ГЛАВА 15

Преимущества и недостатки адаптивного проектирования

Угождать ответами тебе я не обязан.
Уильям Шекспир

Абстрактно говоря, адаптивный веб-сайт — это веб-сайт, который подгоняет отображаемый им контент к размеру фактической области просмотра. Такую функциональную возможность легко рекламировать. Достаточно прийти к клиенту и продемонстрировать чудеса совершенно нового веб-сайта, который волшебным образом приспосабливается к любым конечным устройствам пользователя, который он может использовать сегодня и завтра.

Это сильный коммерческий аргумент, который практически невозможно опровергнуть.

По этой причине веб-сайт, который может адаптировать отображаемый контент к целевому устройству, — необходимая вещь в наше время. Это не просто желание выглядеть красивым и умным — это бизнес-требование. Тем не менее, как это часто происходит, проблемы кроются не столько в стратегических, сколько в тактических деталях.

Как создать веб-сайт, который адаптирует свой контент к размеру устройства просмотра? Применить адаптивное веб-проектирование (Responsive Web Design — RWD).

Основы адаптивного веб-проектирования

Вообще говоря, на практике используются два взаимозаменяемых термина: *адаптивный* (adaptive) и *отзывчивый* (responsive). В некоторых словарях, в частности в словаре Merriam-Webster, термин *responsive* определяется как способность к адаптации. Тем не менее в области веб-проектирования этот термин иногда означает специальное качество, которое выходит за пределы концептуального уровня и больше относится к деталям реализации.

Описывая сайт как *адаптивный*, продавцы и клиенты имеют в виду, что он прекрасно отображается на любых используемых устройствах. С другой стороны, для разработчиков и проектировщиков адаптивный сайт — это сайт, созданный с помощью специального набора технологий и методов, которые в совокупности называются средствами *адаптивного веб-проектирования*. В этой главе мы исследуем преимущества и недостатки подхода RWD, привлекая внимание к возможным проблемам, с которыми вы столкнетесь, если станете вслепую использовать RWD для создания мобильных представлений.

Краткая история RWD

Еще десять лет назад мир веб-проектирования был расколот на два лагеря: настольные браузеры и все остальное. “Все остальное” включало в себя мобильные телефоны и некоторые маленькие устройства, такие как Windows CE. Ранние версии платформы ASP.NET даже имели специальный набор элементов управления с минимальным механизмом обнаружения браузера и средствами визуализации, ориентированными на мобильные устройства. Мобильные элементы управления так и не стали популярными, потому что мобильные устройства почти никогда не использовались для просмотра веб-страниц.

Все изменилось, когда компания Apple вывела на рынок iPhone. И еще больше все изменилось, когда появился iPad, таким образом раскалывая теперь уже мобильный лагерь на два сегмента: смартфоны и планшеты. Добавьте к этому растущее количество торговых марок и моделей смартфонов, и вы получите полную картину. Встроенный механизм обнаружения, ориентированный на синтаксическом анализе выражений в строках агентов пользователей, стал довольно неуправляемым. Если количество строк агентов пользователей (и их вариаций) измеряется тысячами, то определить идеальный контент для вывода на экран устройства становится кошмарной проблемой.

По этой причине требовался другой радикальный подход.

Этан Маркотт (Ethan Marcotte) предложил подход, который он назвал *адаптивным веб-проектированием*. С тех пор аббревиатура RWD становилась все более и более популярной, а методы, лежащие в основе RWD, были восприняты как единственный способ получить адаптивный веб-сайт, который приспосабливается к хост-устройству. В конечном счете подход RWD оправдал себя и теперь широко поддерживается популярными каркасами для проектирования, такими как Bootstrap (см. главу 9). В то же время RWD — это всего лишь один из возможных подходов — и возможно, даже не самый эффективный, когда дело доходит до мобильных устройств.

Подход RWD относительно легко освоить с помощью либо ручного программирования, либо готовых адаптивных HTML-шаблонов. Является ли этот подход самым эффективным способом обслуживания мобильных контентов — это вопрос, который необходимо тщательно исследовать, изучая проекты.

Медиа-запросы CSS

Существуют три краеугольных камня RWD: плавающая сетка, гибкие изображения и медиа-запросы CSS. Идея, лежащая в основе *плавающей сетки* (fluid grid), состоит в том, что самый внешний контейнер изменяет свой размер и адаптирует весь

содержащийся в нем контент к новому размеру. Подход RWD предназначен для макетирования контента, который можно легко продемонстрировать повторно в новом, меньшем или большем, контейнере. Изображения можно было бы легко изменить с помощью атрибутов HTML, но в таком случае они были бы просто растянуты независимо от художественного вкуса и размера загрузки.

Медиа-запросы CSS являются тем связующим звеном, которое необходимо для того, чтобы собрать воедино визуальные элементы HTML, плавающие стили и даже размер контейнеров.

