5 Масштабируемая векторная графика, холст, аудио и видео

Мы уже обсудили большую часть новых элементов HTML5, за исключением тех, которые уникально связаны с сетевыми API, пока находящимися в разработке, и еще одной категории элементов, о которых пойдет речь в этой главе. Это хорошо поддерживаемые элементы, относящиеся к работе с медиа и предназначенные для обращения с масштабируемой векторной графикой (SVG), холстом (Canvas), аудио и видео. Элементы, связанные с вышеупомянутыми веб-API, постоянно изменяются, поэтому мы не будем рассматривать их в этой книге. Элементам из второй категории посвящена эта глава.

Здесь будут рассмотрены основные возможности, которые пригодятся в повседневной практике веб-разработчику клиентского кода. Вы научитесь применять все новые возможности при программировании для мобильных браузеров. Все современные мобильные браузеры (за исключением Opera Mini) поддерживают элементы `<canvas>`, `<video>` и `<audio>`, а также веб-API, обеспечивающие геолокацию, работу с локальным хранилищем данных, создание офлайновых веб-приложений и т. д.

Можно было бы написать по книге на каждую из отдельных тем, рассмотренных в этой главе, — правда, на большинство тем такие книги уже есть. Надеюсь, что смогу дать здесь достаточное количество ознакомительной информации для того, чтобы читатель мог сам принять решение: «Ага, почитаю-ка книгу на эту тему» или «Нет, пока мне это неинтересно». Мы не будем углубленно исследовать ни одну из этих тем, но вы приобретете достаточно знаний, чтобы приступить к самостоятельному изучению того, что вас заинтересует. Гораздо важнее, что вы узнаете о достоинствах и недостатках всех этих технологий в мобильной сфере.

API HTML5 для работы с мультимедиа

Исходная спецификация HTML описывала работу исключительно с текстовым контентом. В ней даже отсутствует информация об элементе `<img>`. С тех пор немало воды утекло. HTML5 позволяет создавать масштабируемую векторную графику по технологии SVG, а также имеет элемент `<canvas>`, в котором содер-

жится пустое пространство для отрисовки — так называемый холст. Кроме графики, HTML5 поддерживает также включение в разметку элементов <video> и <audio>, причем для этого не требуется никаких сторонних плагинов.

SVG

С помощью технологии SVG можно создавать сложные изображения в формате *масштабируемой векторной графики*. Технология SVG появилась в 2001 году и представляет собой открытый стандарт для определения двухмерной векторной графики. «Масштабируемый» аспект SVG означает, что одна и та же графика будет иметь одинаковую резкость и на большом мониторе, и на маленьком дисплее мобильного устройства, не требуя никаких дополнительных изменений.

В спецификации SVG определяется XML-грамматика для фигур, линий, кривых, изображений и текста, в частности такие возможности, как прозрачность, произвольная геометрия, эффекты фильтров (тени, подсветка и т. д.), работа со сценариями и анимация.

Поскольку речь идет о текстовом формате для работы с изображениями, файлы в этом формате зачастую получаются очень маленькими. Такой файл имеет объектную модель, и ее можно изменять с помощью сценариев. Изображения векторные, и поэтому они масштабируются без возникновения мозаичности и зазубренных краев. Этот язык декларативен, и поэтому его легко понять. Вдобавок SVG поддерживает анимацию.

Существуют различные формы SVG, степень их подержки в браузерах различна. Базовая поддержка любых файлов с расширением .svg существует во всех современных браузерах для мобильных устройств с Android, в версии Android 3 и выше. SVG в качестве исходного кода для элемента <img> поддерживается в iOS 3.2 и выше, Android 3.0 и выше и в мобильной версии IE 8 и выше.

Формат файлов SVG в качестве значения для свойства CSS background-image поддерживается в Android 3 и выше, iOS 3.2 и выше, а также давно поддерживается в Opera Mobile. Мы даже можем использовать HTML-элемент <svg> на страницах HTML5 в iOS 5 и выше, Android 3 и выше, IE 9 и выше (а также во всех других современных браузерах). Android 2.3.3 и ниже, Amazon Silk и уже практически исчезнувшая система WebOS от HP — вот и все мобильные браузеры, в которых отсутствует полная поддержка SVG. Статическая SVG поддерживается даже в Opera Mini (как и элемент <canvas>, поддерживаемый во всех мобильных браузерах), но такая SVG не анимируется, поскольку в Opera Mini нет достаточной для этого поддержки JavaScript.

