# HTML5 第三天笔记

## Canvas绘图

### 1. 创建路径

图形的基本元素是路径。路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合。一个路径，甚至一个子路径，都是闭合的。使用路径绘制图形需要一些额外的步骤。

1. 首先，需要创建路径起始点。
2. 然后，使用画图命令去画出路径。
3. 之后，把路径封闭。
4. 一旦路径生成，你就能通过描边或填充路径区域来渲染图形。

#### 1）绘制矩形

##### 绘制矩形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用rect(x, y, width, height)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

rect(x, y, width, height)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示矩形的左上角坐标值。 |
| width和height | 表示矩形的宽度和高度。 |

#### 2）绘制圆形

##### 绘制圆形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用arc(x, y, radius, startAngle, endAngle, anticlockwise)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

arc(x, y, radius, startAngle, endAngle, anticlockwise)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示圆形的圆心坐标值。 |
| radius | 表示圆形的半径。 |
| startAngle | 表示绘制圆形的开始点，值为 0。 |
| endAngle | 表示绘制圆形的结束点，值为 Math.PI\*2。 |
| anticlockwise | 表示是以顺时针还是以逆时针方式绘制圆形，Boolean值。（默认值为false，表示顺时针。） |

#### 3）绘制弧形

##### 绘制圆形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用arc(x, y, radius, startAngle, endAngle, anticlockwise)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

arc(x, y, radius, startAngle, endAngle, anticlockwise)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示圆形的圆心坐标值。 |
| radius | 表示圆形的半径。 |
| startAngle | 表示绘制圆形的开始点（取值范围：0 至 Math.PI\*2。）。 |
| endAngle | 表示绘制圆形的结束点（取值范围：0 至 Math.PI\*2。）。 |
| anticlockwise | 表示是以顺时针还是以逆时针方式绘制圆形，Boolean值。（默认值为false，表示顺时针。） |

**值得注意的是：**

如果绘制的是空心弧形的话，在arc()方法调用后：

* 如果使用closePath()方法的话，绘制的图形会自动将终点和起点连接成线。
* 如果不用closePath()方法的话，绘制的图形会呈现开口状。

#### 4）绘制直线

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 通过stroke()方法进行绘制。

#### 5）绘制折线

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 通过stroke()方法进行绘制。

**值得注意的是：**在绘制折线的时候，lineTo()方法既可以绘制折点，也可以绘制终点。

#### 6）绘制多边形

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 调用closePath()方法，闭合当前绘制的路径。
5. 通过fill()或stroke()方法进行绘制。

### 2. 设置线型

所有画布操作都使用相同的线型，即默认线型。实际上线条的宽度、端点都可以根据实际绘图需要进行调整。

#### 1）设置线宽

lineWidth：指定线条粗细，默认值是1.0。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

context.beginPath();

context.moveTo(100, 150);

context.lineTo(450, 50);

context.lineWidth = 15;

context.stroke();

</script>

#### 2）设置端点形状

lineCap：指定线条端点形状。

* butt：默认，向线条的每个末端添加平直的边缘。
* round：向线条的每个末端添加圆形线帽。
* square：向线条的每个末端添加正方向线帽。

**值得注意的是：**round和square会使线条略变微长。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

// butt line cap (top line)

context.beginPath();

context.moveTo(200, canvas.height / 2 - 50);

context.lineTo(canvas.width - 200, canvas.height / 2 - 50);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'butt';

context.stroke();

// round line cap (middle line)

context.beginPath();

context.moveTo(200, canvas.height / 2);

context.lineTo(canvas.width - 200, canvas.height / 2);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'round';

context.stroke();

// square line cap (bottom line)

context.beginPath();

context.moveTo(200, canvas.height / 2 + 50);

context.lineTo(canvas.width - 200, canvas.height / 2 + 50);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'square';

context.stroke();

</script>

#### 3）设置交点形状

lineJoin：指定两条线之间的连接点形状。

* round：创建圆角。
* bevel：创建斜角。
* miter：默认，创建尖角。

miterLimit：与lineJoin一起使用，当lineJoin设置为miter时，可用于确定线条交接点的延伸范围。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

// set line width for all lines

context.lineWidth = 25;

// miter line join (left)

context.beginPath();

context.moveTo(99, 150);

context.lineTo(149, 50);

context.lineTo(199, 150);

context.lineJoin = 'miter';

context.stroke();

// round line join (middle)

context.beginPath();

context.moveTo(239, 150);

context.lineTo(289, 50);

context.lineTo(339, 150);

context.lineJoin = 'round';

context.stroke();