Типы носителей

Медиа-запросы CSS (CSS media queries) — широко известный стандарт Консорциума Всемирной сети (W3C), появившийся несколько лет назад. Термин *медиа* относится к HTML-определяемым типам носителя, самыми популярными и широко используемыми среди которых являются `screen` (экран) и `print` (принтер). В стандарте HTML определены многие другие типы носителей, включая `handheld` (переносное устройство). На первый взгляд, тип `hendheld` позволяет идеально ограничить приложения таблицей стилей CSS только для мобильных устройств. К сожалению, никто из поставщиков смартфонов не поддерживает этот тип носителя.

Замечание

Когда Apple выпустила iPhone, компания приняла продуманное решение не поддерживать тип носителя `handheld`, потому что, как утверждали в компании Apple, разрешение переносных устройств, предлагаемых W3C, меньше, чем у iPhone. Другие поставщики просто последовали примеру Apple, тем самым ограничив набор типов носителей, ограничившись типами `screen` и `print`, и вынудили разработать язык запросов для дифференциации вывода на устройства, ориентируясь на тип носителя `screen`.

Возможно, это никогда не произойдет, но было бы хорошо, чтобы браузеры смартфонов настраивались на переносные устройства. Это полностью изменило бы смысл определения.

Атрибут `media`, используемый в элементе `LINK`, указывает на контекст, в котором будет использоваться загружаемый файл CSS. Значение по умолчанию, равное `screen`, как следует из имени, предполагает, что соответствующий файл CSS используется при выводе страницы на экран. В свою очередь, значение `print` указывает на то, что соответствующий файл CSS будет использоваться при печати страницы. Как правило, печать CSS отличается от вывода на экран CSS фоновыми изображениями, шрифтами и графическими стилями.

Простой язык запросов для экранных устройств

Поскольку существовал всего один основной тип носителя — `screen`, — в ответ на стремительный рост количества новых экранных устройств был разработан язык запросов, чтобы позволить использование разных таблиц стилей для разных сценариев. Именно эту идею выражает стандарт медиа-запросов CSS.

Браузер, поддерживающий стандарт CSS 3, принимает булево выражение в атрибуте media и выбирает соответствующий файл CSS только тогда, когда результат выражения равен true. Для того чтобы создать осмысленное булевское выражение, необходимо иметь параметры, объединяемые с помощью логических операций AND и OR. В табл. 15.1 представлено большинство популярных и используемых ключевых слов, с помощью которых реализуются медиа-запросы CSS в браузерах.

Таблица 15.1. Основные свойства стандарта медиа-запросов CSS 3

Свойства	Описание
width, hight	Задают ширину и высоту в пикселях окна просмотра предоставления, как правило, окно браузера
orientation	Возвращает строку книжной ориентации, если высота больше или равна ширине. Иначе возвращается строка альбомной ориентации
device-width, device-height	Задают ширину и высоту физического экрана устройства. На самом мобильном устройстве, где приложения выполняются в полноэкранном режиме, эти значения совпадают со свойствами width и height
aspect-ratio	Указывает отношение между свойствами width и height. Как правило, это значение вида “16/9”
device-aspect-ratio	Указывает на отношение между свойствами device-width и decive-height. Как правило, это значение вида “16/9”

Полный список свойств, которые можно использовать в выражениях медиа-запросов CSS, доступен на сайте http://www.w3.org/TR/css3-mediaqueries.

Стандарт медиа-запросов CSS уровня 4

В настоящее время разрабатывается новая версия стандарта медиа-запросов CSS, в которую будут добавлены новые свойства, облегчающие настройку таблиц стилей в соответствии с конкретными форм-факторами устройств. Свойства, добавленные в новый стандарт, приведены в табл. 15.2.

Таблица 15.2. Новые свойства, определенные в стандарте медиа-запросов CSS уровня 4

Свойства	Описание
scripting	Определяет, поддерживаются ли в текущем документе языки сценариев, таких как JavaScript
pointer	Определяет точность указывающего устройства, такого как мышь. Принимает значение none | coarse | fine. Значение coarse относится к сенсорным устройствам, таким как планшеты. Значение fine относится, как правило, к поддержке мыши

Окончание табл. 15.2

Свойства	Описание
`hover`	Определяет способность пользователя оставлять указатель парить над элементами страницы. Принимает значения `none | on-demand | hover`. Значение `hover` относится к подсказкам и настольным устройствам. Значение `on-demand` относится к таким ситуациям, как долгое нажатие кнопки мыши, чтобы вывести на экран дополнительные детали
`resolution`	Относится к разрешению устройства вывода, как правило, плотности пикселей. Может измеряться в разных единицах, включая `dppx`, `cm` и `dpi`

Одно из преимуществ новой версии стандарта заключается в том, что обнаружение устройств, отличающихся от настольных, осуществляется легче, если оно основано на проверке свойств `hover` и `pointer`. Например, свойство `hover`, которое имеет значение `hover`, ясно означает поддержку подсказок, а значит, настольных браузеров. Значение свойства, равное `on-demand`, обычно означает поддержку устройств, работающих под управлением операционной системы Android, в которых долгое нажатие является функцией, заданной по умолчанию. В то же время значение свойства `pointer`, равное `coarce`, предполагает, что основное устройство является сенсорным, вероятно, планшетом.