Как и в любом языке на основе XML, корневой элемент SVG — не <html>. В данном случае корневым элементом является <svg>. Как и все XML-документы, файл SVG начинается с XML-пролога и соответствующего объявления типа документа (SVG DTD). В корневом элементе <svg> содержится весь контент документа. В SVG не используются элементы <head> и <body>. В данном случае весь контент, в том числе все вложенные элементы <svg>, содержится в корневом элементе <svg>.

Начнем знакомство с SVG с изображения японского флага. Он представляет собой простой белый прямоугольник с красным кругом (солнцем) в центре (рис. 5.1).

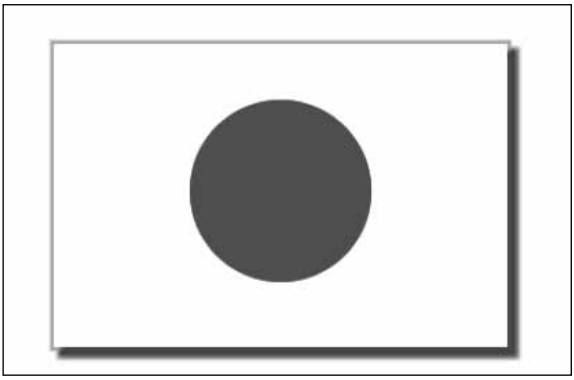

Рис. 5.1. Японский флаг, созданный по технологии SVG

```
1  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
2        "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
3  <svg xmlns="http://www.w3.org/2000/svg" height="220" width="320"
   version="1.0">
4    <title>Японский флаг</title>
5    <desc>Красный круг на белом флаге</desc>
6    <rect x="10" y="10" width="300" height="200"
7      style="fill: #ffffff; stroke: #e7e7e7;"/>
8  <circle cx="160px" cy="107px" r="60px" fill="#d60818"/>
9  </svg>5
```

Итак, что же это все означает? В строках 1–3 содержатся объявление документа SVG DTD и, соответственно, корневой элемент `<svg>`. О корневом элементе необходимо отметить следующее: размер векторной графики специально объявляется. Чтобы вы могли использовать свойство CSS `background-position` с фоновыми изображениями типа SVG, размер SVG должен быть специально объявлен. Это важно и при создании спрайт-файла на SVG.

Можете воспользоваться элементом `<title>` (строка 4), если файл SVG применяется независимо от других ресурсов. В строке 5 расположен элемент `<desc>`. Здесь можно поместить текстовое описание, которое не будет нативно отображаться в браузере, если уже отображается SVG. Если записать информативный контент в `<desc>` или `<title>`, то значительно повысится доступность ресурса. Поскольку SVG поддерживается не во всех экранных дикторах, для повышения доступности можно добавить атрибут `aria-label`.

В строке 6 находится элемент `<rect>`, соответствующий прямоугольнику. В SVG доступны следующие виды фигур и линий: `<path>`, `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<polyline>` и `<polygon>`. Мы располагаем четырьмя значениями атрибутов: `x`, `y`, `width` и `height` — для отступа по оси *X*, отступа по оси *Y* (координаты размещения фигуры

или линии), обозначения ширины и высоты соответственно. Кроме того, здесь применяется атрибут style.

Как и в обычном HTML-документе, в документе SVG можно использовать CSS для оформления элементов этого документа. Стили могут объявляться внутристрочно с помощью атрибута style, как было сделано в предыдущем примере. В качестве альтернативы можно включить в код встроенную или внешнюю таблицу стилей, указывая элементы с помощью селекторов, — точно так же, как это делается в HTML.

Имена свойств CSS немного отличаются от тех, к которым вы привыкли, но они приспособлены для чтения человеком. Свойство fill напоминает background. Оно обеспечивает цвет фона. Свойство stroke похоже на принятое в CSS свойство border. Мы могли бы задать здесь градиент или узор.

На практике существует возможность использовать большинство CSS-свойств и их значений в SVG-файлах. Однако по причинам, связанным с безопасностью содержимого, некоторые производители браузеров[1] не допускают импорта растровых изображений или сценариев при работе с файлами SVG, когда такие файлы включаются в страницу как изображения переднего плана в теге <img>.

В элементе <circle> (строка 8) описан круг красного цвета. У элемента <circle> вместо ширины и высоты есть атрибут r, означающий радиус. Элемент <circle> размещается на экране не в левом верхнем углу, как <rect>, а по центру круга. Значение cx — это координата по оси *X*, соответствующая центру круга, cy — координата по оси *Y*, соответствующая центру круга.

Внимательно изучив атрибуты, вы заметите, что параметр fill использован с элементом <rect> как свойство CSS, а с элементом <circle> — как атрибут.