// bevel line join (right)

context.beginPath();

context.moveTo(379, 150);

context.lineTo(429, 50);

context.lineTo(479, 150);

context.lineJoin = 'bevel';

context.stroke();

</script>

## 处理图像

在HTML5中，不仅可以使用Canvas API来绘制图形，还可以读取磁盘或网络中的图像文件，然后使用Canvas API将该图像绘制在画布中。

### 1. 绘制图像

#### 1）加载图像

* 使用相同页面中的图片。
* 使用相同页面中的其他Canvas元素。
* 可以脚本通过Image( )构造函数创建图像。

#### 2）绘制图像

使用 drawImage() 方法绘制图像，该方法具体用法如下：

drawImage(img, x, y)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要绘制的图像。 |
| x和y | 绘制图像的坐标值。 |

drawImage(img, x, y, width, height)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要绘制的图像。 |
| x和y | 绘制图像的坐标值。 |
| width和height | 设置绘制图像的宽度和高度。 |

<canvas id="myCanvas" width="578" height="400"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

var imageObj = new Image();

imageObj.src = 'darth-vader.jpg';

imageObj.onload = function() {

context.drawImage(imageObj, 69, 50);

};

</script>

**值得注意的是：**若调用 drawImage 时，图片没装载完，那什么都不会发生（在一些旧的浏览器中可能会抛出异常）。因此你应该用load时间来保证不会在加载完毕之前使用这个图片：

var img = new Image(); // 创建img元素

img.src = 'myImage.png'; // 设置图片源地址

img.onload = function(){

// 执行drawImage语句

}

### 2. 平铺图像

所谓图像平铺就是用按一定比例缩小后的图像将画布填满。

1. 加载图像。
2. 通过 createPattern(img, type) 方法设置平铺方式。
3. 将平铺对象赋值给filleStyle或strokeStyle属性。
4. 将平铺的图像进行绘制。

createPattern(img, type)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要平铺的图像。 |
| type | 平铺方式。（no-repeat：不平铺；repeat-x：水平方向平铺；repeat-y：垂直方向平铺；repeat：全方向平铺） |

该方法返回平铺对象。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

var imageObj = new Image();

imageObj.src = 'wood-pattern.png';

imageObj.onload = function() {

var pattern = context.createPattern(imageObj, 'repeat');

context.rect(0, 0, canvas.width, canvas.height);

context.fillStyle = pattern;

context.fill();

};

</script>

**值得注意的是：**若调用 createPattern 时，图片没装载完，那什么都不会发生（在一些旧的浏览器中可能会抛出异常）。因此你应该用load时间来保证不会在加载完毕之前使用这个图片：

var img = new Image(); // 创建img元素

img.src = 'myImage.png'; // 设置图片源地址

img.onload = function(){

// 执行createPattern语句

}

### 3. 切割图像

1. 调用beginPath()方法，创建新建一条路径。
2. 使用rect()或arc()方法
3. 通过clip()方法进行切割。

<canvas id="canvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('canvas');

var context = elem.getContext('2d');

var image=new Image();

image.src="img/flower.jpg";

image.onload=function(){

context.drawImage(image,0,0,280,190);

}

context.beginPath();

context.arc(140,95,60,0,Math.PI\*2,true);

context.closePath();

context.clip();

</script>

### 4. 画布方法

#### 1）状态方法

* save()，保存当前画布属性、状态。
* restore()，恢复画布属性、状态。

<canvas id="canvas" width="578" height="200"></canvas>

<script>

var ctx = document.getElementById('canvas').getContext('2d');

ctx.fillRect(0,0,150,150); // Draw a rectangle with default settings

ctx.save(); // Save the default state

ctx.fillStyle = '#09F' // Make changes to the settings

ctx.fillRect(15,15,120,120); // Draw a rectangle with new settings

ctx.save(); // Save the current state

ctx.fillStyle = '#FFF' // Make changes to the settings

ctx.globalAlpha = 0.5;

ctx.fillRect(30,30,90,90); // Draw a rectangle with new settings

ctx.restore(); // Restore previous state

ctx.fillRect(45,45,60,60); // Draw a rectangle with restored settings

ctx.restore(); // Restore original state

ctx.fillRect(60,60,30,30); // Draw a rectangle with restored settings

</script>

#### 2）转换方法

translate(x, y)：用来移动 canvas 和它的原点到一个不同的位置。

* x 是左右偏移量。
* y 是上下偏移量。

scale(x, y)：用它来增减图形在 canvas 中的像素数目，对形状，位图进行缩小或者放大。

* x,y 分别是横轴和纵轴的缩放因子，它们都必须是正值。
* 值分类：
  + 值比 1.0 小表示缩小。
  + 比 1.0 大则表示放大。
  + 值为 1.0 时什么效果都没有。

