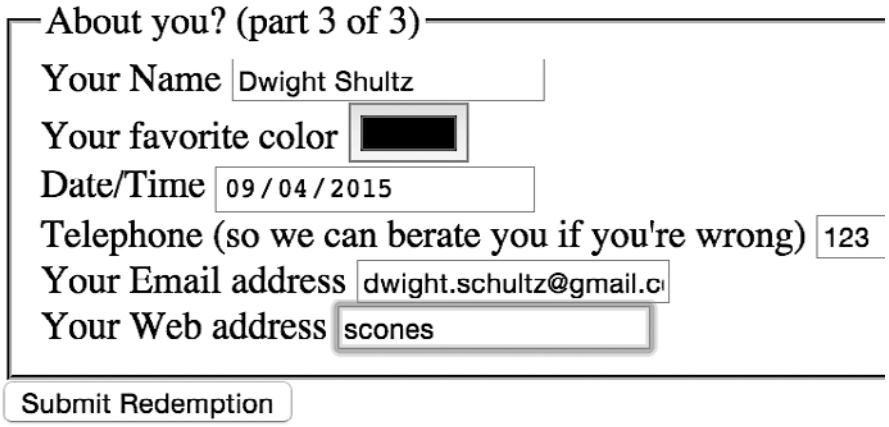
Настроить поле ввода на ожидание URL-адреса можно следующим образом:

```
type="url"
```

Нетрудно догадаться, что тип вводимых данных `url` предназначен для ввода значений URL-адресов. Он похож на типы вводимых данных `tel` и `email` и приводит к настройке на поведение, практически аналогичное поведению стандартного поля ввода текста. Но некоторые браузеры добавляют к предупреждающему сообщению специальную информацию, предоставляемую в случае отправки некорректных значений. В следующий пример кода включен также атрибут `placeholder`:

```
<div>
<label for="web">Your Web address</label>
<input id="web" name="web" type="url" placeholder="www.mysite.com">
</div>
```

На показанной далее копии экрана можно увидеть, что произойдет при некорректном заполнении поля URL-адреса и попытке отправки формы в Chrome.



Здесь так же, как и при объявлении `type="email"`, устройства с сенсорным экраном зачастую вносят изменения в экран ввода на основе данного типа вводимых данных. На следующей копии экрана показано, как выглядит экран ввода устройства iPad при объявлении `type="url"`.

Заметили клавишу .com? Поскольку был использован тип вводимых данных URL, такие клавиши предоставляются устройством, чтобы было проще выполнить

ввод URL-адреса (если на устройствах под управлением iOS нужен сайт в домене, отличном от .com, можно нажать и удерживать клавишу до появления нескольких других популярных доменов верхнего уровня).

tel

Настроить поле ввода на ожидание телефонного номера можно следующим образом:

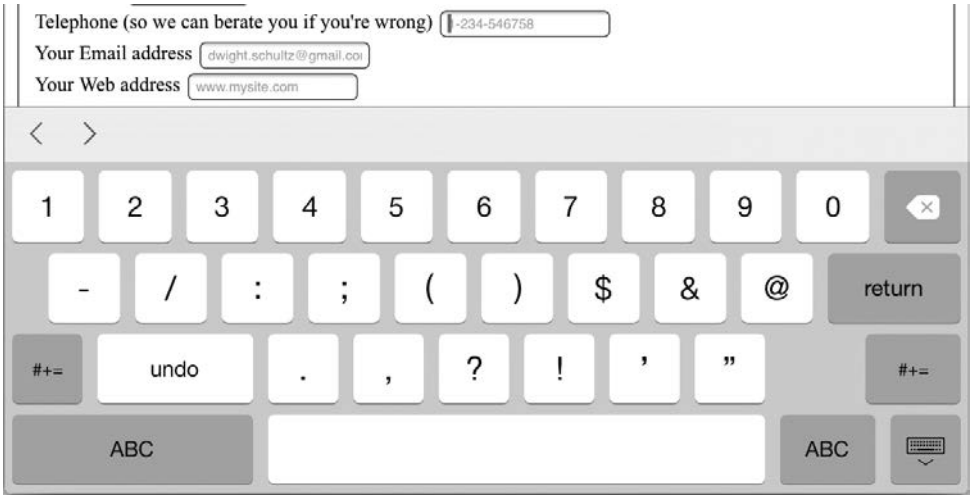
```
type="tel"
```

А вот как выглядит более сложный пример:

```
<div>
    <label for="tel">Telephone (so we can berate you if you're wrong)</label>
    <input id="tel" name="tel" type="tel" placeholder="1-234-546758"
     autocomplete="off" required>
</div>
```

Хотя числовой формат ожидается во многих браузерах, даже самые современные и востребованные из них, такие как Internet Explorer 11, Chrome и Firefox, ведут себя так, как будто это поле ввода текста. Когда введено неверное значение, они не выдают соответствующее предупреждающее сообщение ни при потере полем фокуса, ни при отправке данных формы.

Но есть и хорошие новости: так же, как и в случае с типами вводимых данных `email` и `url`, устройства с сенсорными экранами заботливо подстраиваются под эту разновидность ввода, изменяя экран, чтобы проще было выполнить текущую задачу. Вот как выглядит экран ввода телефонного номера при доступе с iPad под управлением iOS 8.2.



Обратили внимание на отсутствие букв в области клавиатуры? Это позволяет пользователям вводить значение в нужном формате намного быстрее.

ПОДСКАЗКА

Если используемый по умолчанию синий цвет телефонных номеров в iOS Safari при вводе данных типа tel вас раздражает, можно его изменить, воспользовавшись следующим селектором: a[href^=tel] { color: inherit; }.

search

Настроить поле ввода под тип данных `search` можно следующим образом:

```
type="search"
```

Поля с этим типом ведут себя так же, как стандартные поля ввода текста. Рассмотрим следующий пример:

```
<div>
    <label for="search">Search the site...</label>
    <input id="search" name="search" type="search" placeholder= "Wyatt Earp">
</div>
```

Но программные клавиатуры, которые имеются на мобильных устройствах, зачастую предоставляют более подходящий набор клавиш. Вот как выглядит клавиатура iOS 8.2, появляющаяся при получении фокуса полем с типом вводимых данных `search`.

pattern

Поле ввода можно настроить на ожидание ввода конкретного шаблона:

```
pattern=""
```

Атрибут `pattern` позволяет с использованием регулярного выражения указать синтаксис данных, разрешенных для ввода в данном поле.

СВЕДЕНИЯ О РЕГУЛЯРНЫХ ВЫРАЖЕНИЯХ

Если вы еще не сталкивались с регулярными выражениями, я рекомендую начать их изучение со статьи, находящейся по адресу http://en.wikipedia.org/wiki/Regular_expressions.

> Регулярные выражения используются во многих языках программирования в качестве средства выявления соответствия возможным строкам. Хотя поначалу их формат отпугивает, они обладают невероятными эффективностью и гибкостью. Например, можно создать регулярное выражение для проверки соответствия формату пароля или выбора конкретного стиля схемы наименования CSS-класса. В качестве вспомогательного средства создания собственного шаблона регулярного выражения и получения визуального представления о том, как он работает, я рекомендую для начала воспользоваться запускаемым в браузере средством http://www.regexr.com/.

Рассмотрим следующий пример кода:

```
<div>
    <label for="name">Your Name (first and last)</label>
    <input id="name" name="name" pattern="([a-zA-Z]{3,30}\s*)+[a-zA- Z]
        {3,30}" placeholder="Dwight Schultz" required>
</div>
```

Работая над книгой, я приблизительно 458 секунд вел поиск в Интернете, чтобы найти регулярное выражение, которому бы соответствовал синтаксис имени и фамилии. Путем ввода регулярного выражения в атрибут `pattern` поддерживающим его браузерам предписывалось ожидать ввода в поле тех данных, у которых имеется соответствующий синтаксис. Затем в сочетании с атрибутом `required` неверно введенные данные получали в поддерживающих браузерах последующую трактовку. В данном случае я пытался отправить форму, не предоставляя фамилию.