Включение SVG в ваши документы

Можно включить масштабируемую векторную графику непосредственно в документ, воспользовавшись одним из следующих трех элементов: <img>, <object> и <embed>:

```
<img src="flag.svg" alt="Японский флаг"/>
```

или так:

```
<embed type="image/svg+xml" src="flag.svg" width="320" height="220"/>
```

или так:

```
<object data="flag.svg" type="image/svg+xml" width="320" height="220"></object>
```

Обратите внимание: хоть у элементов <embed> и <object> нет атрибута alt, SVG можно сделать доступной. Для улучшения доступности можно описать иллюстрацию в элементе <desc> или <title> либо добавить атрибут aria-label со значением,

[1] В настоящее время в браузерах WebKit и Mozilla не допускается импорт сценариев и растровых изображений в файлах SVG с применением тега <img>, даже если SVG и растровые изображения взяты из одного и того же источника.

соответствующим данному SVG-заголовку. Указывая высоту и ширину для <svg>, эти значения можно не включать в качестве атрибутов в <img>, <embed> или <object>, но необходимо включать в CSS.

Метод «автомобиль клоуна»: SVG для адаптивных изображений переднего плана

Масштабируемая векторная графика может использоваться для создания и подачи адаптивных изображений. Можно активно использовать браузерную поддержку SVG, а также предоставляемую в SVG поддержку медиазапросов и растровых изображений, чтобы выводить на экран нужный рисунок.

По опыту работы с фоновыми изображениями в CSS нам известно, что действительно можно загрузить только нужные картинки. Аналогично, чтобы SVG не скачала сразу все включенные изображения, используем в нашем SVG-файле фоновые картинки CSS, а не изображения переднего плана. При работе с адаптивными SVG мы включаем все файлы изображений, которые, возможно, понадобится предоставить, а потом показываем только нужные изображения в зависимости от поступающих медиазапросов (медиазапросы будут подробнее рассмотрены в главе 7):

```
<svg xmlns="http://www.w3.org/2000/svg"
  viewBox="0 0 300 329" preserveAspectRatio="xMidYMid meet">

<title>Поместите сюда старые атрибуты</title>

<style>
  svg {
    background-size: 100% 100%;
    background-repeat: no-repeat;
  }

  @media screen and (max-width: 400px) {
    svg {
      background-image: url(images/small.png");
    }
  }

  @media screen and (min-width: 401px) and (max-width: 700px) {
    svg {
      background-image: url(images/medium.png);
    }
  }

  @media screen and (min-width: 701px) and (max-width: 1000px) {
    svg {
      background-image: url(images/big.png);
```

```
    }
  }

  @media screen and (min-width: 1001px) {
    svg {
      background-image: url(images/huge.png);
    }
  }
</style>
</svg>
```

Чтобы сохранить нужное соотношение сторон объемлющего элемента и гарантировать, что он будет масштабироваться правильно, используются атрибуты `viewbox` и `preserveAspectRatio`. Значение атрибута `viewbox` — это четыре числа, разделенные пробелами или запятыми: `min-x` (минимум по оси *X*), `min-y` (минимум по оси *Y*), `width` (ширина) и `height` (высота). Определяя значения ширины и высоты области просмотра (viewbox), мы определяем соотношение сторон в SVG-файле.

Поскольку существуют некоторые проблемы безопасности при работе с `<img>` и импортированием растровых картинок в SVG, используем тег `<object>` для помещения на сайт адаптивных изображений. Элемент `<object>` позволяет интерпретировать внешний ресурс как изображение:

```
<object data="awesomefile.svg" type="image/svg+xml"></object>
```

По умолчанию `<object>` должен быть равен по ширине своему родительскому элементу. Но как и при работе с изображениями, мы можем задать высоту и ширину с помощью атрибутов `width` и `height` либо свойств CSS `width` и `height`. Поскольку в SVG-файле содержатся объявления `viewbox` и `preserveAspectRatio`, тег `<object>` по умолчанию будет поддерживать заданное соотношение сторон лишь при условии, что указана любая из величин, `width` или `height`.

Поскольку в данном случае используется тег `<object>`, а не `<img>`, мы не располагаем атрибутом `alt`. Чтобы обеспечить доступность такого метода, когда (и если) экранные дикторы начнут поддерживать SVG[1], необходимо гарантировать, что содержимое элемента `<title>`, используемого с SVG, в точности соответствовало информации, которую вы включили бы в атрибут `alt`.