rotate(angle)：用于以原点为中心旋转 canvas。

* 旋转的角度(angle)，它是顺时针方向的，以弧度为单位的值。

## Chart.js库

### 1. 基础内容

Chart.js是一个简单、面向对象、为设计者和开发者准备的图表绘制工具库。

官方网址：<http://www.chartjs.org/>

Chart.js的特点：

* 基于HTML 5：Chart.js基于HTML5 canvas技术，支持所有现代浏览器，并且针对IE7/8提供了降级替代方案。
* 简单、灵活：Chart.js不依赖任何外部工具库，轻量级（压缩之后仅有4.5k），并且提供了加载外部参数的方法。

### 2. 使用Chart.js

#### 1）使用Chart.js的步骤

* 在HTML页面中引入Chart.js文件。

<script src="Chart.js"></script>

* 创建<canvas>元素：用于显示Chart图表的容器。

<canvas id="myChart" width="400" height="400"></canvas>

* 获取Canvas对象。

document.getElementById("myChart").getContext("2d");`

* 创建Chart图表对象。

new Chart(ctx);

* 通过Chart图表对象进行绘制。

chart.PolarArea(data);

#### 2）Chart.js全局配置

Chart.js 全局配置是在chart.js 第一个正式版本中引入。Chart.js 全局配置用于改变所有图表的类型，避免了需要在每一个图表中单独进行设置。当然，Chart.js 全局配置也可以专门为某一个特定的图表进行配置。

语法如下：

Chart.defaults.global.参数名 = 参数值;

例如如下方式设定：

Chart.defaults.global.responsive = true;

#### 3）绘制图表

##### a. 曲线图

曲线图就是将多个数据点绘制在一条线上，通常被用于展示趋势的数据或两组数据之间的对比。

new Chart(ctx).Line(data, options)

* data：用于设置曲线上的数据、样式及名称。
* options：选项，用于配置曲线图。

##### b. 柱状图

柱状图就是使用柱状方式显示数据的一种方式，通常被用于展示趋势的数据或多组数据之间的比较。

new Chart(ctx).Bar(data, options)

* data：用于设置柱状图上的数据、样式及名称。
* options：选项，用于配置柱状图。

##### c. 饼状图

饼状图可能是所有图表中最为常用的一种，就是将一个圆划分成若干个部分，每个弧形展示每个数据的比例值。通常被用于展示多组数据之间的比例。

new Chart(ctx).Pie(data,options)

* data：用于设置饼图的数据、样式及名称。
* options：选项，用于配置饼图。

##### d. 雷达图

雷达图就是一种展示多个数据点以及它们之间变化的方式，通常被用于比较点的两个或多个不同的数据集。

new Chart(ctx).Radar(data, options)

* data：用于设置雷达图的数据、样式及名称。
* options：选项，用于配置雷达图。

##### e. 环形图

环形图类似于饼状图，但环形图是一个空心的环形形状，通常被用于展示多组数据之间关系的比例。

new Chart(ctx).Doughnut(data,options)

* data：用于设置环形图的数据、样式及名称。
* options：选项，用于配置环形图。

##### f. 极地区域图

极地区域图类似于饼状图，但每一个扇形的角度和半径取决于不同的值，通常被用于需要展示类似于饼状图的比较数据的基础上，还需要展示范围值的比较。

new Chart(ctx).PolarArea(data, options)

* data：用于设置极地区域图的数据、样式及名称。
* options：选项，用于配置极地区域图。

<canvas id="canvas" width="300px" height="300px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext("2d");

// 初始化饼状图的数据 - 以数组格式

var data = [

{

// 当前数据占饼状图的比例

value: 30,

// 当前数据块的颜色

color:"blue",

// 当鼠标悬停在当前数据块时的颜色

highlight: "yellow",

// 当前数据块的提示信息

label: "Blue"

},

{

value: 50,

color: "#46BFBD",

highlight: "#5AD3D1",

label: "Green"

},

{

value: 100,

color: "#FDB45C",

highlight: "#FFC870",

label: "Yellow"

},

{

value: 40,

color: "#949FB1",

highlight: "#A8B3C5",

label: "Grey"

}

];

//Chart.defaults.global.responsive = true;

// 通过画布对象,创建Chart对象

var chart = new Chart(context);

// 利用Chart对象调用图表方法

chart.Pie(data);

</script>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>