第10章 HTML5 Canvas

课程提要

- canvas概述
- 常用方法

10.1 canvas概述

Canvas是HTML5新增的组件，它就像一块幕布，可以用JavaScript在上面绘制各种图表、动画等。没有Canvas的年代，绘图只能借助Flash插件实现，页面不得不用JavaScript和Flash进行交互。有了Canvas，我们就再也不需要Flash了，直接使用JavaScript完成绘制。

10.1.1 基本用法

10.1.1.1 canvas元素

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

canvas看起来和img 元素很相像，唯一的不同就是它并没有 src 和 alt 属性。实际上，<canvas> 标签只有两个属性—— width和height。这些都是可选的，并且同样利用 DOM properties来设置。当没有设置宽度和高度的时候，canvas会初始化宽度为300像素和高度为150像素。该元素可以使用CSS来定义大小，但在绘制时图像会伸缩以适应它的框架尺寸：如果CSS的尺寸与初始画布的比例不一致，它会出现扭曲。

注意： 如果你绘制出来的图像是扭曲的, 尝试用width和height属性为<canvas>明确规定宽高，而不是使用CSS。

10.1.1.2 渲染上下文

<canvas> 元素创造了一个固定大小的画布，它公开了一个或多个渲染上下文，其可以用来绘制和处理要展示的内容。canvas起初是空白的。为了展示，首先脚本需要找到渲染上下文，然后在它的上面绘制。getContext()的方法，这个方法是用来获得渲染上下文和它的绘画功能。getContext()只有一个参数，上下文的格式。

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

10.1.1.3 兼容处理

Internet Explorer 9+, Firefox, Opera, Chrome 以及 Safari 支持canvas标签。Internet Explorer 8 以及更早的版本不支持canvas标签。

由于浏览器对HTML5标准支持不一致，所以，通常在内部添加一些说明性HTML代码，如果浏览器支持Canvas，它将忽略内部的HTML，如果浏览器不支持Canvas，它将显示内部的HTML：

```
<canvas id="canvas" width="300" height="200">您的浏览器不支持canvas</canvas>
```

同样也可以用JS来处理兼容问题：

```
var canvas = document.getElementById('canvas');
if (canvas.getContext) {
    console.log('你的浏览器支持Canvas!');
} else {
    console.log('你的浏览器不支持Canvas!');
}
```

10.1.2 canvas和svg的比较

SVG：

SVG 是一种使用 XML描述 2D 图形的语言。

SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。您可以为某个元素附加 JavaScript 事件处理器。

在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

Canvas：

Canvas 通过 JavaScript 来绘制 2D 图形。

Canvas 是逐像素进行渲染的。

在 canvas 中，一旦图形被绘制完成，它就不会继续得到浏览器的关注。如果其位置发生变化，那么整个场景也需要重新绘制，包括任何或许已被图形覆盖的对象。

两者区别：

Canvas

1. 依赖分辨率
2. 不支持事件处理器
3. 弱的文本渲染能力
4. 能够以 .png 或 .jpg 格式保存结果图像
5. 最适合图像密集型的游戏，其中的许多对象会被频繁重绘

SVG

1. 不依赖分辨率
2. 支持事件处理器
3. 最适合带有大型渲染区域的应用程序（比如谷歌地图）
4. 复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）
5. 不适合游戏应用

10.2 常用方法

10.2.1. 绘制路径

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，你需要创建路径起始点。
2. 然后你使用画图命令去画出路径。
3. 之后你把路径封闭。
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。

以下是绘制路径需要用到的函数：

方法	描述
beginPath()	起始一条路径，或重置当前路径
closePath()	创建从当前点回到起始点的路径

生成路径的第一步叫做beginPath()。本质上，路径是由很多子路径构成，这些子路径都是在一个列表中，所有的子路径（线、弧形、等等）构成图形。而每次这个方法调用之后，列表清空重置，然后我们就可以重新绘制新的图形。

注意：当前路径为空，即调用beginPath()之后，或者canvas刚建的时候，第一条路径构造命令通常被视为是moveTo（），无论实际上是什么。出于这个原因，你几乎总是要在设置路径之后专门指定你的起始位置。

第二步就是调用函数指定绘制路径，本文稍后我们就能看到了。

第三，就是闭合路径closePath(),不是必需的。这个方法会通过绘制一条从当前点到开始点的直线来闭合图形。如果图形是已经闭合了的，即当前点为开始点，该函数什么也不做。