В теге `<object>` SVG встраивается на страницу. SVG извлекает из имеющегося набора такое фоновое изображение, которое соответствует запросу `@media`. Нужное изображение определяется по размеру `<object>`, а не по величине области видимости. В приведенном ранее коде выполняются два HTTP-запроса: один для получения SVG, а другой — для получения изображения подходящего размера. Чтобы уложить всю эту информацию в единственный HTTP-запрос, указывайте в качестве значения атрибута `data` элемента `<object>` экранированный URI данных[2].

---

[1] http://www.iheni.com/just-how-accessible-is-svg/.

[2] Экранирование URI данных требуется для работы с браузером IE9 и выше. Это лишь краткий обзор метода «автомобиль клоуна». Подробное описание, примеры, а также резервные варианты для браузеров, не поддерживающих SVG, приводятся по адресу https://github.com/estelle/clowncar.

Я называю этот метод «автомобиль клоуна», так как мы умещаем множество больших изображений (клоунов) в один файл изображения SVG (автомобиль).

Изучение SVG

Пока мы лишь слегка коснулись темы SVG. Масштабируемую векторную графику можно сделать доступной, она подстраивается под любое разрешение экрана (то есть масштабируется), а также поддерживает анимацию на основе синтаксиса SVG или с применением JavaScript. Полный контроль над каждым элементом обеспечивается благодаря API объектной модели SVG-документа. Возможности SVG настолько широки, что детальное их обсуждение выходит за рамки этой книги. В спецификации W3C подробнее рассказано обо всех этих элементах, атрибутах и анимационном API.

Японский флаг — пример очень простого SVG. Обычно файлы SVG гораздо более сложные. Если вы имеете опыт работы с Adobe Illustrator, то, наверное, знаете, что эта программа позволяет экспортировать иллюстрации в формате SVG. Хотя это и удобный способ создания высокоточных SVG-файлов, при таком подходе создается много кода и работа с программой оказывается затратной.

Amaya — свободное ПО, поддерживающее упрощенную версию SVG, в частности простейшие фигуры, текст, изображения, технологию foreignObject, альфа-прозрачность, трансформации и анимацию. Программу Amaya можно скачать прямо на сайте W3C. Amaya помогает научиться работать с SVG, так как в этой программе можно просматривать и редактировать исходный код. Также можете познакомиться с Inkscape — свободно распространяемым редактором векторной графики, функционально схожим с Illustrator, CorelDraw или Xara. В Inkscape используется формат SVG, соответствующий стандарту W3C.

Масштабируемая векторная графика для игры CubeeDoo

В CubeeDoo мы дважды воспользуемся SVG. Во-первых, у нас будет SVG-спрайт для фонового изображения «фигурной» темы нашей игры. Во-вторых, будем работать с URI данных SVG для значка отключения звука.

Мы предлагаем пользователю на выбор несколько тем. В частности, есть темы с числами, фигурами и цветовая тема. Мы сможем создавать фигуры с помощью простого SVG-спрайта. Код, использованный для создания этого спрайта (спрайт для лицевой стороны одной из карточных колод показан на рис. 5.2), таков:

```
1  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
2      "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
3  <svg xmlns="http://www.w3.org/2000/svg" height="400" width="400"
   version="1.0">
4    <desc>Квадраты, круги, ромбы и треугольники</desc>
```

```
5
6    <!-- Цветные квадраты -->
7    <rect x="10" y="10" width="80" height="80" style="fill: #d60818;"/>
8    <rect x="10" y="110" width="80" height="80" style="fill: #ffff33;"/>
9    <rect x="10" y="210" width="80" height="80" style="fill: #00FF00;"/>
10    <rect x="10" y="310" width="80" height="80" style="fill: #0000FF;"/>
11
12    <!-- Цветные круги -->
13    <circle cx="150" cy="50" r="40" style="fill: #d60818;"/>
14    <circle cx="150" cy="150" r="40" style="fill: #ffff33;"/>
15    <circle cx="150" cy="250" r="40" style="fill: #00FF00;"/>
16    <circle cx="150" cy="350" r="40" style="fill: #0000FF;"/>
17
18    <!-- Ромбы -->
19    <polygon points="250,10 210,50 250,90 290,50" style="fill:
    #d60818;"/>
20    <polygon points="250,110 210,150 250,190 290,150" style="fill:
    #FFFF33;"/>
21    <polygon points="250,210 210,250 250,290 290,250" style="fill:
    #00FF00;"/>
22    <polygon points="250,310 210,350 250,390 290,350" style="fill:
    #0000FF;"/>
23
24    <!-- Треугольники -->
25    <polygon points="310,10 350,90 390,10" style="fill: #d60818;"/>
26    <polygon points="310,110 350,190 390,110" style="fill: #FFFF33;"/>
27    <polygon points="310,210 350,290 390,210" style="fill: #00FF00;"/>
28    <polygon points="310,310 350,390 390,310" style="fill: #0000FF;"/>
29  </svg>
```

