Публикация на тему

API холст

JavaScript API Canvas

Автор
Михалькевич Александр Викторович

Публикация

Наименование API холст
Автор А.В.Михалькевич
Специальность JavaScript API Canvas,
Анотация
Anotation in English
Ключевые слова
Количество символов 461626

Содержание

Введение
1 Основы рисования в браузере
2 Прямоугольник
3 Цвет
4 Градиент
5 Пути
6 Маска
7 Дуги
8 Кривые
9 Текст
10 Тени
11 Трансформация
12 Комбинирование фигур
13 Обработка изображений
14 Узоры
15 Анимация
16 Видео
Заключение
Список использованных источников
Приложения

Введение

В публикации рассматривается API JavaScript - Canvas

1 Основы рисования в браузере

API canvas (холст) позволяет рисовать графические элементы, выводить на экран изображения из файла, анимировать и обрабатывать рисунки и текст. Используя его совместно с другими API можно создавать двухмерные и даже трехмерные игры для Сети.

Элемент создает пустой прямоугольник, внутри которого визуализируются результаты применения методов рисования.

```
<html lang="en">
<head>
  <title>Canvas API</title>
  <script src="canvas.js"></script>
</head>
<body>
  <section id="canvasbox">
    <canvas id="canvas" width="500" height="300"></canvas>
  </section>
</body>
</html>
```

В файле canvas.js подготовим холст к рисованию

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
}
addEventListener("load", initiate);
```

2 Прямоугольник

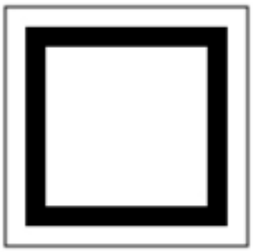
Для рисования прямоугольников доступны следующие методы:

fillRect(x, y, width, height) предназначен для рисования прямоугольника залитого цветом. Верхний левый угол фигуры будет находиться в точке заданной атрибутами x и y.
strokeRect(x, y, width, height) аналогичен предыдущему, но создает пустой, не залитый цветом, прямоугольный контур.
clearRect(x, y, width, height) предназначен для вычитания прямоугольной области, работает как прямоугольный ластик.

Применяя эти методы, нарисуем прямоугольник:

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.strokeRect(100, 100, 120, 120);
```

```
  canvas.fillRect(110, 110, 100, 100);
  canvas.clearRect(120, 120, 80, 80);
}
addEventListener("load", initiate);
```

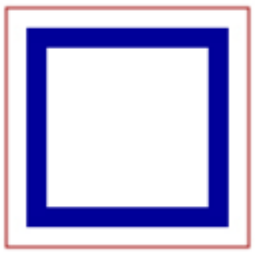


3 Цвет

Для определения свойства цвета можно применять синтаксис CSS со следу-ющими свойствами:

strokeStyle. Определяет цвет линий фигуры.
fillStyle. Определяет цвет внутренней области фигуры.
globalAlpha. Устанавливает уровень прозрачности.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.fillStyle = "#000099";
  canvas.strokeStyle = "#990000";
  canvas.strokeRect(100, 100, 120, 120);
  canvas.fillRect(110, 110, 100, 100);
  canvas.clearRect(120, 120, 80, 80);
}
addEventListener("load", initiate);
```



4 Градиент

Также ,как и в CSS3, градиенты могут быть линейными и радиальными. Возможно установление нескольких цветовых установок, создающих плавные переходы между множеством цветов. Методы:

createLinearGradient(x1, y1, x2, y2) создает объект градиента для последующей визуализации на холсте.

createRadialGradient(x1, y1, r1, x2, y2, r2) создает объект градиента, состоящий из двух окружностей. Значения в скобках представляют собой координаты центров окружностей и их радиусы.
addColorStop(position, color) – определяет цвета, которые будут использоваться для создания градиента. Атрибут position — это значение от 0,0 до 1,0, определяющее, в какой позиции начинается затухание цвета color.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  var grad = canvas.createLinearGradient(0, 0, 500, 500);
  grad.addColorStop(0.5, '#00AAFF');
  grad.addColorStop(1, '#000000');
  canvas.fillStyle = grad;
  canvas.fillRect(10, 10, 100, 100);
  canvas.fillRect(150, 10, 200, 100);
}
addEventListener("load", initiate);
```

5 Пути

Путь — это контур, вдоль которого следует перо, оставляя след. Путь может включать в себя различные виды штрихов: прямые линии, дуги, прямоугольники и т.д.