Создание условных таблиц стилей с помощью медиа-запросов

Теперь посмотрим, как можно использовать свойства медиа-запросов, для того чтобы переключаться между CSS-файлами на ходу. Ниже приведен пример выражения медиа-запроса.

```
<link type="text/css" rel="stylesheet" href="view480.css"
      media="only screen and (max-width: 480px)">
```

Браузер будет использовать файл `view480.css`, только если выражение медиа-запроса имеет значение `true`. В данном примере это происходит, если ширина экрана (окна просмотра браузера) не превышает 480 пикселей. Теоретически веб-страница содержит много элементов `LINK`, каждый из которых ссылается на свою таблицу стилей для конкретных условий. Процесс сравнения не останавливается на первом совпадении, а проходит по всему списку ссылок CSS. Разработчик страницы должен гарантировать, что медиа-запрос только одного элемента `LINK` имеет значение `true`. Браузеры заново проверяют список связанных CSS-файлов, если размер окна изменился из-за действия мыши или вращения экрана.

Как только браузер обнаруживает, что размер экрана или ориентация изменились, он извлекает наиболее подходящий CSS-файл. После этого контент страницы повторно визуализируется в соответствии с инструкциями текущей таблицы стилей. Некоторые свойства мультмедийных запросов (такие, как `width`, `hight` и `orientation`) изменяются динамически. Другие свойства, особенно те, которые впервые включены в стандарт CSS Media Queries Level, являются статичными, потому что они зависят от природы устройства.

Важное замечание

Медиа-запросы работают в двух режимах. Первый режим предназначен для того, чтобы сделать опыт взаимодействия с настольной системой более приятным для пользователей и сохранять внешний вид контента независимо от размера окна браузера. Назначение другого режима — предоставить пользователям приятный опыт взаимодействия с веб-сайтом на конкретном устройстве, отличающемся от настольной системы. Для этого разработчикам не обязательно работать со сложными строками агентов пользователей.

Короче говоря, RWD не делает различий между настольным браузером, размер которого стал равным 480 пикселям, и полноэкранным браузером на смартфоне. Значение имеет лишь размер экрана, а смартфоны и ноутбуки могут работать, используя различные формы соединения (Wi-Fi, 3G), и существенно отличаться по мощности центрального процессора. Этот факт лучше всего выражает сущность RWD. Он одновременно является и главным преимуществом RWD, и его основной слабостью.

Настройка вручную сеточной системы с мультимедийными запросами

Большинство адаптивных веб-сайтов основаны на определенных адаптивных каркасах, чаще всего Bootstrap. Как было показано в главе 9, каркас Bootstrap поставляется со своими собственными выражениями медиа-запросов СМИ и разделяет экран на четыре главных сегмента.

Можно, конечно, переписать параметры настройки Bootstrap или не использовать Bootstrap вообще. В этом случае вы просто получите набор стилей CSS, которые реализуют кустарную сеточную систему. Как уже говорилось, сеточная система является ядром адаптивного проектирования. Рассмотрим пример:

```
.mycontainer {
   clear: both;
   padding: 0;
   margin: 0;
}

/* Общие настройки столбца в контейнере */
.col {
   display: block;
   float: left;
}
/* Настройки столбца, зависящие от позиции: 4 элемента */
.span4_index4 {
   width: 25%;
}
.span4_index3 {
   width: 25%;
   background: #ddd;
}
.span4_index2 {
   width: 25%;
```

```
}
.span4_index1 {
   width: 25%;
   background: #ddd;
}
/* 2x2 при 800px */
@media only screen and (max-width: 800px) {
   .span4_index4 {
      width: 50%;
   }
   .span4_index3 {
      width: 50%;
   }
   .span4_index2 {
      width: 50%;
   }
   .span4_index1 {
      width: 50%;
   }
}

/* Вертикальная настройка при 400px */
@media only screen and (max-width: 400px) {
   .span4_index4 {
      width: 100%;
   }
   .span4_index3 {
      width: 100%;
   }
   .span4_index2 {
      width: 100%;
   }
   .span4_index1 {
      width: 100%;
   }
}
```

Класс CSS `col` определяет основные параметры логического столбца экрана, а классы `span4_*` относятся к представлению экрана в виде сетки, состоящей из 1–4 колонок, которые называются `spans` (промежутки). Классы `span4_*` пронумерованы от 1 до 4 и одинаково разделяют свободное место. Медиа-запросы используются, чтобы решить, сколько столбцов вы хотите видеть на экране данного размера.