注意：当你调用fill()函数时，所有没有闭合的形状都会自动闭合，所以你不需要调用closePath()函数。但是调用stroke()时不会自动闭合。

10.2.1.1 移动笔触

一个非常有用的函数，而这个函数实际上并不能画出任何东西，也是上面所描述的路径列表的一部分，这个函数就是moveTo()。或者你可以想象一下在纸上作业，一支钢笔或者铅笔的笔尖从一个点到另一个点的移动过程。

语法：

```
moveTo(x, y)//将笔触移动到指定的坐标x以及y上。
```

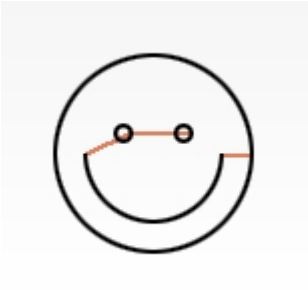
当canvas初始化或者beginPath()调用后，你通常会使用moveTo()函数设置起点。我们也能够使用moveTo()绘制一些不连续的路径。看一下下面的笑脸例子。我将用到moveTo()方法（红线处）的地方标记了。

示例：

```
functiodraw() {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext){
        var ctx = canvas.getContext('2d');
        ctx.beginPath();
        ctx.arc(75,75,50,0,Math.PI*2,true); // 绘制
        ctx.moveTo(110,75);
```

```
        ctx.arc(75,75,35,0,Math.PI,false);   // 口(顺时针)
        ctx.moveTo(65,65);
        ctx.arc(60,65,5,0,Math.PI*2,true);  // 左眼
        ctx.moveTo(95,65);
        ctx.arc(90,65,5,0,Math.PI*2,true);  // 右眼
        ctx.stroke();
    }
}
```

结果如下：



10.2.1.2 线

绘制直线，需要用到的方法lineTo()。

语法：

```
lineTo(x, y) // 绘制一条从当前位置到指定x以及y位置的直线。
```

该方法有两个参数：x以及y ，代表坐标系中直线结束的点。开始点和之前的绘制路径有关，之前路径的结束点就是接下来的开始点。开始点也可以通过moveTo()函数改变。

示例：

```
functiodraw(){
    var canvas = document.getElementById('canvas');
    if (canvas.getContext){
        var ctx = canvas.getContext('2d');
        // 填充三角形
        ctx.beginPath();
        ctx.moveTo(25,25);
        ctx.lineTo(105,25);
        ctx.lineTo(25,105);
        ctx.fill();
        // 描边三角形
        ctx.beginPath();
        ctx.moveTo(125,125);
        ctx.lineTo(125,45);
        ctx.lineTo(45,125);
        ctx.closePath();
        ctx.stroke();
    }
}
```

这里从调用beginPath()函数准备绘制一个新的形状路径开始。然后使用moveTo()函数移动到目标位置上。然后下面，两条线段绘制后构成三角形的两条边。

你会注意到填充与描边三角形步骤有所不同。正如上面所提到的，因为路径使用填充（fill）时，路径自动闭合，使用描边（stroke）则不会闭合路径。如果没有添加闭合路径closePath()到描述三角形函数中，则只绘制了两条线段，并不是一个完整的三角形。

10.2.1.3 圆弧

绘制圆弧或者圆，我们使用arc()方法。当然可以使用arcTo()，不过这个的实现并不是那么的可靠，所以我们这里不作介绍。

语法：

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

注释：

该方法有六个参数：x,y为绘制圆弧所在圆上的圆心坐标。radius为半径。startAngle以及endAngle参数用弧度定义了开始以及结束的弧度。这些都是以x轴为基准。参数anticlockwise为一个布尔值。为true时，是逆时针方向，否则顺时针方向。

示例：

```
//绘制了12个不同的角度以及填充的圆弧
for (var i = 0; i < 4; i++) {
    for (var j = 0; j < 3; j++) {
        ctx.beginPath();
        var x = 25 + j * 50; // x 坐标值
        var y = 25 + i * 50; // y 坐标值
        var radius = 20; // 圆弧半径
        var startAngle = 0; // 开始点
        var endAngle = Math.PI + (Math.PI * j) / 2; // 结束点
        var anticlockwise = i % 2 == 0 ? false: true; // 顺时针或逆时针
        ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);
        if (i > 1) {
            ctx.fill();
        } else {
            ctx.stroke();
        }
    }
}
```