Рассмотрим два метода, предназначенные для создания путей и их закрытия:

beginPath(). Начинает новую фигуру.
closePath(). Закрывает путь, добавляя прямую линию между текущей точкой и исходной точкой пути.

Методы визуализации путей на холсте:

stroke(). Визуализирует путь в виде контура.
fill(). Визуализирует путь в виде залитой цветом фигуры.
clip(). Определяет область обрезки для контекста. Данный метод позволяет задать область обрезки произвольной формы, создав маску. Всё, что остается за пределами маски, на странице не отображается.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  // Здесь пути
  canvas.stroke();
}
addEventListener("load", initiate);
```

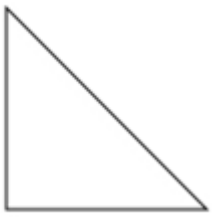
Данный код не создает никаких рисунков. Он лишь сигнализирует о созда-нии путей.

Для описания путей и создания реальной фигуры предназначены следующие методы:

moveTo(x, y). Перемещает кончик пера в указанную позицию.
lineTo(x, y). Создает отрезок между двумя точками: текущей позицией (например, определенной с помощью метода moveTo) и точкой с координатами x и y.
rect(x, y, width, height). Создает прямоугольник, который не сразу визуализируется на холсте, а становится частью пути.
arc(x, y, radius, startAngle, endAngle, direction). Создает дугу или окружность с центром в точке x, y, радиусом и угловым значением объявленным в атрибутах. Последний аргумент — это булево значение, задающее направление рисования: по часовой стрелке или против нее.
quadraticCurveTo(cpx, cpy, x, y). Создает квадратичную кривую Безье, начинающуюся в верхней позиции пера и заканчивающуюся в позиции с координатами x и y. Атрибуты cpx и cpy — это контрольные точки, управляющие формой кривой.
bizierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y). Аналогичен предыдущему, но имеет два дополнительных аргумента, позволяющих определить кубическую кривую Бизье.

Создадим треугльник с помощью описанных методов:

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  canvas.moveTo(100, 100);
  canvas.lineTo(200, 200);
  canvas.lineTo(100, 200);
  canvas.closePath();
  canvas.stroke();
}
addEventListener("load", initiate);
```



Чтобы нарисовать залитый цветом треугольник, необходимо вместо метода stroke() использовать метод fill().

```
canvas.fill();
```

6 Маска

Метод clip() предназначен для создания маски в форме пути, и таким образом, позволяет определить, что будет нарисовано, а что нет.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  canvas.moveTo(100, 100);
  canvas.lineTo(200, 200);
  canvas.lineTo(100, 200);
  canvas.clip();
  canvas.beginPath();
  for(var f = 0; f < 300; f = f + 10){
    canvas.moveTo(0, f);
    canvas.lineTo(500, f);
  }
  canvas.stroke();
}

addEventListener("load", initiate);
```

Цикл for() из листинга создает горизонтальные линии через каждые десять пикселов. Линии пересекают холст слева направо, но на странице мы видим только те фрагменты, которые попадают внутрь треугольной маски.

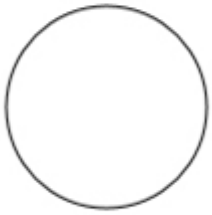
7 Дуги

Для создания фигур, включающих в себя различные дуги, в API предусмотрены специальные

методы.

Метод arc() предназначен для рисования окружностей или дуг. Обратите внимание на значение PI (данный метод ориентируется на значение угла в радианах, а не в градусах). Значение PI в радианах соответствует 180°. Формула PI x 2 в итоге дает 360°.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  canvas.arc(100, 100, 50, 0, Math.PI * 2, false);
  canvas.stroke();
}
addEventListener("load", initiate);
```