В предыдущем примере у нас есть два явных условия плюс условие по умолчанию.

```
@media only screen and (max-width: 800px) { ... }
@media only screen and (max-width: 400px) { ... }
```

По умолчанию будет четыре столбца. Однако, если максимальная ширина экрана меньше 800 пикселей, будет два столбца на строку. Если ширина экрана составит меньше 400 пикселей, то там будет только один столбец на строку. Ниже приведена HTML-разметка страницы (рис. 15.1).

```
<div class="mycontainer">
   <div class="col span4_index1">
      Column #1
   </div>
   <div class="col span4_index2">
      Column #2
   </div>
   <div class="col span4_index3">
      Column #3
   </div>
   <div class="col span4_index4">
      Column #4
   </div>
</div>
```

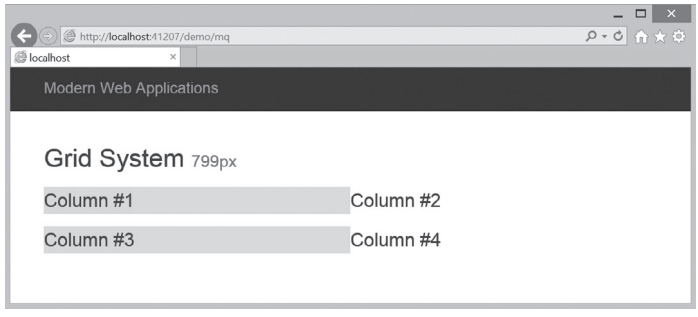

Рис. 15.1. *Простая сеточная система, настроенная вручную*

RWD и независимость устройства

Многие разработчики и технические руководители очень хорошо помнят, каким кошмаром было десять лет назад создание веб-сайтов для разных браузеров. Было время, когда, скажем, у Internet Explorer был свой набор функций, который отличался от функций Firefox, Safari и даже от своей более ранней версии. Это превращало создание разметки для страниц в настоящий хаос. Если разнообразие браузеров еще десять лет назад пугало вас, то будьте уверены, что разнообразие мобильных устройств намного хуже.

Трудно распознать устройство? Тогда не делайте этого. Проявите благоразумие и используйте вместо этого адаптивное веб-проектирование. Это — лозунг апологетов подхода RWD. Конечно, подход RWD предназначен для того, чтобы поставлять годный к употреблению контент на мобильные устройства, даже не зная, что целевое устройство не является настольным компьютером. Вопрос состоит в том, всегда ли эффективен подход RWD.

Нельзя сказать, что он всегда эффективен, это зависит от проектов, контекстов и бизнес-сценариев.

Парадокс RWD: обслуживайте всех, игнорируйте всех

Когда вы соединяетесь с веб-сайтом, сервер посылает вам пакет размеченного контента. В соответствии с принципами RWD вы не должны выполнять анализ строки агента пользователя и можете благополучно игнорировать характеристики запрашивающего браузера. Согласно RWD, сервер обслужит контент независимо от URL запрашивающего агента пользователя, платформы хост-сервера и возможностей устройства. Затем, как только содержание благополучно загрузится на клиентском компьютере, происходит некое волшебство.

Все, что может произойти в пределах контекста клиентского веб-браузера, можно сделать с помощью синтаксиса CSS и атрибутов.

- Изменение расположения нескольких элементов HTML в пределах их контейнеров.
- Смещение элементов к левому или правому краю контейнера.
- Изменение размеров контейнерных элементов с использованием абсолютных или относительных величин.

Но больше, чем все остальное, синтаксис CSS используется для показа и сокрытия элементов. Функция показать/скрыть используется наиболее часто. На реальных веб-сайтах, когда вы изменяете размеры страницы браузера, контейнеры иногда перемещаются к концу страницы. Но чаще они просто становятся скрытыми.

Парадокс RWD состоит в том, что, игнорируя тип и возможности запрашивающего устройства, вы вынуждаете сервер потенциально обрабатывать большое количество контента только для того, чтобы скрыть его с помощью некоторого класса CSS. Пользователи занимают полосу пропускания и тратят ненужные секунды или минуты своей жизни только потому, что ваш веб-сайт загружает контент, который они никогда не увидят (рис. 15.2)!

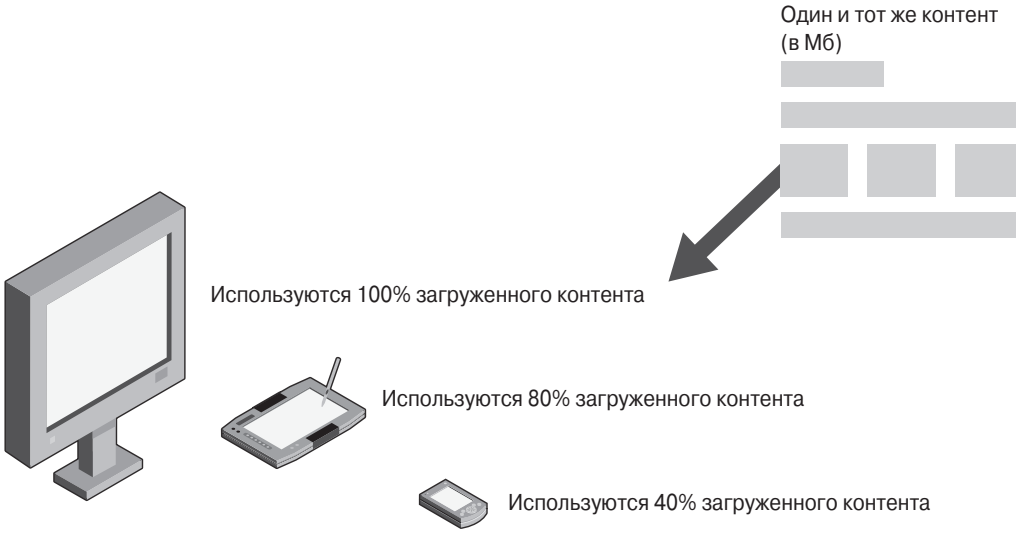

Рис. 15.2. *Один и тот же контент загружается на любой агент пользователя независимо от устройства*