И опять браузеры работают по-разному. Internet Explorer 11 требует, чтобы в поле вносились корректные данные, а Safari, Firefox и Chrome ничего не делают — ведут себя так, как будто имеют дело со стандартным вводом текста.

color

Хотите настроить поле ввода на прием значения цвета в шестнадцатеричном формате? Можете воспользоваться следующим кодом:

```
type="color"
```

Тип ввода `color` инициирует в поддерживающих его браузерах (на данный момент это Chrome и Firefox) появление панели выбора цвета, позволяя пользователям выбирать значения цвета в виде шестнадцатеричного числа. Рассмотрим в качестве примера следующий код:

```
<div>
    <label for="color">Your favorite color</label>
    <input id="color" name="color" type="color">
</div>
```

Ввод даты и времени

Новые типы вводимых данных `date` и `time` задумывались с целью дать пользователям единое представление о выборе значений дат и времени. Если вам когда-либо приходилось покупать на сайте билеты на какое-нибудь мероприятие, то, скорее всего,

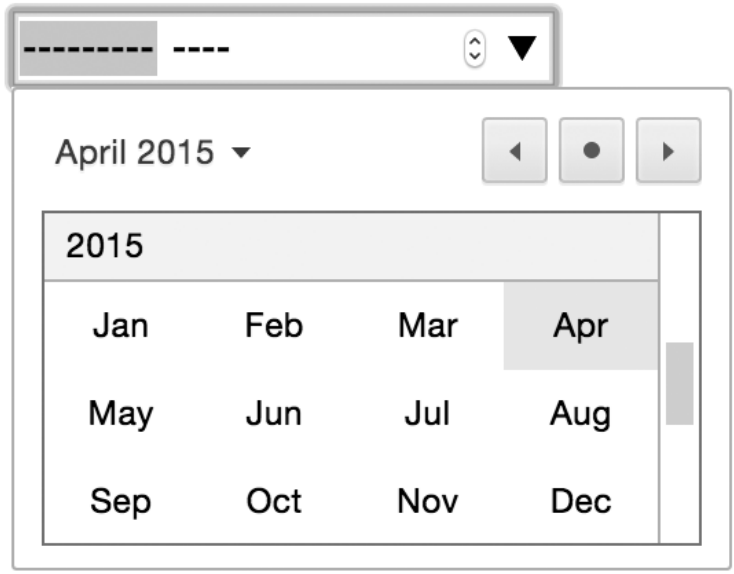
вы уже пользовались панелью выбора даты того или иного вида. Почти всегда эта функциональная возможность обеспечивалось средствами JavaScript (обычно с использованием библиотеки jQuery UI), но есть надежда, что данное необходимое всем средство станет доступным и при использовании простой разметки HTML5.

date

Рассмотрим в качестве примера следующий код:

```
<input id="date" type="date" name="date">
```

Как и в случае с вводом данных типа `color`, чисто браузерная поддержка этого типа развита пока еще слишком слабо. Изначально большинство браузеров работают с такими полями как с обычными, предназначенными для ввода текста. Эта функциональная возможность реализована только в двух современных браузерах: Chrome и Opera. И это неудивительно, поскольку оба они используют один и тот же движок (так называемый Blink, если вам интересно).



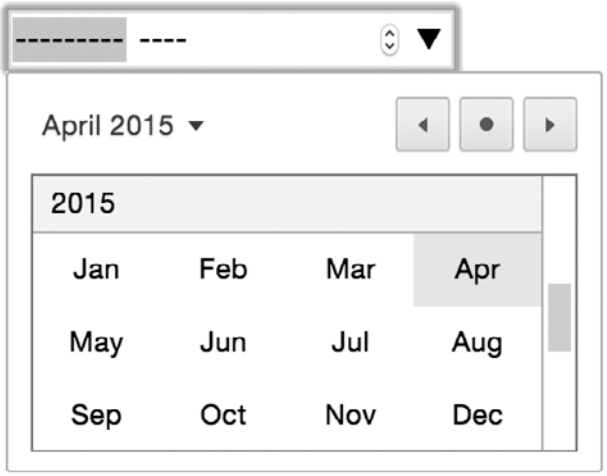
Существует множество типов вводимых данных, имеющих отношение к дате и времени. И вот их краткий обзор.

month

Рассмотрим в качестве примера следующий код:

```
<input id="month" type="month" name="month">
```

Интерфейс позволяет пользователю выбрать какой-нибудь месяц и предоставляет ввод в виде года и месяца, например `2012-06`. Как это выглядит на экране браузера, показано на следующей копии экрана.



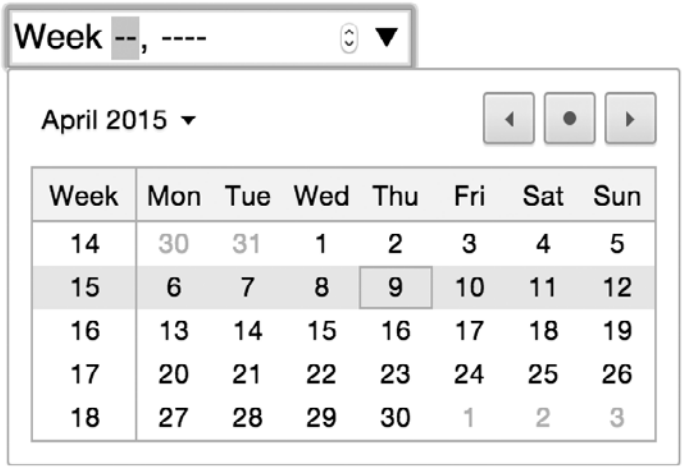
week

Рассмотрим в качестве примера следующий код:

```
<input id="week" type="week" name="week">
```

При использовании типа вводимых данных `week` панель позволяет пользователю выбрать какую-нибудь неделю года и предоставляет ввод в формате `2012-W47`.

На следующей копии экрана показано, как это выглядит в окне браузера.



time

Рассмотрим в качестве примера следующий код:

```
<input id="time" type="time" name="time">
```

Тип вводимых данных `time` позволяет вводить в поле значение в 24-часовом формате, например `23:50`.

Поле появляется в поддерживающих браузерах с элементами прокрутки, но допускает присутствие только подходящих для времени данных.

range

Указание типа вводимой информации `range` приводит к созданию элемента интерфейса под названием «ползунок». Рассмотрим пример:

```
<input type="range" min="1" max="10" value="5">
```

Здесь показано, как этот элемент выглядит в окне браузера Firefox.

По умолчанию устанавливается диапазон от 0 до 100. Но, указывая в нашем примере величины `min` и `max`, мы ограничили выбираемые числа, задав диапазон от 1 до 10.

Одна из существенных проблем, замеченных мною при указании типа вводимого значения `range`, заключается в том, что при перемещении ползунка пользователь не видит текущего значения. Хотя ползунок предназначен только для произвольного выбора чисел, я часто испытывал желание увидеть текущее значение в процессе изменения положения ползунка. Пока способов сделать это средствами HTML5 не существует. Но при крайней необходимости отображения текущего значения при перемещении ползунка этого нетрудно добиться с помощью несложного фрагмента кода JavaScript. Внесем в код предыдущего примера следующие изменения:

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1"
max="10" value="5" onchange="showValue(this.value)"><span
id="range">5</span>
```

К коду были добавлены два фрагмента: атрибут `onchange` и элемент `span`, имеющий идентификатор `range`. Теперь добавим следующий небольшой фрагмент кода JavaScript:

```
<script>
    function showValue(newValue)
    {
        document.getElementById("range").innerHTML=newValue;
    }
</script>
```

Все это позволяет получить текущее значение ползунка и показать его в элементе с идентификатором `range` (в нашем теге `span`). Затем для изменения внешнего вида значения можно воспользоваться каким угодно кодом CSS.

В HTML5 есть и другие функциональные возможности, имеющие отношение к формам. Их полную спецификацию можно найти по адресу http://www.w3.org/TR/html5/forms.html.

Как воспользоваться полифиллами для тех браузеров, которые не поддерживают новые свойства

Все разговоры о формах в HTML5, конечно, весьма интересны. Но, похоже, есть два обстоятельства, ставящие под серьезные сомнения нашу возможность ими воспользоваться: несоответствие, наблюдаемое в способах реализации свойств разными браузерами, и проблемы работы создаваемых конструкций в тех браузерах, которые вообще не поддерживают те или иные новые свойства.

Если есть необходимость поддержки некоторых из этих свойств в тех браузерах, в которых они не реализованы, обратите внимание на библиотеку Webshims Lib, которую можно загрузить по адресу http://afarkas.github.com/webshim/demos/. Эта библиотека полифиллов, написанная Александром Фаркашем (Alexander Farkas), может загружать полифиллы форм, заставляя тем самым браузеры, не поддерживающие соответствующие основные свойства форм HTML5, успешно справляться с решаемыми с их помощью задачами.

ПРИМЕНЯЯ ПОЛИФИЛЛЫ, ПРОЯВЛЯЙТЕ ОСМОТРИТЕЛЬНОСТЬ

При каждом применении полифиллного сценария не забудьте правильно оценить обстановку. Несмотря на свою несомненную пользу, эти сценарии утяжеляют ваш проект. К примеру, для работы Webshims также требуется применение библиотеки jQuery, а там, если вы раньше не работали с jQuery, следует иметь в виду наличие дополнительных зависимостей. Так что связываться с полифиллами без острой необходимости я бы не советовал.

Библиотека Webshims хороша тем, что она добавляет только нужные полифиллы. Если форма просматривается в браузере, имеющем встроенную поддержку соответствующих свойств HTML5, то добавляется совсем мало кода. Более старые браузеры, хотя для них и требуется загрузка большего объема кода (поскольку они изначально обладают меньшей функциональностью), получают возможность обеспечить пользователям сходное восприятие, но уже с соответствующими функциональными возможностями, реализованными с помощью JavaScript.

Но выгоду от применения данной технологии получают не только устаревшие браузеры. Как мы уже видели, полная реализация свойств HTML5 для работы с формами имеется далеко не у всех современных браузеров. Применение на странице библиотеки Webshims закрывает и эти бреши. К примеру, Safari не выдает никаких предупреждающих сообщений, когда форма HTML5 отправляется с пустыми обязательными полями. Пользователь не получает извещения о характере возникшей проблемы, что вряд ли можно назвать идеальным решением. А при добавлении на страницу библиотеки Webshims уведомление делается по ранее рассмотренному сценарию.

А в Firefox отсутствует возможность прокрутки чисел при указании атрибута `type="number"`, но библиотека Webshims силами jQuery обеспечивает весьма удобный альтернативный вариант. Короче говоря, Webshims является весьма эффективным средством, поэтому, если установить этот небольшой, но довольно приятный пакет

и привязать к нему свою конструкцию, можно будет заняться написанием форм со всеми нововведениями HTML5, спокойно осознавая тот факт, что при пользовании формой все увидят то, что им нужно (за исключением тех двоих, которые используют IE6 с выключенным JavaScript, но вы знаете, кем они на самом деле являются, и их не следует брать в расчет!).

Загрузите сначала библиотеку Webshims (http://github.com/aFarkas/webshim/downloads) и извлеките из нее нужный пакет функций. Теперь скопируйте содержимое папки js-webshim в соответствующий раздел своей веб-страницы. Чтобы ничего не усложнять, для данного примера я скопировал его в корневой каталог сайта.

Теперь добавьте в раздел своей страницы следующий код:

```
<script src="js/jquery-2.1.3.min.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
<script>
    // запрос нужных свойств:
    webshim.polyfill('forms');
</script>
```

Изучим этот код по разделам. Сначала в нем делается ссылка на локальную копию библиотеки jQuery (последнюю версию которой можно получить по адресу www.jquery.com) и на сценарий Webshim:

```
<script src="js/jquery-2.1.3.min.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
```

А потом сценарию указывается на необходимость загрузки всех нужных полифиллов:

```
<script>
    // запрос нужных свойств:
    webshim.polyfill('forms');
</script>
```

Вот и все, что нужно было сделать. Теперь отсутствующие функциональные возможности автоматически добавятся за счет загрузки соответствующего полифилла. Превосходно!

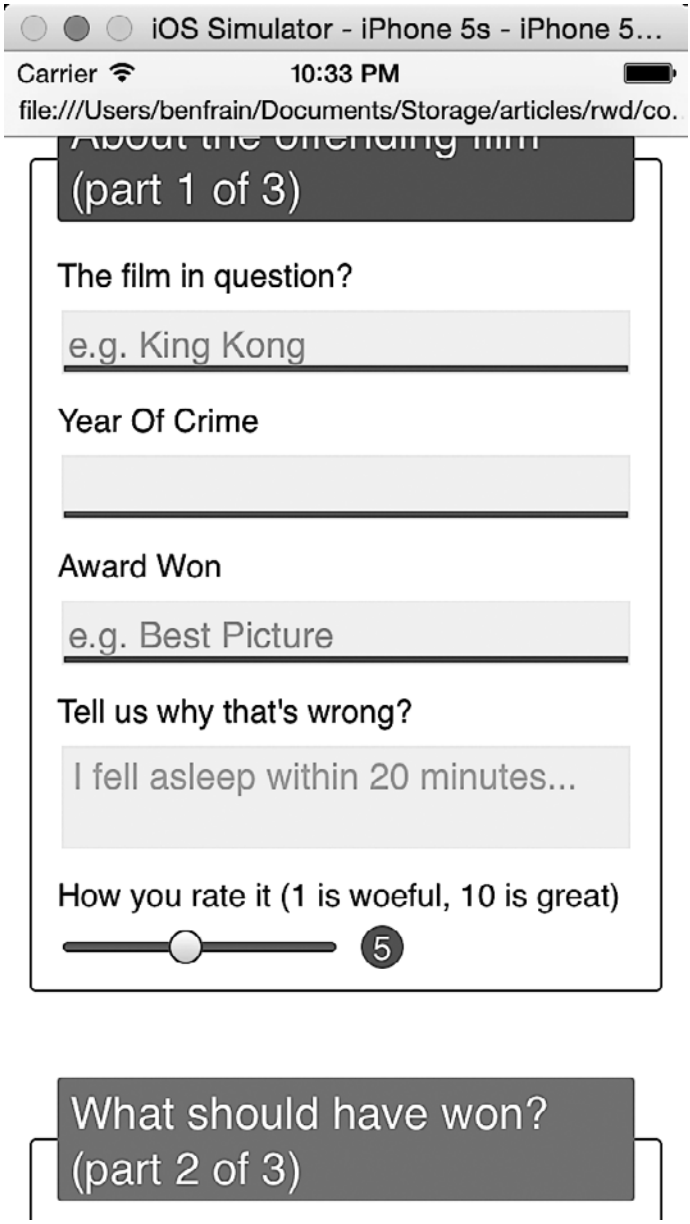
Придание формам HTML5 стилевого оформления с помощью CSS3

Теперь, после того как наши формы приобрели полный набор функциональных возможностей для работы в любых браузерах, появилась потребность в повышении их привлекательности при работе с окнами просмотра различных размеров. Я, конечно, не считаю себя великим дизайнером, но все же думаю, что с применением ряда технологий, изученных в предыдущих главах, нам удастся повысить эстетическую привлекательность нашей формы.

Код формы со стилевым оформлением можно увидеть в файле каталога example_09-02, и не забудьте: если у вас еще нет кода примеров, его можно получить по адресу http://rwd.education.

В данный пример я включил также две версии таблицы стилей: styles.css — версию, включающую префиксы производителей (добавленные с помощью Autoprefixer), и styles-unprefixed.css — версию с чистым кодом CSS. При желании разобраться в том, что к чему применяется, легче работать с последней версией.

Вот как форма выглядит в небольшом по размеру окне просмотра с применением ряда основных стилевых настроек.



А вот как она выглядит в более крупном окне просмотра.

Если посмотреть на код CSS, то в нем можно заметить множество технологических приемов, рассмотренных в предыдущих главах. Например, Flexbox (глава 3) был использован для создания гибких элементов с равномерным шагом, а преобразования и переходы (глава 8) — для увеличения размеров полей ввода, получающих фокус, и вертикального переворота кнопки вопроса о готовности и отправки формы (ready/submit) при получении ею фокуса. Блочные тени и градиенты (глава 6) использовались для выделения различных областей формы. Медиазапросы (глава 2) применялись для переключения Flexbox-направления для различных

размеров окон просмотра, а селекторы CSS Level 3 (глава 5) — для выбора по признакам несоответствия селекторам.

Мы не станем здесь еще раз вдаваться в подробности данных технологий. Лучше сконцентрируемся на двух моментах: во-первых, как визуально обозначить поля, требующие обязательного заполнения (и в качестве дополнения к этому как обозначить наличие в них введенного значения), и во-вторых, как создать эффект заливки, когда поле получает пользовательский фокус.

Обозначение полей, требующих обязательного заполнения

Обязательные для заполнения поля можно обозначить для пользователя с помощью только кода CSS, например:

```
input:required {
    /* стили */
}
```

Используя этот селектор, к обязательным для заполнения полям можно добавить границу, или контур, или фоновое изображение. Фантазировать здесь можно бесконечно! Можно также использовать конкретный селектор для нацеливания на поле ввода, обязательное для заполнения, только при получении им фокуса, например:

```
input:focus:required {
    /* стили */
}
```

Но таким образом стили будут применяться к самому полю ввода. А что, если понадобится изменить стили в отношении связанного с ним элемента `label`? Я решил, что буду помечать поля, обязательные для заполнения, небольшой звездочкой сбоку от надписи. Но тут возникает проблема. Дело в том, что, если элемент получает некое состояние (под которым я подразумеваю `hover`, `focus`, `active`, `checked` и т. д.), CSS позволяет применять изменения к его дочерним элементам, к самому этому элементу или к непосредственно примыкающему к нему и последующим одноуровневым родственным элементам. В следующих примерах используется состояние `:hover`, но такой вариант, несомненно, вызовет проблемы на устройствах с сенсорным экраном:

```
.item:hover .item-child {}
```

При использовании предыдущего селектора стили применяются к `item-child` при прохождении над ним указателя мыши:

```
.item:hover ~ .item-general-sibling {}
```

А при использовании показанного ранее селектора при прохождении указателя стили применяются к элементу `item-general-sibling`, если он находится на том же DOM-уровне, что и сам элемент, и следует за ним:

```
.item:hover + .item-adjacent-sibling {}
```

Благодаря предыдущему коду, когда указатель будет находиться над элементом, стили будут применяться к элементу `item-adjacent-sibling`, если по отношению

к исходному элементу тот является примыкающим одноуровневым родственным элементом, следующим непосредственно за ним в DOM-модели.

Итак, вернемся к нашему вопросу. Если имеется форма с надписями и полями, где надпись расположена выше поля ввода, чтобы дать нам требуемую основу разметки, то мы попадаем в тупик:

```
<div class="form-Input_Wrapper">
    <label for="film">The film in question?</label>
    <input id="film" name="film" type="text" placeholder="e.g. King
        Kong" required/>
</div>
```

В этой ситуации использование одного лишь кода не позволяет изменить стиль надписи, расположенной над полем ввода, каким бы оно ни было — обязательным для заполнения или нет (поскольку оно следует в разметке после надписи). Мы можем изменить порядок следования этих двух элементов в разметке, но тогда придется смириться с надписью, расположенной под полем ввода.

И все же Flexbox предоставляет нам возможность без особого труда наглядно поменять элементы местами (если вы еще не прочитали, как это делается, обратитесь к материалам главы 3). При этом можно воспользоваться следующей разметкой:

```
<div class="form-Input_Wrapper">
    <input id="film" name="film" type="text" placeholder="e.g. King
        Kong" required/>
    <label for="film">The film in question?</label>
</div>
```

а затем просто применить к родительскому элементу объявление `flex-direction: row-reverse` или `flex-direction: column-reverse`. Такие объявления меняют визуальный порядок следования дочерних элементов на обратный, позволяя получить эстетически более привлекательное размещение надписи над полем (при меньших по размеру окнах просмотра) или слева от поля ввода (при более крупных окнах просмотра). Теперь мы можем заняться фактическим обеспечением обозначений полям, подлежащим обязательному заполнению, и их выделением при получении фокуса.

Благодаря пересмотренной разметке теперь это можно сделать с применением селектора непосредственно примыкающего одноуровневого элемента:

```
input:required + label:after { }
```

Этот селектор в конечном итоге предписывает применять правило к каждой надписи, которая следует за полем ввода, имеющим атрибут `required`. А вот как выглядит код CSS для данного раздела:

```
input:required + label:after {
    content: "*";
    font-size: 2.1em;
    position: relative;
    top: 6px;
    display: inline-flex;
```

```
    margin-left: .2ch;
    transition: color, 1s;
}

input:required:invalid + label:after {
    color: red;
}

input:required:valid + label:after {
    color: green;
}
```

Затем, если фокус устанавливается на поле ввода, обязательное для заполнения, и в него вводится соответствующее значение, звездочка меняет цвет на зеленый. Мелочь, а приятно.

ПРИМЕЧАНИЕ

Наряду с уже рассмотренными существуют и другие селекторы, как уже реализованные, так и задекларированные. Для получения самого актуального перечня этих селекторов изучите последнюю редакторскую правку Selectors Level 4 Specification, которая находится по адресу http://dev.w3.org/csswg/selectors-4/.

Создание эффекта заливки фона

В главе 6 мы научились выполнять линейные и радиальные градиенты для использования в качестве фоновых изображений, но, к сожалению, переход между двумя фоновыми изображениями невозможен (что вполне резонно, поскольку браузер, по сути, растрирует объявление в изображение). И тем не менее мы можем обеспечивать переходы между значениями сопутствующих свойств, таких как `background-position` и `background-size`. Мы воспользуемся этим обстоятельством для создания эффекта заливки в момент получения фокуса полем ввода или областью ввода текста.

К полю ввода добавляются следующие свойства и значения:

```
input:not([type="range"]),
textarea {
    min-height: 30px;
    padding: 2px;
    font-size: 17px;
    border: 1px solid #ebebeb;
    outline: none;
    transition: transform .4s, box-shadow .4s, background-position .2s;
    background: radial-gradient(400px circle, #fff 99%, transparent 99%), #f1f1f1;
    background-position: -400px 90px, 00;
    background-repeat: no-repeat, no-repeat;
    border-radius: 0;
    position: relative;
}
```

```
input:not([type="range"]):focus,
textarea:focus {
    background-position: 00, 00;
}
```

В первом правиле генерируется сплошной белый радиальный градиент, имеющий позицию за пределами видимости. Расположенный за ним фоновый цвет (шестнадцатеричное значение, указанное после `radial-gradient`) не имеет смещения, поэтому предоставляет исходный цвет. Когда поле ввода получает фокус, фоновая позиция `radial-gradient` устанавливается обратно на исходные значения, и поскольку происходит возвращение к установке `background-image`, создается красивый переход между двумя настройками фона. Результатом станет заливка поля другим цветом при получении фокуса.

ПРИМЕЧАНИЕ

При придании стилевого оформления частям исходного пользовательского интерфейса у разных браузеров находятся собственные селекторы и возможности. Для удобства работы с множеством конкретных селекторов Аврелий Уэнделкен (Aurelius Wendelken) собрал их во внушительный перечень. Я сделал для себя копию этого перечня (или ответвление в терминологии управления Git-версиями), которую можно найти по адресу https://gist.github.com/benfrain/403d3d3a8e2b6198e395.

Резюме

В данной главе были изучены способы применения новых HTML5-атрибутов для работы с формами. Они позволяют нам сделать формы более удобными, а собираемые с их помощью данные — лучше соответствующими конкретным требованиям. Кроме того, при необходимости мы можем поддерживать эту новую разметку в актуальном состоянии с помощью сценариев-полифиллов, написанных на JavaScript, чтобы давать пользователям единое представление о свойствах форм независимо от возможностей применяемых пользователями браузеров.

Мы приблизились к финишу путешествия по адаптивным конструкциям на основе HTML5 и CSS3. Хотя за то время, которое мы провели вместе, был охвачен большой объем материала, я прекрасно понимаю, что поделиться информацией на все случаи жизни просто невозможно. Поэтому в последней главе хотел бы взглянуть на подходы к адаптивному веб-дизайну на более высоком уровне и постараться рассказать о накопленном опыте, чтобы направить в нужное русло ваш первый или последующий проект разработки адаптивной конструкции.

10 Подходы к адаптивному веб-дизайну

В моих любимых романах и фильмах всегда присутствует сцена с наставником, дающим герою ценный совет и некие магические предметы. Известно, что все это пригодится, но неизвестно, когда или каким образом.

Итак, в заключительной главе я хочу сыграть роль наставника (к тому же с моей поредевшей шевелюрой на роль героя я вряд ли подойду). При этом хочу, чтобы вы, мой прилежный ученик, потратили еще немного времени на усвоение заключительных советов, прежде чем направите все свои усилия на поиск собственных решений в области адаптивного веб-дизайна.

Эта глава наполовину будет состоять из философских размышлений и наставлений, а наполовину — из подборки не связанных между собой советов и технических приемов. Надеюсь, в определенные моменты поиска решений в области разработки адаптивных конструкций эти советы смогут вам пригодиться. А вот и вопросы, которые мы будем рассматривать:

- обкатка дизайна в браузере и на реальных устройствах на самых ранних стадиях;
- задание контрольных точек применительно к дизайну;
- использование принципа постепенного усложнения;
- определение матрицы браузерной поддержки;
- применение на практике принципа постепенного усложнения;
- привязка контрольных точек CSS к JavaScript;
- отказ от использования сред разработки CSS при создании конечного продукта;
- выработка наиболее практичных решений;
- написание как можно более простого кода;
- скрытие, показ и загрузка содержимого для всевозможных окон просмотра;
- возложение самых трудных задач визуального оформления на CSS;
- использование средств контроля качества кода;
- анализ и тестирование производительности веб-страниц (webpagetest.org);
- применение более скоростных и эффективных технологий;
- отслеживание появления очередных грандиозных нововведений.

Обкатка дизайна в браузере на самых ранних стадиях

Чем больше адаптивных проектов я разрабатывал, тем более важной мне представлялась обкатка дизайна в среде браузера на самых ранних стадиях. Если вы и дизайнер и разработчик в одном лице, то дело упрощается. Как только сложится более или менее конкретное визуальное представление о насущных потребностях, следует обкатать прототип в браузере и далее прорабатывать идею в среде браузера. Этот подход можно расширить за счет прогона сразу всех высококачественных полностраничных прототипов. Или же присмотреться к использованию таких средств, как Style Tiles, занимающих место между подборкой изображений и полноценным макетом. Во введении в Style Tiles (http://styletil.es/) это средство описывается следующим образом:

> «Style Tiles является дизайнерской подборкой, состоящей из шрифтов, цветовых решений и элементов интерфейса с целью передать суть визуального фирменного стиля для веб-технологий».

Я понял, что графический результат такого рода может пригодиться для того, чтобы заинтересованные стороны получили представление о внешнем виде и обменялись мнениями, не проводя при этом бесконечные раунды сложных переговоров.

Задание контрольных точек применительно к дизайну. Я хотел бы повторить мысль, высказанную в предыдущих главах: пусть дизайн определяет, где нужно расставить контрольные точки. Когда дизайн оценивается в браузере, это процесс существенно облегчается. Всегда следует вносить изменения в дизайн, исходя из начального варианта для самых небольших экранов, и идти по нарастающей до дизайна для самых крупных экранов. Поэтому по мере увеличения размеров окон просмотра нужно смотреть, до каких пределов дизайн способен работать, прежде чем потребуется вводить контрольную точку.

Вы также поймете, что данный способ облегчит программирование дизайна. Сначала нужно создать код CSS для самых скромных по размеру окон просмотра, а затем последовательно добавлять в медиазапросах изменения к различным элементам, например:

```
.rule {
    /* Стили для самых скромных окон просмотра */
}

@media (min-width: 40em) {
    .rule {
        /* Изменения для окон просмотра среднего размера */
    }
}