Для создания дуги с определенным углом в градусах нужно воспользоваться формулой:

Math.PI/180 x градусы

Дуга с углом в 45°.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  var radians = Math.PI / 180 * 45;
  canvas.arc(100, 100, 50, 0, radians, false);
  canvas.stroke();
}
addEventListener("load", initiate);
```

8 Кривые

Метод quadraticCurveTo() предназначен для создания квадратичной кривой Безье, а метод bezieCurveTo() – для рисования кубической кривой Бизье.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  canvas.moveTo(50, 50);
  canvas.quadraticCurveTo(100, 125, 50, 200);
  canvas.moveTo(250, 50);
  canvas.bezierCurveTo(200, 125, 300, 125, 250, 200);
  canvas.stroke();
}
addEventListener("load", initiate);
```

Ширину, вид и окончание линий можно настраивать. Для этого имеется че-тыре свойства:

lineWidth. Определяет толщину линии.
lineCap. Определяет форму окончания линии. Может принимать следующие значения: butt, round или square.
lineJoin. Определяет форму соединения двух линий. Возможные значения: round, bevel и miter.
miterLimit. Используется совместно со свойством lineJoin и определяет протяженность соединения двух линий в случае, если свойству lineJoin присвоено значение miter.

Перечисленные свойства влияют на весь путь. После каждого изменения характеристик линии необходимо создавать новый путь.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.beginPath();
  canvas.arc(200, 150, 50, 0, Math.PI * 2, false);
  canvas.stroke();
  canvas.lineWidth = 10;
  canvas.lineCap = "round";
  canvas.beginPath();
  canvas.moveTo(230, 150);
  canvas.arc(200, 150, 30, 0, Math.PI, false);
```

```
  canvas.stroke();
  canvas.lineWidth = 5;
  canvas.lineJoin = "miter";
  canvas.beginPath();
  canvas.moveTo(195, 135);
  canvas.lineTo(215, 155);
  canvas.lineTo(195, 155);
  canvas.stroke();
}
addEventListener("load", initiate);
```

9 Текст

Для добавления текста на холст нужно определить несколько свойств и вызвать подходящий метод.

Свойства Текста:

font. Синтаксис аналогичен CSS-ситнаксису свойства font

textAlign. Возможные варианты выравнивания по горизонтали. Описываются значениями start, end, left, right и center.

textBaseLine. Выравнивание по вертикали. Возможные значения: top, hanging, middle, alphabetic, ideographic и bottom.

Методы текста:

strokeText(text, x, y [, max-size]). Текст выводится в точках x, y. Возможено передавать четвертый параметр, определяющий максимальный размер текста.
fillText(text, x, y). Аналогичен предыдущему методу, но визуализирует текст, как залитые цветом фигуры.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.font = "bold 24px verdana, sans-serif";
  canvas.textAlign = "start";
  canvas.fillText("my message", 100, 100);
}
addEventListener("load", initiate);
```

Для работы с текстом еще есть один метод, который измеряет текст, - **measureText**(). Он возвращает информацию о размере указанного текста. Благодаря методу measureText() и свойству width можно узнать длину текста по горизонтали.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.font = "bold 24px verdana, sans-serif";
  canvas.textAlign = "start";
  canvas.textBaseline = "bottom";
  canvas.fillText("My message", 100, 124);
  var size = canvas.measureText("My message");
  canvas.strokeRect(100, 100, size.width, 24);
}
addEventListener("load", initiate);
```

10 Тени

Тени можно создавать для любых путей и текста. Для этого предусмотрены следующие свойства:

shadowColor. Цвет тени.
shadowOffsetX. Указание насколько нужно отступить от объекта по горизонтали.
shadowOffsetY. Указание насколько нужно отступить от объекта по вертикали.
shadowBlur. Размытость тени.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.shadowColor = "rgba(0, 0, 0, 0.5)";
  canvas.shadowOffsetX = 4;
  canvas.shadowOffsetY = 4;
  canvas.shadowBlur = 5;
  canvas.font = "bold 50px verdana, sans-serif";
  canvas.fillText("my message", 100, 100);
}
addEventListener("load", initiate);
```