Сказка о веб-проектировании, ориентированном в первую очередь на мобильные устройства

В основе RWD лежит аксиома, утверждающая, что шаблоны необходимо проектировать, ориентируясь в первую очередь на мобильные устройства. Этот подход, по существу, диктует, чтобы вы сначала создавали шаблон представления для маленьких экранов, а затем добавляли все новые и новые элементы контента для более крупных экранов.

Этот подход, возможно, оправдан, если вы обычный проектировщик, но если вы разработчик веб-страниц, то в результате всегда сталкиваетесь с необходимостью обработки отдельной порции контента. И этот контент всегда предназначен для самого большого экрана, котоорый может поддерживать приложение.

В целом выбор подхода, ориентированного на мобильные устройства в первую очередь, или подхода, ориентированного на мобильные устройства в последнюю очередь, которые также известны как *прогрессивное улучшение* (progressive enhancement) и *постепенная деградация* (graceful degradation) соответственно, зависит, главным образом, от предпочтений проектировщиков. Все зависит от того, как вы предпочитаете изучать новую систему.

С точки зрения веб-разработки стратегия, ориентированная в первую очередь на мобильные устройства, не дает никаких преимуществ, потому что она требует, чтобы сервер по-прежнему обрабатывал весь контент, пока не появится хороший механизм для распознавания устройств.

Адаптация RWD к мобильным устройствам

RWD — прекрасный подход, если речь идет о высокоуровневых устройствах, поскольку он затрагивает вопросы, связанные с вычислительной мощностью, графикой и экранами. На настольный компьютер загружается весь контент страницы. Вы платите за это один раз, и это, вероятно, огромная цена, потому что вы, вероятно, связаны с некоторой быстрой сетью Wi-Fi.

В данном контексте, если вы все еще в состоянии получить релевантную информацию в окне измененного браузера, это — замечательная особенность веб-сайта. Тот же самый механизм, который позволяет вам легко изменять размеры окна браузера, позволяет вам обслуживать мобильные браузеры без дальнейших усилий.

Все это хорошо, если не учитывать того, что измененное настольное окно браузера и полноэкранный браузер смартфона — совершенно разные вещи.

Важное замечание

Размер экрана нельзя рассматривать только как физическую величину. Несмотря на то что размер 15-дюймового экрана ноутбука или настольного компьютера совпадает с размером нескольких 5-дюймовых экранов смартфона, размещенных рядом, большие или маленькие экраны сильно различаются по дизайну и даже по сценариям использования. Иногда восприятие приложения изменяется, если оно рассматривается сквозь призму мобильных пользователей. Если вы слепо следуете принципам RWD, то не учитываете специфику устройств.

При использовании подхода RWD для обработки контента на устройствах, отличающихся от настольных систем, возникает необходимость оптимизировать несколько видов вашей работы. В первую очередь это относится к обработке изображений.

Работа с изображениями

Синтаксис CSS является прекрасным инструментом, если для адаптации приложения к меньшему размеру экрана достаточно переместить только несколько HTML-контейнеров. В HTML-контейнере, который, как правило, является блочным элементом вроде `DIV`, могут находиться обычный текст, текст разметки и изображения.

По разным причинам и текст, и изображения необходимо тщательно пересмотреть и проверить для всех возможных размеров экранов, которые вы планируете использовать. В частности, обработка изображений в рамках адаптивного веб-сайта может оказаться довольно проблематичной.

Попробуем разобраться в причинах и найти соответствующие решения.

Перемещение за пределы элемента `IMG`

В языке HTML5 есть всего один способ добавить изображение на веб-страницу — элемент `IMG`, который может указать на заданное изображение одним из двух способов.

- Отдельный URL (наиболее распространенный способ).
- Вложенный текстовый поток Base64.

Когда URL непосредственно указывает на графический файл, тот загружается, кешируется, а затем его размеры изменяются в зависимости от сочетания атрибутов `width` и `height`, указанных в стиле CSS. На первый взгляд, это замечательно. Действуя на ширину и высоту и выражая их в процентах, вы можете легко адаптировать изображение к контейнеру, сохраняя соотношение ширины и высоты и пропорции на странице.

Хотя этот подход вполне работоспособный, он часто оказывается слишком упрощенным по нескольким причинам.

- Вы загружаете изображения, которые, вероятно, намного больше, чем нужно на некоторых устройствах.
- Большое изображение может представлять собой крупный или дальний план, что затрудняет выбор меньшего размера для одного и того же изображения.