10.2.1.4 贝塞尔曲线

贝塞尔曲线一般用来绘制复杂有规律的图形。它包括二次贝塞尔曲线和三次贝塞尔曲线。

语法：

二次贝塞尔曲线

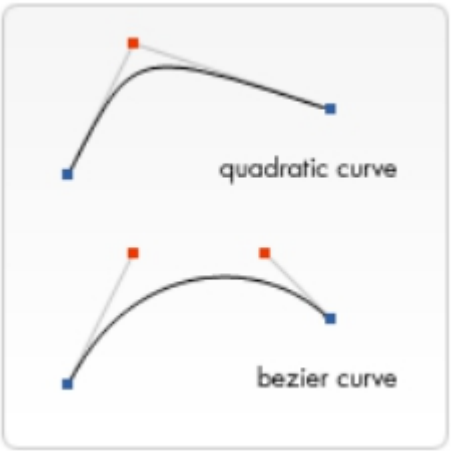
```
quadraticCurveTo(cp1x, cp1y, x, y)//绘制二次贝塞尔曲线，cp1x,cp1y为一个控制点，x,y为结束点。
```

三次贝塞尔曲线

```
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

绘制三次贝塞尔曲线，cp1x,cp1y为控制点一，cp2x,cp2y为控制点二，x,y为结束点。

注释：



上边的图能够很好的描述两者的关系，二次贝塞尔曲线有一个开始点（蓝色）、一个结束点（蓝色）以及一个控制点（红色），而三次贝塞尔曲线有两个控制点。

参数x、y在这两个方法中都是结束点坐标。cp1x,cp1y为坐标中的第一个控制点，cp2x,cp2y为坐标中的第二个控制点。

示例1：

```
var ctx = canvas.getContext('2d');
// 二次贝塞尔曲线
ctx.beginPath();
ctx.moveTo(75, 25);
ctx.quadraticCurveTo(25, 25, 25, 62.5);
ctx.quadraticCurveTo(25, 100, 50, 100);
ctx.quadraticCurveTo(50, 120, 30, 125);
ctx.quadraticCurveTo(60, 120, 65, 100);
ctx.quadraticCurveTo(125, 100, 125, 62.5);
ctx.quadraticCurveTo(125, 25, 75, 25);
ctx.stroke();
```

示例2：

```
var ctx = canvas.getContext('2d');
//三次贝塞尔曲线
ctx.beginPath();
ctx.moveTo(75, 40);
ctx.bezierCurveTo(75, 37, 70, 25, 50, 25);
ctx.bezierCurveTo(20, 25, 20, 62.5, 20, 62.5);
ctx.bezierCurveTo(20, 80, 40, 102, 75, 120);
ctx.bezierCurveTo(110, 102, 130, 80, 130, 62.5);
ctx.bezierCurveTo(130, 62.5, 130, 25, 100, 25);
ctx.bezierCurveTo(85, 25, 75, 37, 75, 40);
ctx.fill();
```

10.2.2 绘制矩形

不同于SVG，HTML中的元素canvas只支持一种原生的图形绘制：矩形。所有其他的图形的绘制都至少需要生成一条路径。

canvas提供了三种方法绘制矩形：

```
fillRect(x, y, width, height)//绘制一个填充的矩形
strokeRect(x, y, width, height)//绘制一个矩形的边框
clearRect(x, y, width, height)//清除指定矩形区域，让清除部分完全透明。
```

注释：

绘制矩形的方法之中每一个都包含了相同的参数。x与y指定了在canvas画布上所绘制的矩形的左上角（相对于原点）的坐标。width和height设置矩形的尺寸。

示例：

```
var canvas = document.getElementById('canvas');
if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
    ctx.fillRect(25,25,100,100);
    ctx.clearRect(45,45,60,60);
    ctx.strokeRect(50,50,50,50);
}
```

fillRect()函数绘制了一个边长为100px的黑色正方形。clearRect()函数从正方形的中心开始擦除了一个60*60px的正方形，接着strokeRect()在清除区域内生成一个50*50的正方形边框。

10.2.3 绘制文本

10.2.3.1 基本方法

canvas 提供了两种方法来渲染文本:

```
fillText(text, x, y [, maxWidth])
```

在指定的(x,y)位置填充指定的文本，绘制的最大宽度是可选的。

```
strokeText(text, x, y [, maxWidth])
```

在指定的(x,y)位置绘制文本边框，绘制的最大宽度是可选的。

示例1：

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.fillText("Hello world", 10, 50);
}
```