@media (min-width: 70em) {
    .rule {
        /* Изменения для более крупных окон просмотра */
    }
}
```

Просмотр и обкатка дизайна на реальных устройствах

Приступать к созданию «лаборатории устройств» для тестирования вашей работы можно с самых старых моделей телефонов и планшетов. Наличие ряда разнообразных устройств может принести большую пользу, позволив не только отследить работу дизайна на различных устройствах, но и выявить особенности разметки и вывода элементов на экран. Ведь известие о не работающем в конкретной среде проекте, считающемся уже завершенным, вряд ли кого-то обрадует. Приступайте к тестированию как можно раньше и проводите его как можно чаще! Все окупится сторицей. Можно, к примеру, купить устаревшие модели телефонов и планшетных компьютеров на eBay или приобрести их у друзей и знакомых, обновляющих свою технику.

ИСПОЛЬЗУЙТЕ ДЛЯ СИНХРОНИЗАЦИИ РАБОТЫ ТАКИЕ СРЕДСТВА, КАК BROWSERSYNC

Одним из наиболее эффективных средств экономии времени, которым я пользуюсь в последнее время, является BrowserSync. После его настройки при сохранении работы любые изменения, вносимые, к примеру, в CSS, внедряются в браузер, не требуя при этом постоянного обновления экрана. Если это вас не сильно впечатлило, добавлю: обновляются и любые другие окна браузеров на устройствах, находящихся в той же Wi-Fi-сети. При этом при внесении каждого изменения уже не приходится брать каждое тестируемое устройство и щелкать на кнопке обновления. Данное средство синхронизирует также прокрутку и щелчки на элементах интерфейса. Настоятельно рекомендую: http://browsersync.io/.

Использование принципа постепенного усложнения

В предыдущих главах был вкратце упомянут принцип постепенного усложнения. Полагаю, об этом подходе к разработке, в котором я вижу немалую практическую пользу, нелишне будет напомнить еще раз. Основной замысел постепенного усложнения заключается в том, что весь ваш интерфейсный код (HTML, CSS, JavaScript) начинается с замысла наименьшего общего знаменателя. Затем код постепенно усложняется с расчетом на устройства и браузеры с более широкими возможностями. Если вы привыкли работать в обратном порядке, то все может показаться неким упрощением, и так оно и есть: если вы имеете оптимальные наработки, а затем нашли способ добиться работоспособности в устройствах и браузерах с самыми скромными возможностями, то поймете, что постепенное усложнение является наиболее простым подходом.

Представьте себе маломощное устройство с весьма ограниченными возможностями: отсутствует JavaScript, не поддерживается Flexbox, нет поддержки CSS3 и CSS4. Что в подобных обстоятельствах можно сделать, чтобы обеспечить пользователю удобство восприятия содержимого?

Важнее всего создать на HTML5 выразительную разметку с точным описанием всего этого содержимого. Если создаются чисто текстовые или иные сайты, осно-

ванные на содержимом, то эта задача считается наиболее простой. В таком случае следует сконцентрироваться на правильном использовании элементов `main`, `header`, `footer`, `article`, `section` и `aside`. Это не только поможет отделить друг от друга различные разделы кода, но и обеспечит вашим пользователям повышенную доступность информации без каких-либо дополнительных затрат с вашей стороны.

Если создается что-нибудь вроде приложения, основанного на веб-технологиях, или визуальные компоненты пользовательского интерфейса — ползунки, вкладки, раскрывающиеся панели и т. п., нужно будет подумать о том, как превратить визуальную схему в доступную разметку. Дело в том, что хорошо продуманная разметка очень важна, поскольку она обеспечивает базовый уровень восприятия для всех пользователей. Чем больше полезных функциональных свойств можно будет предоставить средствами HTML, тем меньше придется иметь дело с CSS и JavaScript для поддержки устаревших браузеров. Ведь никто — я уверен, что никто, — не любит создавать код для поддержки старых браузеров.

ПРИМЕЧАНИЕ

Для дальнейшего изучения данной темы и получения полезных практических примеров я рекомендую следующие статьи: http://www.cssmojo.com/how-to-style-a-carousel/ и http://www.cssmojo.com/use-radio-buttons-for-single-option/. Они могут дать глубокое понимание того, как довольно сложные взаимодействия могут быть обработаны конструкциями, созданными на основе HTML и CSS.

И ни в коем случае не нужно считать переход к подобному видению предмета каким-то подвигом. Скорее всего, это подход, который сослужит вам неплохую службу в стремлении выполнять как можно меньший объем работы для поддержки слабых браузеров.

А теперь поговорим о браузерах.

Определение матрицы браузерной поддержки

Для разработки удачного адаптивного веб-дизайна весьма важную роль может сыграть предварительное изучение браузеров и устройств, поддержку которых необходимо обеспечить вашему проекту. Мы уже выясняли, почему в данном отношении полезно будет применять принцип постепенного усложнения. При правильном подходе к делу это позволит основному объему вашего сайта сохранять функциональность даже в самых старых браузерах.

Но обстоятельства могут сложиться так, что вам потребуется стартовать с более высокого набора предварительных требований. Возможно, это будет работа над проектом, в котором большое значение имеет JavaScript, как нередко и случается. В таком случае вы все равно можете придерживаться принципа постепенного усложнения. Просто это усложнение будет происходить с других начальных позиций.

Самое главное здесь — установить, какой будет именно ваша начальная позиция. Тогда и только тогда можно будет определить и согласовать объем визуальных

и функциональных наработок для различных намеченных для поддержки браузеров и устройств.

Функциональное, но не эстетическое единообразие

Нереально и неприемлемо добиваться того, чтобы сайт выглядел и работал одинаково в любом браузере. Помимо порой весьма странных особенностей, присущих конкретным браузерам, существуют также важные функциональные аспекты. Например, мы должны брать в расчет нацеливание на прикосновение к кнопкам или ссылкам на сенсорных экранах, не относящихся к устройствам, основанным на применении мыши.

Поэтому частью вашей роли, как разработчика адаптивных веб-приложений, является доведение до сознания тех, кому вы подотчетны (начальников, клиентов, акционеров), что поддержка устаревших браузеров не означает сохранение неизменного внешнего вида в устаревших браузерах. Я придерживаюсь того мнения, что все браузеры в матрице поддержки получат функциональное, но не эстетическое единообразие. Это означает, что если вам нужно создать систему оформления заказа, то все пользователи должны иметь возможность пройти через нее и приобрести товары. В более современных браузерах все может быть представлено пользователям в более привлекательном виде с точки зрения визуального преставления и удобства взаимодействия с сайтом, но возможность добиться выполнения основной задачи должна быть у всех.

Выбор поддерживаемых браузеров

Обычно, когда речь заходит о браузерах, которые следует поддерживать, имеется в виду глубина экскурса в прошлое, в которую нужно заглядывать. В зависимости от ситуации есть две возможности, которые следует рассмотреть.

Если дело касается существующего сайта, посмотрите на статистику его посещений в Google Analytics или подобных системах. Получив представление, можно будет сделать ряд прикидок. Например, если стоимость поддержки браузера X ниже расчетных затрат на его поддержку, то нужно поддерживать браузер X!

Кроме того, если, по статистике, теми или иными браузерами пользуются менее 10 % пользователей, следует просмотреть статистику за предыдущие периоды и выявить тенденцию. Как объем использования менялся за последние 3, 6 и 12 месяцев? Если он составляет 6 % и его значение за последние 12 месяцев уменьшилось в два раза, то это более чем убедительный довод для принятия решения по исключению этого браузера из числа тех, к которым должны применяться конкретные усовершенствования.

Если дело касается нового проекта, статистика для которого еще не наработана, то я обычно придерживаюсь политики «двух предыдущих». Это текущая версия плюс две предыдущие версии каждого браузера. Например, если текущей является версия Internet Explorer 12, рассматривайте применение усовершенствований для этой версии, а также для IE10 и IE11 — двух ее предшественников. Проще, конечно, выбирать из самых популярных браузеров, подвергаемых постоянным обновлениям и имеющим короткий цикл выпуска следующей версии (к примеру, Firefox и Chrome).

Создание нескольких уровней пользовательского восприятия

А теперь представим себе, что акционеры уже в курсе дела. Представим также, что у вас есть определенная подборка браузеров, на которых вы хотите обеспечить более высокий уровень пользовательского восприятия. Теперь нужно настроиться на создание нескольких таких уровней. Хотелось бы ничего не усложнять, поэтому везде, где возможно, я провожу оптимизацию для определения простого базового уровня и более совершенного уровня.

Базовый уровень пользовательского восприятия обеспечит минимально жизнеспособная версия сайта, а улучшенный уровень — наиболее полнофункциональная и эстетически привлекательная версия. Вам может понадобиться более точно подогнать характеристики уровней, разветвляя, к примеру, уровень восприятия в зависимости от функциональных возможностей браузеров, используя признаки поддержки Flexbox или трехмерных преобразований. Независимо от способа определения уровней нужно определять как сами уровни, так и то, что вы собираетесь дать пользователю на каждом из них. Затем можно будет приступить к созданию кода для уровней.

Практическое обеспечение уровней пользовательского восприятия. В настоящее время самую действенную помощь в улучшении и разветвлении уровней пользовательского восприятия на основе возможностей устройств оказывает средство Modernizr. Хотя здесь уже не обойтись без добавления в ваш проект JavaScript, я считаю, что это вполне оправданно.

При написании кода CSS следует помнить, что базовый уровень восприятия должен обеспечиваться кодом без медиазапросов и селекторов, требующих наличия классов, добавляемых средством Modernizr. Затем с помощью Modernizr можно расположить по уровням нарастающее количество возможностей улучшения пользовательского восприятия, основываясь на возможностях браузеров. Если вернуться к уже рассмотренному коду, находящемуся в файле каталога example_08-07, можно уяснить данный замысел и изучить шаблон кода, примененный к схеме меню, выходящему за пределы холста.

Привязка контрольных точек CSS к JavaScript

Обычно, когда что-либо основанное на веб-технологиях предполагает какую-то работу в интерактивном режиме, к делу подключается JavaScript. При разработке проекта адаптивного приложения вам, скорее всего, при разных размерах окон просмотра потребуются различные действия. Не только в CSS, но и в JavaScript.

Предположим, что нам нужно вызвать конкретную функцию JavaScript при достижении конкретной контрольной точки в CSS (хочу напомнить, что понятие контрольной точки используется для определения точки, в которой адаптивный дизайн должен претерпеть существенные изменения). Предположим, что контрольной точкой считается 47,5 rem (при основном размере шрифта 16 пикселов это будет эквивалентно 760 пикселам) и нам нужно при этом размере всего лишь

запустить функцию. Очевидным решением будут простой замер ширины экрана и вызов функции, если значение совпадает с тем значением, которое решено было принять за вашу контрольную точку CSS.

JavaScript всегда будет возвращать значение ширины в пикселах, а не в rem, что является первым осложнением. Но даже если контрольные точки в CSS заданы в пикселах, это все равно будет означать, что у нас имеются два места, в которые нужно вносить обновления и изменения этих значений при изменении размеров окна просмотра.

К счастью, есть более подходящий способ. Впервые я ознакомился с этой технологией на сайте Джереми Кейта (Jeremy Keith) http://adactio.com/journal/5429/.

Полный код можно найти в файле каталога example_10-01. Но основной замысел состоит в том, что в CSS вставляется нечто такое, что может быть легко считано и правильно воспринято кодом JavaScript.

Рассмотрим этот прием в CSS:

```
@media (min-width: 20rem) {
    body::after {
        content: "Splus";
        font-size: 0;
    }
}
@media (min-width: 47.5rem) {
    body::after {
        content: "Mplus";
        font-size: 0;
    }
}
@media (min-width: 62.5rem) {
    body::after {
        content: "Lplus";
        font-size: 0;
    }
}
```

В каждой контрольной точке, которую мы хотим связать с JavaScript, используется псевдоэлемент `after` (можно использовать и `before`, так как подойдет любой из них) и указывается содержимое этого псевдоэлемента, чтобы дать имя нашей контрольной точке. В предыдущем примере я использовал `Splus` для экранов не меньше меньших, `Mplus` — для экранов не меньше средних и `Lplus` — для экранов не меньше больших. Здесь можно использовать любые значимые для вас имена и изменять значения там, где в этом усматривается смысл (использовать другие показатели ориентации, высоты, ширины и т. д.).

СОВЕТ

Псевдоэлементы ::before и ::after вставляются в DOM в качестве теневых DOM-элементов. Псевдоэлемент ::before вставляется в качестве первого дочернего элемента своего родительского элемента, а ::after — в качестве его последнего дочернего элемента. В этом можно убедиться, заглянув в инструментарий разработчика вашего браузера.

При использовании этих настроек CSS мы можем просмотреть DOM-дерево и увидеть псевдоэлемент `::after`.

Затем в нашем коде JavaScript это значение можно будет прочитать. Сначала мы присвоим значение переменной:

```
var size = window.getComputedStyle(document.body,':after').
getPropertyValue('content');
```

А затем, располагая всем этим, можем что-либо сделать. Чтобы подтвердить эту концепцию, я создал простую самостоятельно вызываемую функцию (самостоятельность просто означает, что она выполняется сразу же, как только браузер проведет ее синтаксический анализ), которая после загрузки страницы выводит в окне предупреждения разные сообщения в зависимости от размера окна просмотра:

```
;(function alertSize() {
    if (size.indexOf("Splus") !=-1) {
        alert('I will run functions for small screens');
    }
    if (size.indexOf("Mplus") !=-1) {
        alert('At medium sizes, a different function could run');
    }
    if (size.indexOf("Lplus") !=-1) {
        alert('Large screen here, different functions if needed');
    }
})();
```

Надеюсь, что в реальных проектах ваша фантазия пойдет дальше вывода предупреждающего сообщения, и думаю, что из такого подхода к решению проблемы вы извлечете немалую пользу. И у вас никогда не возникнут опасения относительно рассогласованности медиазапросов CSS и зависимых от ширины окна просмотра функций JavaScript.

Отказ от использования сред разработки CSS при создании конечного продукта

Существует множество свободно распространяемых сред, помогающих быстро создать прототипы и построить адаптивные сайты. В качестве двух самых распространенных из них можно назвать Bootstrap (http://getbootstrap.com/) и Foundation

(http://foundation.zurb.com/). Несмотря на то что это весьма интересные проекты, в особенности для обучения созданию адаптивных визуальных шаблонов, я полагаю, что в конечном продукте их использования нужно избегать.

Я обсуждал эту тему со многими разработчиками, начинавшими проекты с использованием одной из таких сред, а затем вносившими в код изменения, отвечающие насущным потребностям. Этот подход может принести большую выгоду в плане быстрого создания прототипов (например, для демонстрации клиентам ряда приемов взаимодействия с продуктом), но я полагаю, что в проектах, которые предназначены для производственной деятельности, их использование неприемлемо.

Во-первых, с технической точки зрения вполне возможно, что, положившись с самого начала на одну из таких сред, вы получите в результате проект, объем кода которого будет значительно превышать реальные потребности самого проекта. Во-вторых, с точки зрения эстетики из-за высокой популярности таких сред вполне возможно, что ваш проект в итоге будет очень похож на бесчисленное множество других подобных проектов.

И наконец, если просто переносить код в свой проект и подстраивать его под свои нужды, вы вряд ли будете в полной мере понимать, что делается «под капотом». Осмыслить код, помещаемый в свой проект, можно, только очертив круг проблем и добившись их решения.

Выработка наиболее практичных решений

Когда дело доходит до разработки веб-интерфейса, меня особенно настораживает оторванный от жизни идеализм. Хотя нужно неизменно стремиться делать все как полагается, побеждать все-таки должен именно практицизм. Хочу привести пример, окончательный код которого находится в файле каталога example_10-02. Предположим, что имеется кнопка, требующая стилевого оформления и открывающая меню, которое выходит за пределы общего холста. Естественным желанием может стать использование примерно такой разметки:

```
<button class="menu-toggle js-activate-off-canvas-menu">
    <span aria-label="site navigation">&#9776;</span> menu
</button>
```

Простенько и со вкусом. Мы имеем дело с кнопкой, поэтому и воспользовались элементом `button`. В отношении данной кнопки мы применили два HTML-класса, один из которых станет привязкой для придания ей стиля с помощью CSS (`menu-toggle`), а второй — привязкой к коду JavaScript (`js-activate-off-canvas-menu`). Кроме того, мы воспользовались атрибутом `aria-label` (более подробно стандарт ARIA рассмотрен в главе 4), чтобы довести до модуля чтения с экрана значение символа, находящегося внутри `span`-контейнера. В данном примере используется HTML-код `☰`, представляющий собой символ Юникода под названием «триграмма для небес». Он применяется здесь по той простой причине, что похож на значок гамбургера, часто используемый как символ меню.

СОВЕТ

Если вам нужен полезный совет по поводу места и способа использования атрибута aria-label, я настоятельно рекомендую следующую статью на сайте разработчиков Opera, написанную Хейдоном Пикерингом (Heydon Pickering): https://dev.opera.com/articles/ux-accessibility-aria-label/.

Похоже, пока дела идут как нельзя лучше. Осмысленность, весьма доступная разметка и классы для разделения сфер интересов. Великолепно. Добавим стилевое оформление:

```
.menu-toggle {
    appearance: none;
    display: inline-flex;
    padding: 0 10px;
    font-size: 17px;
    align-items: center;
    justify-content: center;
    border-radius: 8px;
    border: 1px solid #ebebeb;
    min-height: 44px;
    text-decoration: none;
    color: #777;
}
[aria-label="site navigation"] {
    margin-right: 1ch;
    font-size: 24px;
}
```

Откроем это в Firefox и увидим там следующее.

Получилось не совсем то, на что мы надеялись. В данном случае браузер решил, что мы зашли слишком далеко: Firefox просто не разрешил нам использовать элемент `button` в качестве Flex-контейнера. Для разработчика это вполне реальная конфликтная ситуация. И что мы сделали неправильно: выбрали не тот элемент или не то эстетическое оформление? В идеале нам хотелось получить меню в виде значка гамбургера слева и слова menu справа.

СОВЕТ

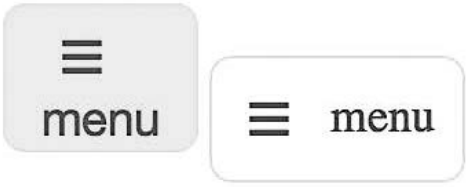
В предыдущем коде можно заметить применение свойства appearance. Оно используется для удаления исходного стилевого оформления, задаваемого браузерами для элементов формы, и имеет весьма давнюю историю. Одно время оно входило в спецификацию W3C, затем было оттуда убрано, оставаясь в версиях с использованием префиксов производителей в браузерах на основе Mozilla и WebKit. К нашему общему удовольствию, теперь оно вернулось в стандарты: http://dev.w3.org/csswg/css-ui-4/#appearance-switching.

Когда ссылка превращается в кнопку. Не стану лгать. В подобной парадоксальной ситуации я выбираю из двух последнее. Затем пытаюсь компенсировать выбор неверного элемента за счет выбора следующего наиболее подходящего элемента и задания ARIA-роли там, где это возможно. В данном случае, хотя наша кнопка меню ссылкой, конечно же, не является (ведь она же не уводит пользователя в какое-нибудь другое место), она все же представляет собой тег `a`, которым я и воспользуюсь. Я решил, что следующий наиболее подходящий выбор будет больше похож на кнопку, чем на любой другой элемент. А использование ссылки даст возможность добиться желаемого эстетического результата.

Рассмотрим разметку, с которой я начну работать. Обратите внимание на добавление тегу ARIA-роли, указывающей, что он для вспомогательной технологии расширения сферы доступности сайта играет роль кнопки, а не ссылки, как было бы по умолчанию:

```
<a class="menu-toggle js-activate-off-canvas-menu" role="button">
    <span aria-label="site navigation">&#9776;</span> menu
</a>
```

Пусть это и не самое лучшее, но зато вполне выверенное с практической точки зрения решение. Вот как выглядят эти два элемента (элемент `button` слева и тег `a` справа) в Firefox (версии 39.0a2, кому интересно) рядом друг с другом.



Разумеется, для данного упрощенного примера мы можем изменить отображение, перейдя от технологии flex к блочной технологии, и, экспериментируя с отступами, добиться эстетически приемлемой картины. Или же можем сохранить элемент `button` и вложить в него другой семантически незначимый элемент (`span`), из которого сделать Flex-контейнер. Но, какому бы подходу ни было отдано предпочтение, у каждого из них есть свои достоинства и недостатки.

В конечном счете выбор наиболее разумной разметки зависит только от нас. По одну сторону баррикад стоят разработчики, создающие разметку только с `div`- и `span`-контейнерами, чтобы гарантировать отсутствие нежелательных стилей от браузера. Расплачиваться приходится отсутствием смысловых значений, наследуемых от элементов, что выливается в отсутствие широкой доступности сайта. А по другую сторону стоят педанты разметки, желающие пользоваться только теми элементами, которые они считают правильными, независимо от того, насколько неприглядным получится конечный визуальный результат. Наша позиция находится где-то посредине, и я считаю ее наиболее продуманной и продуктивной.

Использование как можно более простого кода

От доступности новых эффективных технических приемов можно впасть в безудержную эйфорию. Памятуя об этом, нацельтесь на решение стоящих перед вами задач по достижению адаптивности с использованием как можно более простого кода. Например, если нужно придать стилевое оформление пятому элементу в списке и имеется доступ к разметке, не используйте селектор `nth-child`, как здесь:

```
.list-item:nth-child(5) {
    /* Стили */
}
```

При наличии доступа к разметке сделайте все проще, добавив к элементу HTML-класс:

```
<li class="list-item specific-class">Item</li>
```

После чего придайте ему стиль с помощью этого простого класса:

```
.specific-class {
    /* Стили */
}
```

Такой код не только будет проще восприниматься, но и позволит без каких-либо усилий с вашей стороны получить более широкую поддержку, поскольку устаревшие версии Internet Explorer не поддерживают селекторы `nth-child`.

Скрытие, показ и загрузка содержимого для всевозможных окон просмотра

Один из наиболее активно продвигаемых принципов адаптивного веб-дизайна гласит: если чего-то нет на экране при самом меньшем по размеру окне просмотра, этого не должно быть и на более крупных экранах.

Суть этого принципа заключается в том, что пользователи должны иметь возможность достичь одинаковых целей (покупка товара, чтение статьи, выполнение интерфейсной задачи) при любом размере окна просмотра. Определенный здравый смысл в этом, безусловно, есть. В конечном счете мы как пользователи будем разочарованы, когда зайдем на сайт с определенной целью и не сможем ничего сделать только потому, что наш экран меньше по размеру.

Кроме того, это означает, что при увеличении полезной площади экрана мы не должны считать себя вынужденными добавлять что-либо просто для заполнения пространства (к примеру, виджеты, рекламные объявления или ссылки). Если пользователь может обходиться без этих дополнений на экранах меньших размеров, он прекрасно обойдется без них и на более крупных экранах. Отображение дополнительного содержимого в более крупных окнах просмотра также означает, что содержимое либо имелось в окнах просмотра меньших размеров и просто находилось в скрытом состоянии (обычно с использованием `display: none;` в CSS), либо

было загружено при конкретном размере окна просмотра с помощью JavaScript. Короче говоря, либо содержимое было загружено, но оставалось невидимым, либо его видимость считалась излишней.

В целом я считаю, что хороший совет лучше принципов. Как бы то ни было, этот принцип заставляет дизайнеров и разработчиков более тщательно продумывать состав отображаемого на экране содержимого. Но как всегда бывает в веб-дизайне, ничто не обходится без исключений.

Я всеми силами сопротивляюсь загрузке новой разметки для различных окон просмотра, но иногда без этого просто не обойтись. Мне приходилось работать над сложными пользовательскими интерфейсами, для которых вполне оправданным требованием было использование в более широких окнах просмотра другой разметки и других конструкций.

В таких случаях для замены одной области разметки на другую использовался код JavaScript. Это сценарий был далек от идеала, но зато был наиболее практичным. Если по каким-то причинам JavaScript не мог работать, пользователи получали разметку экрана, предназначенную для экранов наименьшего размера. Они могли выполнять те же самые задачи, просто разметка для этого была не самой оптимальной.

По мере создания кода для все более сложного адаптивного веб-дизайна с подобными вариантами, вероятнее всего, придется столкнуться и вам, и в таком случае нужно будет положиться на собственные рассуждения по поводу наилучшего варианта для того или иного сценария. И тем не менее никто не станет осуждать вас за то, что для достижения своих целей вы воспользовались переключением видимости какой-либо части разметки с помощью свойства `display: none`.

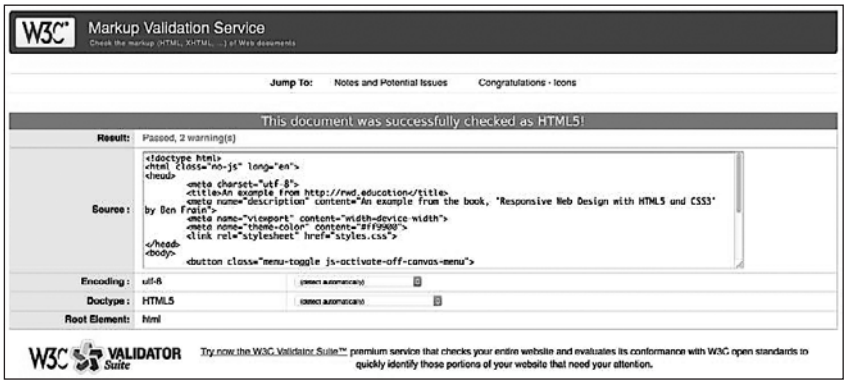
Возлагайте всю трудную работу по визуальному оформлению на CSS. Сложно не согласиться с тем, что JavaScript обеспечивает такой уровень интерактивности веб-страниц, которого просто невозможно достичь одними лишь средствами CSS. Но там, где это возможно, в области визуального оформления нужно стремиться возлагать всю самую трудную работу на CSS. На практике это означает отказ от исключительного использования JavaScript для анимации меню, его появления и исчезновения, включения и выключения (я имею в виду методы jQuery, используемые для показа и скрытия элементов). Вместо этого JavaScript следует использовать для простых изменений классов в соответствующих разделах разметки. Затем эти изменения будут инициировать анимацию или показ меню средствами CSS.

СОВЕТ

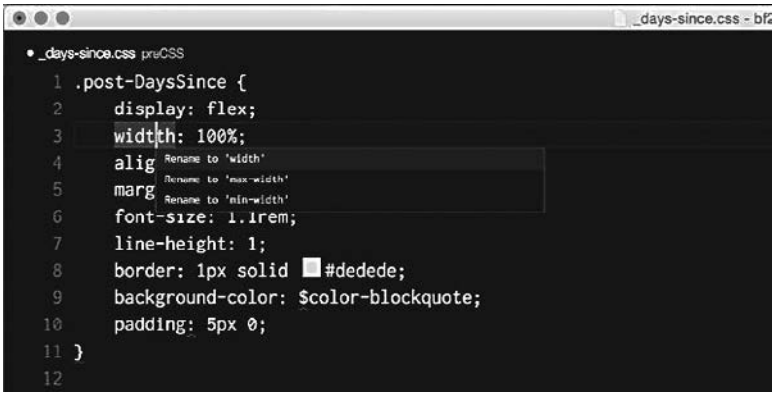
Чтобы добиться наивысшей производительности, при переключении классов в HTML добавляйте класс как можно ближе к тому элементу, к которому нужно применить эффект. Например, если нужно, чтобы блок появлялся поверх другого элемента, добавьте класс к самому ближнему общему родительскому элементу. Тогда ради достижения оптимальной производительности можно будет обеспечить внесение изменений только в данный конкретный раздел страницы и не заставлять браузер снова прорисовывать ее более крупные части. У Пола Льюиса (Paul Lewis) есть отличный бесплатный курс по повышению производительности под названием Browser Rendering Optimization, который можно найти по адресу https://www.udacity.com/course/browser-rendering-optimization--ud860.

Средства контроля качества кода

В общем-то, при написании кода HTML и CSS прощаются многие огрехи. Можно просчитаться с количеством вложенных элементов, забыть поставить кавычки или слеш в самозакрывающемся теге и при этом даже не всегда заметить какие-либо проблемы. Несмотря на это, мне чуть ли не еженедельно приходится сталкиваться с собственными просчетами в разметке. Иногда это ляпы вроде опечаток. А иногда школярские ошибки типа вложения `div` в `span` (разметка, при которой элемент блочного уровня `div` попадает в линейный элемент `span`, приводит к непредсказуемым результатам). К счастью, нам в помощь разработаны специальные инструментальные средства. В худшем случае, если возникли проблемы, обратитесь по адресу http://validator.w3.org/ и вставьте на этот сайт свою разметку. В результате будут помечены все ошибки и получены номера их строк, что существенно облегчит их устранение.



А еще лучше установить и настроить средства контроля качества вашего кода HTML, CSS и JavaScript. Или же выбрать текстовый редактор с определенным уровнем встроенной проверки правильности кода. Тогда по мере набора будут сразу же обозначаться проблемные области кода. Посмотрите на пример простой ошибки в написании кода CSS, которая сразу же была замечена редактором кода от компании Microsoft.



Я специально набрал `widtth` вместо `width`. Редактор это заметил и указал на ошибку, предлагая при этом ряд вполне разумных альтернатив. По возможности нужно пользоваться именно такими инструментальными средствами. Лучше потратить время на их подбор, чем на выискивание простых синтаксических ошибок в своем коде.

Производительность

Наряду с эстетическим восприятием не менее важно рассмотреть вопросы производительности своих адаптивных веб-конструкций. Но производительность представляется чем-то вроде движущейся цели. Например, браузеры обновляются и улучшают способы обработки ресурсов, взамен существующих оптимальных методов разрабатываются новые технологии, получающие со временем достаточный уровень браузерной поддержки и становящиеся на путь широкого внедрения. И этот перечень можно продолжить.

Но есть некоторые базовые особенности реализации, остающиеся практически неизменными (по крайней мере для большинства из них до широкого внедрения HTTP2 все останется как есть). К ним относятся следующие.

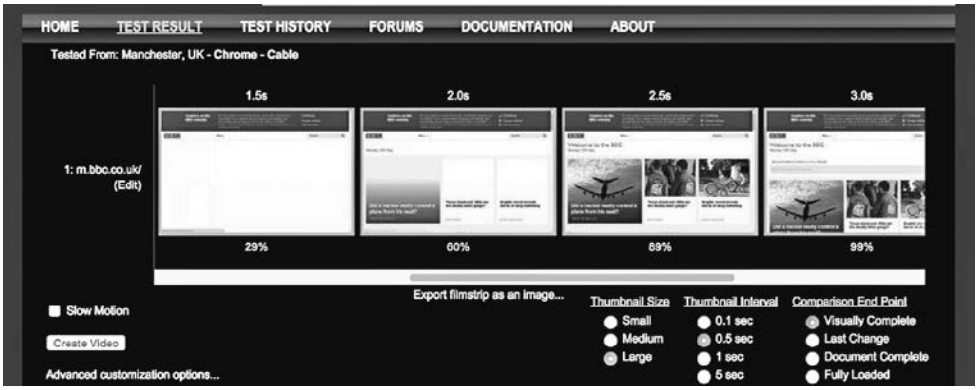
1. Сведение к минимуму количества ресурсов (например, не загружайте 15 файлов JavaScript, если объединяете их в одно целое).
2. Максимальное облегчение страницы (если есть возможность сжать изображения, значительно уменьшив их размер по сравнению с исходным, то так и нужно поступать).
3. Отсрочка загрузки второстепенных ресурсов (если можно отложить загрузку CSS и JavaScript до вывода страницы на экран, это может существенно улучшить восприятие скорости загрузки страницы).
4. Обеспечение как можно более ранней возможности использования страницы (обычно соблюдение этой рекомендации является побочным продуктом выполнения всех предыдущих).

Имеются также весьма эффективные средства для определения уровня производительности и ее оптимизации. Лично я предпочитаю пользоваться сайтом http://webpagetest.org/.

В наипростейшем варианте набирается URL-адрес и делается щелчок на кнопке START TEST. Сайт покажет полный анализ страницы, но, что еще полезнее, он покажет раскадровку страницы по мере ее загрузки, позволяя сконцентрироваться на как можно более быстром завершении вывода страницы на экран.

На следующей странице показано, как выглядит раскадровка загрузки главной страницы сайта BBC.

Перед попыткой оптимизации производительности нужно обязательно провести количественные измерения, иначе вы не сможете понять, насколько эффективной была работа по повышению производительности. Затем нужно внести поправки, выполнить тестирование и повторить цикл.



В преддверии великих перемен

Одно из обстоятельств, повышающих интерес к разработке веб-интерфейса, заключается в работе в условиях быстрых перемен. Всегда есть что-то новое для изучения, и веб-сообщество всегда придумывает способы, как при решении тех или иных задач все улучшить, ускорить и сделать намного эффективнее.

Например, за три года до выхода данного издания книги адаптивных изображений (получаемых с помощью атрибута `srcset` и элемента `picture`, подробно рассмотренных в главе 3) не было и в помине. Тогда, чтобы получить более подходящие для окон просмотра различных размеров изображения, нам приходилось пользоваться хитроумными обходными средствами от сторонних разработчиков. Теперь же, когда эти насущные потребности нашли свое отражение в стандартах W3C, мы можем с удовольствием пользоваться стандартными средствами.

Аналогично этому не так давно Flexbox еще только мерещился тем, кто создавал спецификацию. И даже по мере развития спецификации ее реализация давалась с большим трудом до тех пор, пока Андрей Ситник (Andrey Sitnik) и такие же умные, как он, ребята из Evil Martians (https://evilmartians.com/) не создали Autoprefixer, после чего мы смогли с относительной простотой воспользоваться кросс-браузерным кодом.

В будущем нам предстоит осваивать еще более захватывающие возможности. К примеру, в главе 4 уже упоминалось средство Service Workers (http://www.w3.org/TR/service-workers/), предоставляющее один из лучших способов создания приложения на основе веб-технологий, способных работать в автономном режиме.

Есть также Web Components — коллекция стандартов, составленная из Shadow DOM (http://w3c.github.io/webcomponents/spec/shadow/), Custom Elements (http://w3c.github.io/webcomponents/spec/custom/) и HTML Imports (http://w3c.github.io/webcomponents/spec/imports/), которая позволяет создавать полностью предсказуемые и многократно используемые компоненты.

На подходе и другие более совершенные средства вроде CSS Level 4 Selectors (http://dev.w3.org/csswg/selectors-4/) и CSS Level 4 Media Queries, частично рассмотренные в главе 2.

И наконец, на горизонте забрезжили еще более грандиозные перемены, связанные с появлением протокола HTTP2. Он обещает все, что сейчас считается передовыми наработками, просто выбросить на свалку. Чтобы получить о нем более глубокое представление, предлагаю прочитать заметки Дэниела Стенберга (Daniel Stenberg) о том, что такое HTTP2, свободно распространяемые в PDF-формате. Или же в качестве краткого обзора прочтите великолепную статью Мэтта Уилкокса (Matt Wilcox) *HTTP2 for front-end web developers* (https://mattwilcox.net/web-development/http2-for-front-end-web-developers).

Резюме

Поскольку время нашего с вами общения подошло к концу, ваш покорный слуга, он же автор этой книги, надеется, что теперь в вашем распоряжении имеются все технологии и инструментальные средства, необходимые для того, чтобы приступить к разработке вашего следующего сайта или веб-приложения в адаптивной манере.

Я убежден, что расчетливый подход к веб-проектам наряду с внесением ряда изменений в существующие рабочие процессы, сложившиеся методики и технологии позволит вам создавать адаптивные веб-конструкции, обеспечивающие сайты с высокими показателями скорости работы, гибкости и легкости в поддержке, у которых будет потрясающий внешний вид независимо от устройств, с которых их будут посещать.

За время, проведенное вместе, мы охватили весьма обширный объем информации, рассмотрели методики, технологии, способы оптимизации производительности, спецификации, основы организации рабочего процесса, вопросы применения различных инструментальных средств и многое другое. Я не думаю, что кому-то удастся все это усвоить за один присест. Поэтому, когда в следующий раз вам нужно будет вспомнить тот или иной синтаксис или освежить в памяти что-либо относящееся к рассмотренным в книге приемам разработки адаптивных конструкций, я надеюсь, что вы снова вернетесь к углубленному изучению материалов, изложенных на ее страницах. Я же здесь буду ждать вашего возвращения.

А пока желаю вам удачи на вашем нелегком, но увлекательном пути разработки адаптивного веб-дизайна.

До новых встреч.