Почему вы должны загружать изображение объемом 2 Мбайт на маленькое устройство, если можно загрузить лишь его небольшой фрагмент? Можно ли сэкономить полосу пропускания и заставить устройство намного быстрее визуализировать страницу. Даже если размер загрузки не особо волнует вас, не забывайте, что изображение объемом 2 Мбайт может содержать как очень много деталей (например, крупный план какого-то объекта), так и очень мало (например, пейзаж на дальнем плане).

Когда изображение уменьшается, пиксели могут перепутаться или слиться в сплошное цветное пятно. Изменяя размеры изображения для веб-сайта, необходимо быть уверенным, что изображение не потеряет смысл. В этом случае устройство и ширина его экрана имеют большое значение.

Вперед — к адаптивным изображениям

Идеально было бы использовать разные изображения для разных устройств и разной ширины экрана. Идеально было бы, чтобы браузер выбирал и загружал самое подходящее изображение, основываясь на типах носителя и атрибутах СМИ. Это напоминает адаптивные изображения.

Ожидается, что в ближайшем будущем браузеры будут поддерживать новый элемент HTML для изображений, который не будет ограничиваться единственным исходным файлом. Согласно точке зрения консорциума W3C на адаптивные изображения, эта функция появится в следующем поколении веб-браузеров:

```
<picture alt="">
   <source media="(min-width: 800px)" srcset="large-1.jpg 1x, large-2.jpg 2x">
   <source media="(min-width: 600px)" srcset="med-1.jpg 1x, med-2.jpg 2x">
   <source srcset="small-1.jpg 1x, small-2.jpg 2x">
   <img src="small-1.jpg">
</picture>
```

Элемент `PICTURE` работает на двух уровнях, позволяя оптимизировать как полосу пропускания с помощью атрибута `srcset`, так и художественное руководство с помощью элемента `source`. Художественное руководство (art direction) — это деловые соображения, гарантирующие, что контент изображения является наилучшим для данного контекста. Иначе говоря, художественное руководство включает человеческий фактор, который устанавливает идеальный контент для заданного контекста и устройства.

Контентом атрибута `srcset` управляет браузер, который использует свой собственный алгоритм для выбора изображения, наиболее подходящего по размеру. Элемент `source`, наоборот, напоминает медиа-запросы: изображение выбирает автор страницы на основе сценария.

Когда художественное руководство действительно имеет значение

Для того понять роль художественного руководства при выборе самого подходящего изображения, взгляните на рис. 15.3. Эта страница находится в открытом доступе на сайте `http://www.expoware.org/wit.html`. Я взял изображение, сделанное с верхнего ряда открытой трибуны теннисного стадиона.

Поскольку это изображение представляет собой снимок, сделанный издалека и имеющий статичный фон, простое изменение размеров может привести к появлению квадрата, залитого сплошным цветом с почти неразличимым содержанием. Исходное изображение все же можно разумно уменьшить до размера 100×100, но это можно сделать только вручную.

Скрытые затраты, связанные с элементом `PICTURE`

В настоящее время элемент `PICTURE` является экспериментальным элементом HTML и по умолчанию поддерживается только в узком ассортименте браузеров, включая версии Chrome, Firefox и Opera, выпущенных во второй половине 2014 г. Internet Explorer и Safari его не поддерживают, но у Microsoft Edge такая поддержка есть.

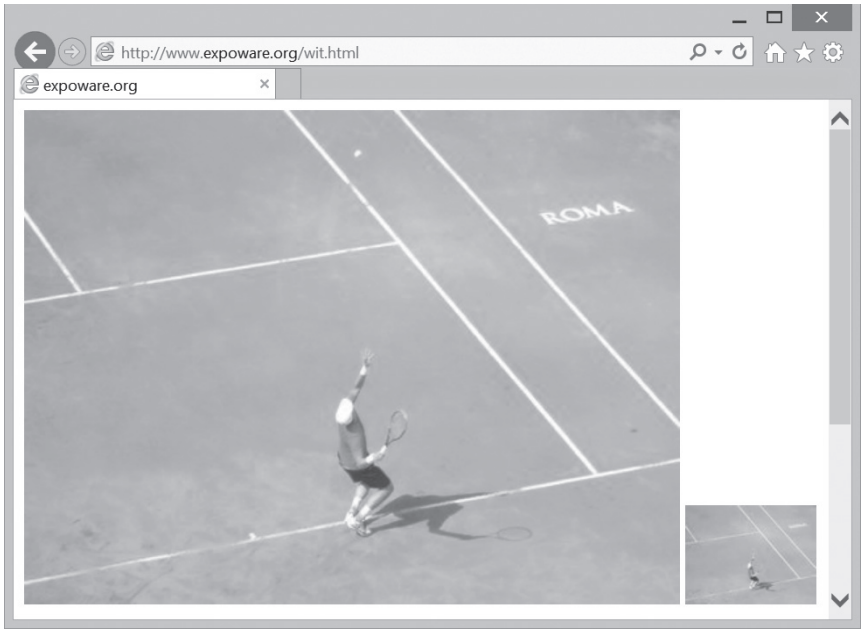

Рис. 15.3. *Уменьшенное изображение, генерируемое автоматически*

Среди мобильных браузеров сложилась аналогичная ситуация. Элемент `PICTURE` существует на Android-устройствах и везде, где доступен браузер Opera. Впрочем, существует полизаполнение (polyfill) с помощью библиотеки Modernizr, позволяющее использовать этот элемент в более широком диапазоне браузеров (см. `http://scottjehl.github.io/picturefill`).