В строке 1 находится объявление типа документа (DTD). В строке 3 объявляется корневой элемент, в нее также включаются данные о высоте и ширине SVG-изображения. Хотя спецификации этого и не требуют, необходимо включить эти атрибуты, если вы планируете использовать SVG-изображение в качестве фонового. В строке 4 находится описание, повышающее не только доступность, но и поисковую оптимизацию.

В строках 7–10 находятся объявления для четырех квадратов. Строка 9 означает: «Построй прямоугольник, начиная с точки, расположенной в 10 пикселах от левого края и в 210 пикселах от верхнего края. Задай для прямоугольника ширину 80 пикселов и высоту 80 пикселов. Заполни этот прямоугольник цветом #00FF00».

```
<rect x="10" y="210" width="80" height="80" style="fill: #00FF00;"/>
```

В строках 13–16 определяются четыре круга (диска). Строка 16 означает: «Найди точку, расположенную в 150 пикселах от левого края и в 350 пикселах от верхнего края и сделай ее центром круга радиусом 40 пикселов. Задай для этого круга фоновый цвет #0000FF».

```
<circle cx="150" cy="350" r="40" style="fill: #0000FF;"/>
```

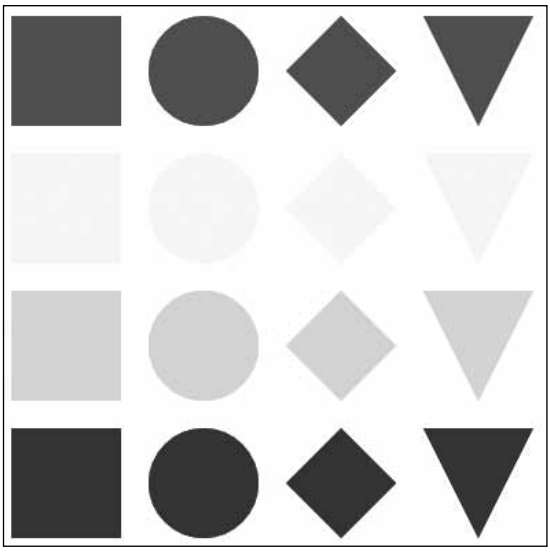

Рис. 5.2. SVG-спрайты фигур

В строках 18–28 определяются восемь многоугольников: четыре ромба и четыре треугольника. Для описания многоугольников определяются их вершины. Строка 19 означает: «У этого многоугольника четыре вершины, верхняя точка находится в 250 пикселах от левого края и в 50 пикселах от верхнего края. Вторая точка находится в 210 пикселах от левого края и в 50 пикселах от верхнего края. Нижняя точка находится в 90 пикселах от верхнего края, а самая правая — в 290 пикселах от левого края и 50 пикселах от верхнего края. Область, ограниченная этими четырьмя точками, должна быть заполнена цветом #d60818 — это оттенок красного».

```
<polygon points="250,10 210,50 250,90 290,50" style="fill: #d60818;"/>
```

Мы решили построить квадраты, круги, ромбы и треугольники, расположенные углом вниз.

Мы также могли бы включить эти маленькие картинки в виде URI данных непосредственно в CSS-файл либо оформить их как изображения переднего плана. Например, вот как можно включить закодированный SVG в виде URI данных:

```
background-image: url(data:image/svg+xml,%3Csvg%20xmlns%3D%22http%3A%2F%2F
www.w3.org%2F2000%2Fsvg%22%20version%3D%221.0%22%3E%3Crect%20x%3D%220%22%20y
%3D%220%22%20fill%3D%22%23abcdef%22%20width%3D%22100%25%22%20height
%3D%22100%25%22%20%2F%3E%3C%2Fsvg%3E);
```

В CubeeDoo у нас будет также значок для отключения звука. Вот URI данных для него:

```
background-image:
  background-image:
  url("data:image/svg+xml;utf8,%3Csvg%20xmlns=
'http://www.w3.org/2000/svg'%20width='100'%20height='100'%3E
```

```
%3Cpolygon%20points='39,13%2022,28%206,28%206,47%2022,48%2039,63%2039,14'%20
style='stroke:#111111;stroke-width:5;stroke-linejoin:round;
fill:#111111;'%20/%3E%3Cpath%20d='M%2048,50%2069,26'%20%20style='fill:none;
stroke:#111111;stroke-width:5;stroke-linecap:round'%20/%3E%3Cpath%20%20d='M%20
69,50%2048,26'%20style='fill:none;stroke:#111111;stroke-width:5;
stroke-linecap:round'%20/%3E%3C/svg%3E");
```