示例2：

```
functiodraw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.strokeText("Hello world", 10, 50);
}
```

10.2.3.2 设置有样式的文本

可以通过设置属性来改变canvas显示文本的方式。

属性	解释
font = value	当前我们用来绘制文本的样式. 这个字符串使用和 CSS font 属性相同的语法. 默认的字体是 10px sans-serif。
textAlig= value	文本对齐选项. 可选的值包括：start, end, left, right or center. 默认值是 start
textBaseline = value	基线对齐选项. 可选的值包括：top, hanging, middle, alphabetic, ideographic, bottom。默认值是 alphabetic。
directio= value	文本方向。可能的值包括：ltr, rtl, inherit。默认值是 inherit。

示例：

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.font = "40px serif";
    ctx.textBaseline = "hanging";
    ctx.strokeText("Hello world", 0, 100);
}
```

10.2.4 绘制图像

canvas更有意思的一项特性就是图像操作能力。可以用于动态的图像合成或者作为图形的背景，以及游戏界面（Sprites）等等。浏览器支持的任意格式的外部图片都可以使用，比如PNG、GIF或者JPEG。 你甚至可以将同一个页面中其他canvas元素生成的图片作为图片源。

引入图像到canvas里需要以下两步基本操作：

1. 获得一个指向HTMLImageElement的对象或者另一个canvas元素的引用作为源，也可以通过提供一个URL的方式来使用图片；
2. 使用drawImage()函数将图片绘制到画布上

10.2.4.1 由零开始创建图像

可以用脚本创建一个新的 HTMLImageElement对象。要实现这个方法，我们可以使用很方便的Image()构造函数。

```
var img = new Image();   // 创建一个<img>元素
img.src = 'myImage.png'; // 设置图片源地址
```

当脚本执行后，图片开始装载。

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事件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.onload = function(){
  // 执行drawImage语句
}
img.src = 'myImage.png'; // 设置图片源地址
```

10.2.4.2 绘制图像

一旦获得了源图对象，我们就可以使用 drawImage 方法将它渲染到 canvas 里。

（1）最基础的一种：

语法：

```
drawImage(image, x, y)
```

其中 image 是 image 或者 canvas 对象，x 和 y 是其在目标 canvas 里的起始坐标。

示例1：

```
var ctx = document.getElementById('canvas').getContext('2d');
var img = new Image();
img.onload = function() {
    ctx.drawImage(img, 0, 0);
}
img.src = 'images/ Canvas_backdrop.png';
```

（2）缩放图片

drawImage 方法的又一变种是增加了两个用于控制图像在 canvas 中缩放的参数。

语法：

```
drawImage(image, x, y, width, height)
```

这个方法多了2个参数：width 和 height，这两个参数用来控制 当向canvas画入时应该缩放的大小。

示例2：

```
var ctx = document.getElementById('canvas').getContext('2d');
var img = new Image();
img.onload = function(){
    for (var i=0;i<4;i++){
        for (var j=0;j<3;j++){
            ctx.drawImage(img,j*100,i*50,100,50);
        }
    }
}
img.src = 'images/flower.png';
```