Итак, если вы не собираетесь поддерживать длинный список старых браузеров, то использование элемента `PICTURE` — определенно выбор, достойный рассмотрения. Однако использование элемента `PICTURE` не свободно от проблем и может быть связано со скрытыми затратами.

Сила элемента `PICTURE` заключается в том, что он позволяет браузерам разумно выбирать самое подходящее изображение из диапазона вариантов. Поддержание многократных версий одного и того же изображения требует от проектировщиков дополнительной работы и приводит к хранению множества изображений на сервере. Это, вероятно, не проблема для относительно статичных изображений, таких как фоновые изображения или изображения, которые редко обновляются. Использование элемента `PICTURE` для реализации динамичных галерей может быть довольно дорогим удовольствием.

Замечание

Интересное решение, занимающее промежуточное положение между элементами `PICTURE` и `IMG`, представлено службой ImageEngine Lite. Эта служба представляет собой серверный инструмент, который действует как прокси, автоматически изменяет размеры и обрабатывает исходное изображение

так, как лучше всего подходит для устройства. Более подробно служба ImageEngine Lite описана в следующей главе. За дополнительной информацией обращайтесь на сайт `http://web.wurfl.io`.

Работа со шрифтами

Подход RWD позволяет перемещать контент в контейнерах переменного размера. А как насчет размера текста, содержащегося в нем? Рассмотрим, как эту задачу решает каркас Bootstrap, который широко используется для создания адаптивных представлений.

Абсолютные и относительные единицы

В каркасе Bootstrap параметры настройки для размера шрифта и высоты линии помещаются на уровень `BODY` и немного изменяются для других текстовых элементов, таких как `P`. Для увеличения и уменьшения размера шрифтов также используются классы CSS, такие как `small` и `lead`. До появления версии Bootstrap 3.x все размеры шрифта выражались в пикселях. Ожидается, что после появления версии Bootstrap 4 эта ситуация изменится.

Решение использовать пиксели в каркасе Bootstrap является спорным, но это определенно было разумным в то время, когда оно принималось. Разработчики думают прежде всего о пикселях, ведь именно пиксели обеспечивают неограниченный контроль над текстом, гарантируя согласованное предоставление во всех браузерах. Альтернативный подход предполагает выражение размера шрифта с помощью относительных единиц, таких как EM или REM.

Различие между EM и REM — довольно тонкое. Обе единицы выражают размер шрифта как долю чего-то: EM устанавливает его как долю размера шрифта непосредственного родительского шрифта, тогда как REM всегда ориентируется на базовый размер шрифта на уровне HTML. Используя для выражения размера шрифта единицу REM, вы работаете с фактическими числами только в одном месте — в корне, а все остальное измеряется автоматически.

```
html { font-size: 14px; }
h1 { font-size: 3rem; }
h2 { font-size: 2.5rem; }
h3 { font-size: 2rem; }
```

Обратите внимание на то, что основным аргументом в пользу шрифтов REM является их доступность. Размер шрифта, основанный на пикселях, не изменяется автоматически при увеличении системных шрифтов или шрифтов браузера для людей с проблемами зрения. Шрифты, измеренные с помощью единиц EM, наоборот, увеличиваются как доля размера шрифта браузера. Это означает, что единицы EM не снижают доступность, но создание доступных решений на основе единиц REM требует большего количества работы, потому что пиксели никогда не учитываются ни на одном уровне HTML.

В универсальных каркасах, таких как Bootstrap, использование шрифтов EM, вероятно, вызвало бы много проблем для множества браузеров. В итоге шрифты EM остаются прекрасным выбором для веб-сайтов, ориентированных на доступность.

С другой стороны, шрифты REM имеют смысл, если вы не хотите работать с визуальными контрольными точками самостоятельно. В этом случае вы устанавливаете основанный на пикселе шрифт на HTML-уровне и забываете о числах.

Единицы измерения окна просмотра

Другой способ выразить размер шрифта основан на использовании размеров окна просмотра браузера. В этом случае размер шрифта задается в процентах от размера окна просмотра:

```
h1 { font-size: 5.9vw; }
```

В данном примере размер элементов `H1` установлен равным 5,9% ширины окна браузера. Само собой разумеется, что, поскольку окно браузера изменено, размер шрифта изменится. Все это происходит автоматически без использования вами синтаксиса CSS.

Существует несколько вариантов единиц измерения окна браузера.

Можно использовать единицу VW, чтобы установить размер шрифта как процент ширины окна просмотра. Можно использовать единицу VH, чтобы установить размер шрифта как процент высоты окна просмотра. Наконец, можно использовать единицы VMIN и VMAX как долю минимального и максимального размера окна просмотра (по ширине или высоте).