Пути в этом примере человеку прочитать почти невозможно. Они были созданы с помощью Amaya. Но синтаксис должен быть вам знаком. Здесь используется свойство CSS `background-image`. Вместо применения `url(path/mute.jpg)` или даже `url(path/mute.svg)` мы задействуем код `url("data:image/svg+xml;utf8,<svg.../></svg>");`. Весь SVG-файл экранирован и заключен в кавычки.

Для тех версий Internet Explorer, которые поддерживают SVG (в настоящее время это IE9 и IE10), URI данных должны экранироваться — этого требует спецификация.

Холст

Спецификация холста для HTML5 (HTML5 Canvas) — это API JavaScript для создания рисунков. API холста позволяет определять на HTML-странице объект контекста холста, представляемый в виде элемента `<canvas>`. В этом элементе можно рисовать. Можно даже включать в CSS-файлы изображения, отрисованные на холсте, и использовать их в качестве фоновых рисунков.

Можно рисовать как в двухмерном, так и в трехмерном (WebGL) контексте. Двухмерный контекст для рисования поддерживается во всех современных браузерах. WebGL все прочнее закрепляется в мобильном пространстве, но при необходимости эту технологию следует включать в программу только при наличии соответствующего аппаратного ускорения. Такое требование обусловлено соображениями производительности.

Двухмерный контекст для рисования предоставляет простой, но мощный API для выполнения быстрых операций отрисовки на плоской растровой поверхности. Специального файлового формата для этого не существует, вы можете рисовать только с помощью сценария. У вас не будет никаких узлов DOM для отрисовываемых фигур, ведь в `<canvas>` вы рисуете пикселы, а не векторы. При наличии единственного узла холст более удобен для использования на мобильных устройствах, но анимация JavaScript значительно нагружает процессор устройства и заряд батареи быстро расходуется. Однако эффективность использования батареи повышается, если применяется аппаратное ускорение.

Ваш первый <canvas>. Здесь будет сделано лишь базовое введение в работу с холстом, поэтому мы обсудим только простейшие фигуры и линии. Если вы не знаете JavaScript, то синтаксис на первый взгляд может показаться запутанным. Имеющим опыт работы с JavaScript разобраться будет гораздо проще.

Сначала нужно добавить элемент `<canvas>` в ваш документ. В HTML эта операция выполняется за один шаг:

```
<canvas id="flag" width="320" height="220">
  Ваш браузер не поддерживает холст. Иначе вы бы увидели флаг.
</canvas>
```

Вот и весь код холста, который относится к HTML. Я могла бы написать просто `<canvas></canvas>`. Атрибут `id` указан здесь для упрощения нацеливания при работе с JavaScript, хотя такое нацеливание можно реализовать и на уровне DOM. Кроме того, я включила сюда альтернативный контент для пользователей, у которых браузеры не поддерживают `<canvas>` либо содержимое `<canvas>` не видно по каким-то другим причинам.

ПРИМЕЧАНИЕ

На момент написания книги API элемента <canvas> реализован таким образом, что не поддерживает никаких средств обеспечения доступности, кроме атрибута aria-label.

Итак, мы создали пустое пространство для отрисовки, или холст. Все остальное будет происходить в коде JavaScript. В данном примере мы снова создадим японский флаг (рис. 5.3).

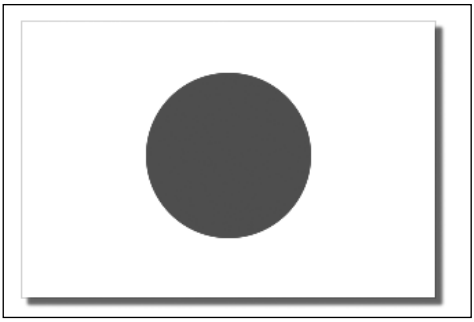

Рис. 5.3. Изображение японского флага, созданное на холсте

Приступим к рисованию на холсте. Далее вся работа идет только с кодом на JavaScript. Мы можем нацелиться на узел `<canvas>` с помощью простого JavaScript одним из следующих способов:

```
document.getElementById('flag')
document.getElementsByTagName('canvas')[0]
document.querySelector('#flag')
```