11 Трансформация

Рассмотрим пять методов трансформации:

translate(x, y). Применяется для переноса начала координат.
rotate(angle). Поворачивает холст вокруг начала координат на указанный угол.
scale(x, y). Масштабирует все нарисованные на холсте элементы.
transform(m1, m2, m3, m4, dx, dy). Применяет новую матрицу трансформаций поверх текущей, модифицируя таким образом весь холст.
setTransform(m1, m2, m3, m4, dx, dy). Отменяет текущую трансформацию и определяет

новую на основе переданных в атрибуте значений.

Применимы к одному тексту методы translate(), rotate() и scale().

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.font = "bold 20px verdana, sans-serif";
  canvas.fillText("TEST", 50, 20);
  canvas.translate(50, 70);
  canvas.rotate(Math.PI / 180 * 45);
  canvas.fillText("TEST", 0, 0);
  canvas.rotate(-Math.PI / 180 * 45);
  canvas.translate(0, 100);
  canvas.scale(2, 2);
  canvas.fillText("TEST", 0, 0);
}
addEventListener("load", initiate);
```

Сперва мы нарисовали текст на холсте в точке с координатами (50, 20) с размером 20 px. После этого, с помощью метода translate() перенесли начало координат в точку (50, 70) и, с помощью метода rotate(), повернули холст на 45 градусов.

После этого, определенные в предыдущем шаге значения, считаются значениями по умолчанию. Поэтому, для того чтобы вернуть текст в исходное состояние, снова вызываем rotate() с такими же, но отрицательными значениями. Наконец, с помощью метода scale() увеличиваем масштаб холста.

Каждая последующая трансформация накладывается на предыдущую. Например, если мы применим масштабирование scale(2, 2), а затем еще раз scale(2, 2), то холст увеличится в четыре раза.

Для определения характеристик матрицы используются методы trasform() и setTransform().

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.transform(3, 0, 0, 1, 0, 0);
  canvas.font = "bold 20px verdana, sans-serif";
  canvas.fillText("TEST", 20, 20);
  canvas.transform(1, 0, 0, 10, 0, 0);
  canvas.font = "bold 20px verdana, sans-serif";
```

```
  canvas.fillText("TEST", 20, 20);
}
addEventListener("load", initiate);
```

Восстановление состояния

Из-за накопительного эффекта состояний трансформаций, возвращаться к начальному состоянию без специальных методов бывает затруднительно. Рассмотрим методы восстановления холста.

save(). Сохраняет состояние холста, включая все определенные для него ранее трансформации, значения свойств, стилей и т.д.
restore(). Восстанавливает последнее сохраненное состояние.

Отмена предыдущих трансформаций:

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.save();
  canvas.translate(50, 70);
  canvas.font = "bold 20px verdana, sans-serif";
  canvas.fillText("TEST1", 0, 30);
  canvas.restore();
  canvas.fillText("TEST2", 0, 30);
}
addEventListener("load", initiate);
```

12 Комбинирование фигур

Для определения каким образом фигуры, выводящиеся на холст, должны комбинироваться с другими фигурами, существуео свойство globalCom-positeOperation. Рассмотрим возможные значения данного свойства:

source-over – новая фигура визуализируется поверх уже имеющихся на холсте.
source-in – визуализируется только та часть фигуры, которая перекрывает предыдущую фигуру.
source-out – визуализируется только та часть фигуры, которая не перекрывает предыдущую.
source-atop – визуализируется только та часть фигуры, которая перекрывает предыдущую фигуру. Предыдущая фигура сохраняется целиком, но остальные фрагменты новой фигуры становятся прозрачными.
lighter – визуализируются обе фигуры, но цвет перекрывающихся путей определяется путем сложения цветовых значений.
xor – визуализируются обе фигуры, но перекрывающиеся фрагменты становятся прозрачными.
destination-over – это противоположность значению по умолчанию. Новые фигуры визуализируются позади фигур уже добавленных на холст.
destination-in – сохраняются только те фрагменты существующих фигур, которые перекрываются новой. Все остальные, включая новую фигуру, становятся прозрачными.
destination-out – сохраняются только те фрагменты существующих фигур, которые не

перекрываются новой фигурой. Все остальные, включая новую фигуру, остаются прозрачными.
destination-atop – существующие фигуры и новая фигура становятся прозрачными, за исключением тех фрагментов, где они перекрываются.
darker – визуализируются обе фигуры, но цвет перекрывающихся фрагментов определяется вычитанием цветовых значений.
copy — визуализируется только новая фигура, остальные становятся прозрачными.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  canvas.fillStyle = "#666666";
  canvas.fillRect(100, 100, 200, 80);
  canvas.globalCompositeOperation = "source-atop";
  canvas.fillStyle = "#DDDDDD";
  canvas.font = "bold 60px verdana, sans-serif";
  canvas.textAlign = "center";
  canvas.textBaseline = "middle";
  canvas.fillText("TEST", 200, 100);
}
addEventListener("load", initiate);
```

13 Обработка изображений

Для работы с изображениями предусмотрен только один метод: drowImage(). Возможные варианты использования:

drowImage(image, x, y). Вывод изображения в точку с координатами x и y.
drowImage(image, x, y, width, height). Таким образом, можно масштабировать изображение, прежде чем его помещать в холст.
drowImage(image, x1, y1, width1, height1, x2, y2, width2, height2). Таким образом, можно отрезать часть изображения и вывести его в указанной точке холста, одновременно поменяв размер. Значения x1 и y1 определяют координаты верхнего угла отрезаемого фрагмента изображения. Значения width1 и heigh1 задают размер этого изображения. Остальные значения (x2, y2, width2, height2) объявляют точку, в которой будет выводиться изображение и его размер.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas=elem.getContext('2d');
  var img = document.createElement('img');
  img.setAttribute('src', 'http://obmenka.by/media/img/we.jpg');
```

```
  img.addEventListener("load", function(){
    canvas.drawImage(img, 20, 20);
  });
}
addEventListener("load", initiate);
```

Изменение размера изрображения

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  var img = document.createElement('img');
  img.setAttribute('src', 'http://www.minkbooks.com/content/snow.jpg');
  img.addEventListener("load", function(){
    canvas.drawImage(img, 0, 0, elem.width, elem.height);
  });
}
addEventListener("load", initiate);
```

Т.к. холст может работать только с загруженными изображениями, мы поместили метод drowImage в анонимную функцию, которая вызывается прослушивателем addEventListener по событию load. Таким образом, метод drowImage() внутри функции выводит изображение только в после того, как загрузка завершена.

```
function initiate(){
  var elem = document.getElementById('canvas');
  var canvas = elem.getContext('2d');
  var img = document.createElement('img');
  img.setAttribute('src', 'http://www.minkbooks.com/content/snow.jpg');
  img.addEventListener("load", function(){
    canvas.drawImage(img, 135, 30, 50, 50, 0, 0, 200, 200);
  });
}
addEventListener("load", initiate);
```

Кроме метода drowImage(), который работает непосредственно с изображением, существует еще несколько методов, работающих с данными полученного изображения. Рассмотрим три метода для обработки изображения.

getImageData(x, y, width, height). Считывает прямоугольную часть холста и преобразует ее в массив с данными.
putImageData(imagedata, x, y). Превращает данные, на которые ссылается imagedata в изображение и выводит его на холст в точку с координатами x и y. Таким образом, это противоположность методу getImageData().
createImageData(width, height). Создает данные для пустого изображения. Все пикселы пустого изображения черные пикселы.