Замечание

Меня всегда восхищало использование единиц окна просмотра для измерения шрифтов, но я всегда считал, что их довольно сложно использовать, особенно если вы используете их на страницах, которые будут просматриваться на мобильных и настольных устройствах. В настоящее время я применяю их только при разработке одного приложения, но в очень жестком сценарии: на страницах, созданных как экраны, которые должны храниться во фреймах или в контейнерах фиксированного размера.

Изменения в Bootstrap 4

Основная причина, по которой авторы Bootstrap не выбрали REM с самого начала, была связана с совместимостью браузеров. Например, у браузера Internet Explorer 8 нет поддержки измерений REM. Выбор REM сделал бы Internet Explorer 8 несовместимым с Bootstrap.

По этой причине каркас Bootstrap 3 использует пиксели, но это не мешает использовать шрифты REM или EM на вашей стороне. Каркас Bootstrap 4 не будет поддерживать браузер Internet Explorer 8. Впоследствии вы сможете выбирать наилучший размер шрифта без ограничений. Если вы намереваетесь гарантировать поддержку Internet Explorer 8, то придерживайтесь версии Bootstrap 3.x.

Работа с ориентацией

Одна из основных причин создания адаптивного сайта заключается в желании предоставить пользователям устройства возможность использовать контент с

максимально возможной свободой. При просмотре, например, на планшете сайт должен быть в состоянии отражать изменения ориентации с книжной на альбомную, и наоборот.

Как распознать изменение ориентации и что с ней делать? Прежде всего, это не аспект программирования, который можно обнаружить на стороне сервера. Это состояние, которое можно распознать только на стороне клиента.

Использование медиа-запросов

Для сайта изменение в ориентации является обычным изменением размеров окна, при котором устанавливаются другая высота и ширина. В зависимости от контента, с которым вы работаете, для реакции на изменения может оказаться достаточно стилей CSS. В таком случае лучшее, что можно сделать, — добавить несколько выражений медиа-запросов, как показано ниже.

```
@media screen and (orientation: portrait) {
/* стили для книжной ориентации */
}
@media screen and (orientation: landscape) {
/* стили для альбомной ориентации */
}
```

Если вы используете каркас Bootstrap, то вам не придется постоянно принимать какие-либо меры, чтобы явно поддерживать книжную и альбомную ориентации, потому что изменение ориентации будет распознаваться и обрабатываться как обычное изменение размеров. Но в более сложных ситуациях простого переключения CSS-файлов может оказаться недостаточно, и от клиента потребуется создание определенного кода на языке JavaScript.

Обнаружение события, связанного с браузером

Изменение ориентации можно распознать программно, используя событие `orientationchange` или событие `resize` для браузеров, которые не предлагают другого, более специфичного события. Когда вы обнаруживаете изменение размеров, то не можете непосредственно определить текущую ориентацию — `portrait` или `landscape`. Однако ее легко можно определить с помощью несложных математических вычислений.

Еще один вариант — применить метод `matchMedia` из объекта `window`:

```
var orientation = window.matchMedia("(orientation: portrait)");
orientation.addListener(function(portrait) {
   if(portrait.matches) {
      // Адаптировать контент к книжной ориентации
   }
   else {
      // Адаптировать контент к альбомной ориентации
   }
});
```

Метод `matchMedia` поддерживается на большинстве новых браузеров.

Резюме

Совершенно очевидно, что адаптивные веб-представления в наши дни абсолютно необходимы. Адаптивное веб-проектирование является практикой программирования, следуя которой можно создавать полностью адаптивные веб-представления. Поскольку этот подход поддерживается популярными каркасами, такими как Bootstrap, создание адаптивных представлений становится возможным практически для всех. Отметим, тем не менее, что каркас Bootstrap не совершает чудес — он просто предлагает специальные инструменты, которые разработчики и проектировщики могут использовать для создания адаптивных представлений.

Следует сказать, что создание адаптивных представлений — нелегкое дело. Они не работают одинаково во всех возможных сценариях. В основе адаптивного веб-проектирования лежат плавающие сетки — контейнеры переменных размеров, которые могут перемещать любой контент, предназначенный для этого. Относительно простая идея плавающей сетки требует большого объема сложного кода и, по существу, способности переключать таблицы стилей CSS на лету.

Именно для этого предназначены медиа-запросы CSS. Медиа-запросы выбирают таблицу стилей CSS, основываясь на булевом выражении, написанном относительно нескольких свойств браузера. Самое важное из этих свойств — ширина экрана. Дифференциация контента исключительно на основе ширины экрана не принимает во внимание реальные возможности соответствующих устройств. Подход RWD, по существу, считает обязательным игнорирование устройства — он лишь обрабатывает разные макеты в зависимости от ширины экрана.

В этой главе мы рассмотрели основную механику медиа-запросов. (Глава 9 содержит больше деталей о том, как создать плавающие сетки в реальных приложениях.) Кроме того, обсудили аспекты, позволяющие улучшить подход RWD.

В следующей главе мы исследуем решения, которые включают распознавание устройства и стратегию, ориентированную на мобильные устройства.