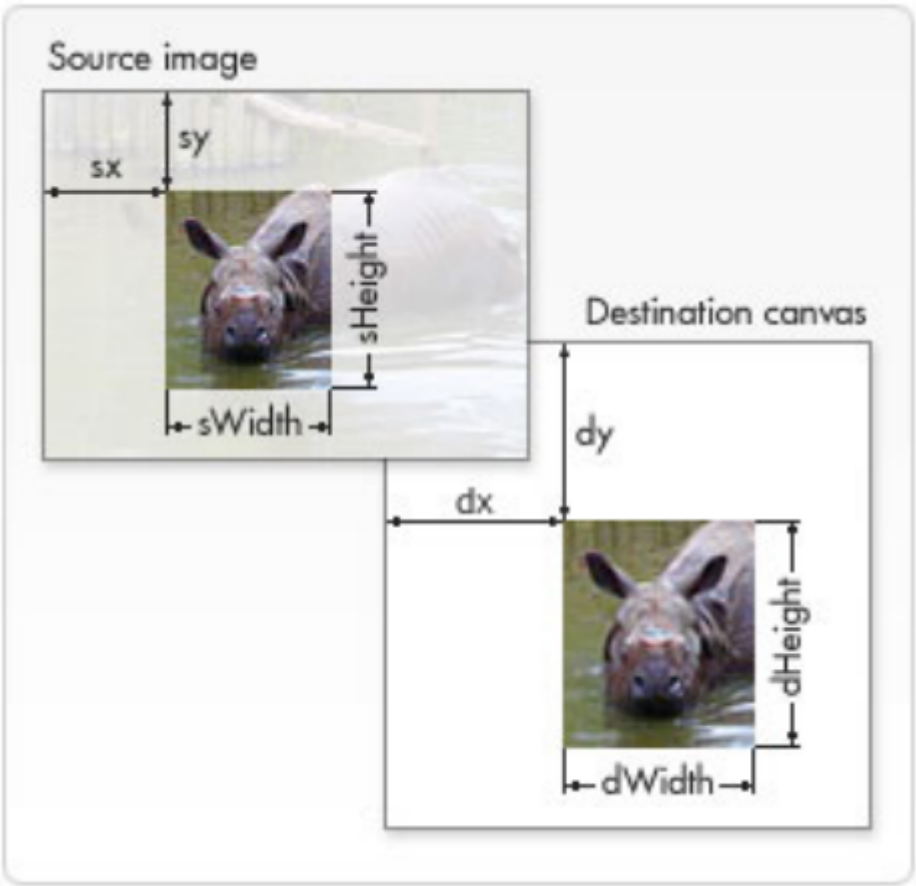
（3）切片

drawImage 方法的第三个也是最后一个变种有8个新参数，用于控制做切片显示的。

语法：

```
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
```

第一个参数和其它的是相同的，都是一个图像或者另一个 canvas 的引用。其它8个参数参照下边的图解，前4个是定义图像源的切片位置和大小，后4个则是定义切片的目标显示位置和大小。



示例3：

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,40,88,180,150,21,20,100,104);
    }
    img.src = 'images/rhino.jpg';
}
```

10.2.5 其他

canvas其他常用方法：

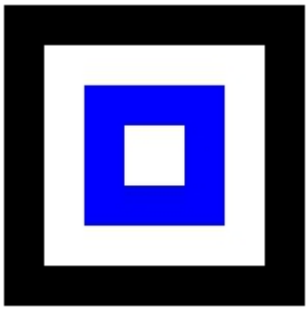
方法	描述
save()	保存当前环境的状态
restore()	返回之前保存过的路径状态和属性

对canvas中特定元素的操作实际上是对整个画布进行了操作，所以如果不对canvas进行save以及restore，那么每一次绘图都会在上一次的基础上进行操作，最后导致错位。

示例：

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext('2d');
ctx.fillRect(10,10,150,150);
ctx.save();
ctx.fillStyle="white";
ctx.fillRect(30,30,110,110);
ctx.save();
ctx.fillStyle="blue";
ctx.fillRect(50,50,70,70);
ctx.restore();//回到上一个状态,即 ctx.fillStyle="white";
ctx.save();
ctx.fillRect(70,70,30,30);//所以此处没有设定fillStyle的时候颜色为white，注意哦！如果在白色矩形后面也restore一下刚此处的fillStyle就为黑色了
ctx.restore();
```

结果如下：



第一步是用默认设置画一个大四方形，然后保存一下状态。改变填充颜色画第二个小一点的白色四方形，然后再保存一下状态。再次改变填充颜色绘制更小一点的蓝色四方形。然后我们调用了restore方法将设置回到前一个save状态下的fillStyle = "white"，即在不设定颜色值的情况下再绘制最小的矩形时其添充色为白色。

一旦我们调用 restore，状态堆中最后的状态会弹出，并恢复所有设置。如果不是之前用 save 保存了状态，那么我们就需要手动改变设置来回到前一个状态，这个对于两三个属性的时候还是适用的，一旦多了，我们的代码将会猛涨。 简而言之restore方法就可以理解成将其对应的当前save状态下的设置全部恢复为前一个状态。

10.3 课程总结

- canvas概述
- 常用方法

10.4 实训

1、 使用canvas画一个五角星。如下图：

2、 使用canvas实现验证码图片，需求如下：

1）随机生成干扰线

2）显示随机4位数（数字或者字母）

3）当点击验证码图片时，干扰线和4位数随机变化

效果图如下：