Каждое изображение можно представить в виде последовательности целых чисел, соответсвующих компонентам RGBA (по четыре значения на каждый пиксел). Группа значений, несущих такую информацию, составляют одно-мерный массив. Позиция каждого из элементов массива вычисляется по формуле

(width x 4 x Y) + (X x 4) = соответствует красному цвету

Результат вычислений соответствует первому пикселу. Для получения цвета для остальных компонентов необходимо прибавлять по единице для каждого компонента

(width x 4 x Y) + (X x 4) + 1 = зеленый

(width x 4 x Y) + (X x 4) + 2 = синий

(width x 4 x Y) + (X x 4) + 3 = альфа-канал

Негатив изображения:

```
var canvas, img;
function initiate(){
  var elem = document.getElementById('canvas');
  canvas = elem.getContext('2d');
  img = document.createElement('img');
  img.setAttribute('src', 'snow.jpg');
  img.addEventListener("load", modimage);
}
function modimage(){
  canvas.drawImage(img, 0, 0);
  var info = canvas.getImageData(0, 0, 175, 262);
  var pos;
  for(var x = 0; x < 175; x++){
    for(var y = 0; y < 262; y++){
      pos = (info.width * 4 * y) + (x * 4);
      info.data[pos] = 255 - info.data[pos];
      info.data[pos+1] = 255 - info.data[pos+1];
      info.data[pos+2] = 255 - info.data[pos+2];
    }
  }
  canvas.putImageData(info, 0, 0);
}
addEventListener("load", initiate);
```

Ширина изображения в примере равна 350 px, высота — 262 px. Поэтому, передавая методу getImageData параметры (0, 0, 175, 262), мы вырезаем половину исходного изображения. Вырезанное изображение сохраняется в переменную info. Метод getImageData возвращает объект, который можно обработать, обратившись к его свойствам width, height, data.

Далее, для того, чтобы создать негатив изображения, необходимо обработать каждый пиксел исходной части изображения. Для описания каждого цвета используется значение от 0 до 255. Следовательно, чтобы получить негатив цвета, нужно вычесть из 255 значение цвета

негатив = 255 — цвет

Данные вычисления необходимо выполнить для каждого пиксела. Поэтому мы создали два цикла (один для строк, второй для столбцов).

14 Узоры

Процедура добавления узоров аналогична работе с градиентами: нужно создать узор с помощью метода createPattern().

createPattern(image, type), где атрибут image предоставляет собой ссылку на изображение, а атрибут type может принимть одно из четырех значений: repeat, repeat-x, repeat-y или no-repeat.

```
var canvas, img;
function initiate(){
  var elem = document.getElementById('canvas');
  canvas = elem.getContext('2d');
  img = document.createElement('img');
  img.setAttribute('src', 'http://www.minkbooks.com/content/bricks.jpg');
  img.addEventListener("load", modimage);
}
function modimage(){
  var pattern = canvas.createPattern(img, 'repeat');
  canvas.fillStyle = pattern;
  canvas.fillRect(0, 0, 500, 300);
}
addEventListener("load", initiate);
```

15 Анимация

Для анимирования объектов на холсте не существует ни специальных мето-дов, ни четко определенной последовательности действий. Нарисованные объекты на холсте передвинуть нельзя. Строить анимированное изображение можно одним способом: стирая часть изображения и строя новые фигуры.