Затем инициализируем двухмерный контекст и начнем рисовать, пользуясь командами из API для двухмерных контекстов. Итак, рисуем японский флаг:

```
1  <script>
2  var el= document.getElementById("flag");
3
4  if (el && el.getContext) {
5    var context = el.getContext('2d');
6      if (context) {
7        context.fillStyle = "#ffffff";
```

```
8        context.strokeStyle = "#cccccc";
9        context.lineWidth = 1;
10       context.shadowOffsetX = 5;
11       context.shadowOffsetY = 5;
12       context.shadowBlur = 4;
13       context.shadowColor = 'rgba(0, 0, 0, 0.4)';
14       context.strokeRect(10, 10, 300, 200);
15       context.fillRect(10, 10, 300, 200);
16       context.shadowColor='rgba(0,0,0,0)';
17       context.beginPath();
18       context.fillStyle = "#d60818";
19       context.arc(160, 107, 60, 0, Math.PI*2, false);
20       context.closePath();
21       context.fill();
22       }
23  }
24  </script>
```

В строке 2 мы находим элемент `<canvas>` по его атрибуту `id`. Перед тем как создать двухмерный контекст, убеждаемся, что элемент-холст найден *и* браузер поддерживает холст. Для этого в строке 4 проверяется наличие метода `getContext`.

ПРИМЕЧАНИЕ

Можно задействовать сценарии обнаружения возможностей, например Modernizr. Они позволяют определить, поддерживает ли браузер работу с холстом и другие современные функции. Modernizr позволяет обнаруживать все доступные возможности либо отдельно взятые возможности, которыми вы планируете пользоваться. Здесь мы не применяем Modernizr, а рассматриваем, как обнаруживать возможности напрямую. Если только перед вами не стоит задача свести к минимуму использование внешних сценариев и HTTP-запросов, то Modernizr следует применять во всех случаях, когда это представляется целесообразным.

В строке 5 я создаю ссылку на контекст с помощью метода `getContext(contextId)` элемента-холста. `2d` — это как раз тот контекст, который нужен для работы с `<canvas>`. Если удалось создать контекст, что и проверяется в строке 6, то мы наконец можем приступать к рисованию. Весь оставшийся код сценария занят именно рисованием.

Уже существует (пока чисто экспериментальная) возможность использовать рисунок с холста в качестве фонового изображения. В браузере WebKit это можно делать с помощью CSS[1], не вызывая элемент-холст на уровне DOM, а включая его в качестве фонового изображения:

```
background: -webkit-canvas(theCanvas);
```

код для CSS и:

```
var context = document.getCSSCanvasContext("2d", "theCanvas", 320, 220);
```

код для JavaScript. В коде для JavaScript второй параметр — это имя используемого холста (в коде CSS это имя указывается без кавычек).

[1] Firefox 4+ также поддерживает работу с холстом в CSS и позволяет динамически создавать виртуальный элемент-холст с применением элемента -moz-element('#myCanvas').

К 6-й и даже к 13-й строке мы еще ничего не успели нарисовать. Все, что мы сделали, — это определили контекст холста, на котором можем отрисовывать и перерисовывать пикселы.

Перед отрисовкой фигуры необходимо определить для нее желаемый внешний вид. Для этого зададим свойства объекта `context`. Мы определяем внешний вид границ (`stroke` и `linewidth`), цвета фона (`fill`) и теней (`shadowOff`, `setX`, `shadowOffsetY`, `shadowBlur` и `shadowColor`) первого прямоугольника. Этот прямоугольник рисуем с помощью метода `strokeRect()` в строке 14. Передаем такие же параметры, как и в более раннем примере с SVG: 10, 10, 300, 200. Эти четыре значения указывают соответственно отступ по оси *X*, отступ по оси *Y*, ширину и высоту.

Когда сценарий выполнит команду, он о ней сразу же «забывает» и переходит к следующей строке кода. В отличие от примера с SVG в предыдущем разделе, тот прямоугольник, который отрисован на холсте, не входит в состав объектной модели документа. Поскольку `stroke`, `fill`, `linewidth` и `border` — это свойства, программа их запоминает, но браузер и сценарий «не представляют», что уже отрисовано. Если вы хотите отслеживать, что и где рисуется на холсте, используйте в этом контексте метод `getImageData()`. Данный метод позволяет контексту получать соответствующие вашим пикселам значения красного, зеленого, голубого и альфа-прозрачности.