Рассмотрим простой пример, в котором будем очищать холст методом clearRect() и снова рисовать на нем фигуры.

```
var canvas;
```

```
function initiate(){
  var elem = document.getElementById('canvas');
  canvas = elem.getContext('2d');
  addEventListener('mousemove', animation);
}
function animation(e){
  canvas.clearRect(0, 0, 700, 300);
  var xmouse = e.clientX;
  var ymouse = e.clientY;
  var xcenter = 220;
  var ycenter = 150;
  var ang = Math.atan2(ymouse - ycenter,xmouse - xcenter);
  var x = xcenter + Math.round(Math.cos(ang) * 10);
  var y = ycenter + Math.round(Math.sin(ang) * 10);
  canvas.beginPath();
  canvas.arc(xcenter, ycenter, 20, 0, Math.PI * 2, false);
  canvas.moveTo(xcenter + 70, 150);
  canvas.arc(xcenter + 50, ycenter, 20, 0, Math.PI * 2, false);
  canvas.stroke();
  x = x+70
  canvas.beginPath();
  canvas.moveTo(x + 10, y);
  canvas.arc(x, y, 10, 0, Math.PI * 2, false);
  canvas.moveTo(x + 60, y);
  canvas.arc(x + 50, y, 10, 0, Math.PI * 2, false);
  canvas.fill();
}
addEventListener("load", initiate);
```

Мы создали рисунок глаз, следящих за указателем мыши. Для перемещения зрачков обновляем позицию соответствующих элементов каждый раз, когда указатель мыши сдвигается. Для этого в функции initiate() используется прослушиватель событий mousemove, который вызывает функцию animation().

Выполнение функции начинается с очистки холста инструкцией clearRect(0, 0, 300, 500). После этого считывается позиция указателя мыши, а в переменных xcenter и ycenter сохраняется местоположение первого глаза.

После инициализации переменных, вычисляем угол наклона невидимого отрезка, соединяющего две эти точки. Для этого используется стандартный метод atan2.

Math.atan2(y, x)

Метод atan2 возвращает числовое значение между -PI и PI, представляющее собой угол Theta для точки (x,y). Это угол, отсчитываемый против часовой стрелки и измеряемый в радианах, между положительным лучом оси X и точкой (x,y). Заметим, что порядок аргументов у этой функции такой, что координата по Y передается первой, а по X - второй.

Методу atan2 передаются отдельно значения x и y, а (atan) - отношение этих двух аргументов.

Затем, на основе угла, по формуле

xcenter + Math.round(Math.sin(ang)x10)

вычисляем точные координаты центра зрачка. Число 10 — это расстояние от центра глаз до центра зрачка

Получив нужные значения, рисуем на холсте глаза. Первый путь объединяет две окружности — получим глаза. Первый метод arc() рисует окружность с координатами xcenter и ycenter. Второй вызов метода arc() создает аналогичную окружность на 50 пикселов правее первой, для чего ему передается инструкция arc(xcenter+50, 150, 20, 0, Math.PI*2, false).

Анимированная часть рисунка определяется вторым путем. Для создания этого пути используются переменные x и y со значениями, вычисленными ранее на основе величины угла. Оба зрачка визуализируются как черные круги с помощью метода fill().

Процесс повторяется при каждом срабатывании события mouseover.

16 Видео

Возможности canvas не ограничиваются рисованием. Данный API позволяет обрабатывать видео на лету.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Video on Canvas</title>
  <style>
    section{
      float: left;
    }
  </style>
  <script>
    var canvas, video;
    function initiate(){
      var elem = document.getElementById('canvas');
      canvas = elem.getContext('2d');
      video = document.getElementById('media');
      canvas.translate(483, 0);
      canvas.scale(-1, 1);
      setInterval(processFrames, 33);
    }
    function processFrames(){
      canvas.drawImage(video, 0, 0);
    }
    addEventListener("load", initiate);
  </script>
</head>
```

```
<body>
  <section>
    <video id="media" width="483" height="272" autoplay>
      <source src="http://www.minkbooks.com/content/trailer2.mp4">
      <source src="http://www.minkbooks.com/content/trailer2.ogg">
    </video>
  </section>
  <section>
    <canvas id="canvas" width="483" height="272"></canvas>
  </section>
</body>
</html>
```

Впрочем, для работы с видео, существует API Video, что уже является темой отдельной публикации

Заключение

Список использованных источников

Приложения