Когда в строке 15 мы рисуем второй прямоугольник с помощью метода `fillRect` (этот метод при отрисовке прямоугольников использует заданное ранее свойство `fillStyle`), требуется вновь сообщить координаты, так как объектная модель документа не сохранила никакой информации о первом прямоугольнике (хотя она и имеет доступ к данным о пикселах).

Вызовы обоих методов для работы с прямоугольниками (в строках 14 и 15) имеют одни и те же параметры: 10, 10, 300, 200. Мы нарисовали прямоугольник с заливкой прямо поверх прямоугольника с оттенением. Мы могли бы создать объект с такими координатами и передать его обоим методам, но не можем приказать холсту получить доступ к координатам первого прямоугольника и скопировать их во второй после вызова метода.

Сначала мы нарисовали контур прямоугольника, потом заполнили его цветом. Если бы мы действовали в обратном порядке, то оттенение располагалось бы поверх цвета заливки. Поскольку начало координат у двух прямоугольников находится в одной и той же точке, а ширина границы составляет всего 1 пиксел, то ширина результирующей границы будет равна всего 0,5 пиксела. Внутреннюю часть границы накроет заливка.

Как было указано ранее, когда мы приступаем к отрисовке диска (солнца) на флаге, DOM должна «вспомнить», что вы нарисовали, как только диск окажется на холсте. Действительно, JavaScript фиксирует значения установленных вами свойств, например значение `shadowColor`. Кроме того, программа запоминает последние шаги отрисовки независимо от того, произошла ли в итоге отрисовка. Однако пикселы, создаваемые на холсте, — это всего лишь разноцветные точки. Поскольку нам не нужна тень у красного круга, перед отрисовкой `shadowColor` мы должны сделать его прозрачным. Это выполняется в строке 16.

Команды по отрисовке круга начинаются с `beginPath()` (строка 17) и заканчиваются `closePath()` (строка 20). Сценарий запоминает шаги отрисовки независимо от того, оказались ли соответствующие элементы на холсте. Если мы нарисуем круг, а потом, не закрывая контекста, еще несколько линий, то все этапы отрисовки круга будут сохраняться в памяти и линия может пересечь круг — допустим, разделить его пополам. Чтобы избежать этого, мы открываем и закрываем отрисовку отдельных контуров командами `beginPath()` и `closePath()` соответственно.

Описываем круг: `context.arc(x-offset, y-offset, radius, startAngle, endAngle, anticlockwise)` добавляет точки для дугообразного контура, создавая виртуальную окружность. Это круг, описываемый аргументами `context.arc(160, 107, 60, 0,Math.PI*2, false);`. В качестве начальной точки выберем стартовый угол, который послужит нам правым горизонтом. Конечную точку определим по заключительному углу. Проведем линию от начального угла в заданном направлении (в нашем случае — по часовой стрелке). Если конечный угол окажется меньше 2π, то круг получится уплощенным: начальная и конечная точки будут соединены прямой линией. При значении π получится полукруг.

Кроме того, переопределим цвет заливки с белого на красный (строка 18). Затем закрасим отрисованный круг с помощью метода `fill()` (строка 21), который заполняет область под описанной дугой цветом `fillStyle`.

Здесь мы практически не коснулись возможностей `<canvas>`. Рекомендую посмотреть сайт http://ie.microsoft.com/testdrive/Graphics/CanvasPad/Default.html — на этой интересной странице можно научиться работать на холсте с простыми фигурами, цветами, тенями, текстом, изображениями, трансформациями, анимацией и движениями мыши.

Сравнение холста и масштабируемой векторной графики

Технологии Canvas HTML5 и SVG обладают некоторым сходством, их часто сравнивают, но нередко подчеркивают и различия между ними. Обе эти веб-технологии позволяют создавать в браузере насыщенную графику, но на самом деле между ними существует фундаментальная разница.

Как мы уже убедились, в SVG отрисовка происходит на языке XML. Напротив, на холсте мы рисуем с помощью JavaScript. На холсте рисуются отдельные пикселы: как только пиксел оказывается на своем месте, программа о нем «забывает». SVG же создает узлы объектной модели документа. Эти узлы остаются доступными до тех пор, пока не будут удалены либо пока пользователь не покинет страницу. У обеих технологий есть свои достоинства и недостатки.

Рисунки, выполненные в виде масштабируемой векторной графики, не зависят от разрешения экрана. Поэтому SVG отлично подходит для применения в пользовательских интерфейсах любых размеров, масштабируется и подстраивается под величину конкретного экрана. Файлы SVG создаются в формате XML, благодаря чему обеспечивается их доступность. SVG можно анимировать с помощью