第 7 章

使用 HTML 5 绘制图形

HTML 5 有很多新特性，其中一个最值得提及的特性就是 HTML canvas，它可以对 2D 图形或位图进行动态、脚本的渲染。使用 canvas 可以绘制一个矩形区域，然后使用 JavaScript 可以控制其每像素，例如可以用它来画图、合成图像，或制作简单的动画。本章介绍如何使用 HTML 5 绘制图形。

7.1 添加 canvas 的步骤

canvas 标签是一个矩形区域，它包含 width 和 height 两个属性，分别表示矩形区域的宽度和高度。这两个属性都是可选的，并且都可以通过 CSS 来定义，其默认值是 300px 和 150px。

canvas 在网页中的常用形式如下：

```
<canvas id="myCanvas" width="300" height="200"
  style="border:1px solid #c3c3c3;">
    Your browser does not support the canvas element.
</canvas>
```

在上述示例代码中，id 表示画布对象名称，width 和 height 分别表示宽度和高度。最初的画布是不可见的，此处为了观察这个矩形区域，使用了 CSS 样式，即 style 标记。style 表示画布的样式。如果浏览器不支持画布标记，会显示画布中间的提示信息。

画布 canvas 本身不具有绘制图形的功能，它只是一个容器。如果读者对于 Java 语言非常了解，就会发现 HTML 5 的画布和 Java 中的 Panel 面板非常相似，都可以在容器中绘制图形。既然 canvas 画布元素放好了，就可以使用脚本语言 JavaScript 在网页上绘制图形了。

使用 canvas 结合 JavaScript 绘制图形，一般情况下需要下面几个步骤。

step 01 JavaScript 使用 id 来寻找 canvas 元素，即获取当前画布对象：

```
var c = document.getElementById("myCanvas");
```

step 02 创建 context 对象：

```
var cxt = c.getContext("2d");
```

getContext 方法返回一个指定 contextId 的上下文对象，如果指定的 id 不被支持，则返回 null，当前唯一被强制必须支持的是“2d”，也许在将来会有“3d”。注意，指定的 id 是大小写敏感的。对象 cxt 建立之后，就可以拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

step 03 绘制图形：

```
cxt.fillStyle = "#FF0000";
cxt.fillRect(0,0,150,75);
```

fillStyle 方法将其染成红色，fillRect 方法规定了形状、位置和尺寸。这两行代码可以绘制一个红色的矩形。

7.2 绘制基本形状

画布 canvas 结合 JavaScript 可以绘制简单的矩形，还可以绘制一些其他的常见图形，如直线、圆等。

7.2.1　绘制矩形

用 canvas 和 JavaScript 绘制矩形时，涉及一个或多个方法，这些方法如表 7-1 所示。

表 7-1　绘制矩形的方法

方　法	功　能
fillRect	绘制一个矩形，这个矩形区域没有边框，只有填充色。这个方法有 4 个参数，前两个表示左上角的坐标位置，第 3 个参数为长度，第 4 个参数为高度
strokeRect	绘制一个带边框的矩形。该方法的 4 个参数的解释同上
clearRect	清除一个矩形区域，被清除的区域没有任何线条。该方法的 4 个参数的解释同上

【例 7.1】绘制矩形(示例文件 ch07\7.1.html)。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="300" height="200"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var cxt = c.getContext("2d");
cxt.fillStyle = "rgb(0,0,200)";
cxt.fillRect(10,20,100,100);
</script>
</body>
</html>
```

在上面的代码中，定义了一个画布对象，其 id 名称为 myCanvas，高度和宽度都为 500 像素，并定义了画布边框的显示样式。代码中首先获取画布对象，然后使用 getContext 获取当前 2d 的上下文对象，并使用 fillRect 绘制一个矩形。其中涉及一个 fillStyle 属性，fillStyle 用于设定填充的颜色、透明度等，如果设置为“rgb(200,0,0)”，则表示一个不透明颜色；如果设置为“rgba(0,0,200,0.5)”，则表示一个透明度为 50%的颜色。

在 IE 11.0 中浏览，效果如图 7-1 所示，可以看到网页中在一个蓝色边框内显示了一个蓝色矩形。

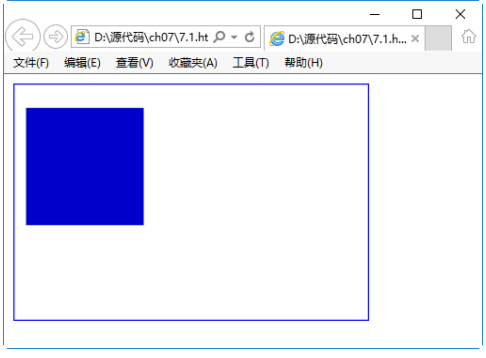

图 7-1　绘制矩形

7.2.2　绘制圆形

在画布中绘制圆形，可能要涉及下面几个方法，如表 7-2 所示。

表 7-2　绘制圆形的方法

方　法	功　能
beginPath()	开始绘制路径
arc(x,y,radius,startAngle, endAngle,anticlockwise)	x 和 y 定义的是圆的原点；radius 是圆的半径；startAngle 和 endAngle 是弧度，不是度数；anticlockwise 用来定义画圆的方向，值是 true 或 false
closePath()	结束路径的绘制
fill()	进行填充
stroke()	该方法设置边框

路径是绘制自定义图形的好方法。在 canvas 中，通过 beginPath()方法开始绘制路径，然后就可以绘制直线、曲线等，绘制完成后，调用 fill()和 stroke()完成填充和边框设置，通过 closePath()方法结束路径的绘制。

【例 7.2】绘制圆形(示例文件 ch07\7.2.html)。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="200"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var cxt = c.getContext("2d");
cxt.fillStyle = "#FFaa00";
cxt.beginPath();
cxt.arc(70,18,15,0,Math.PI*2,true);
cxt.closePath();
cxt.fill();
</script>
</body>
</html>
```

在上面的 JavaScript 代码中，使用 beginPath 方法开启一个路径，然后绘制一个圆形，最后关闭这个路径并填充。在 IE 11.0 中浏览，效果如图 7-2 所示。

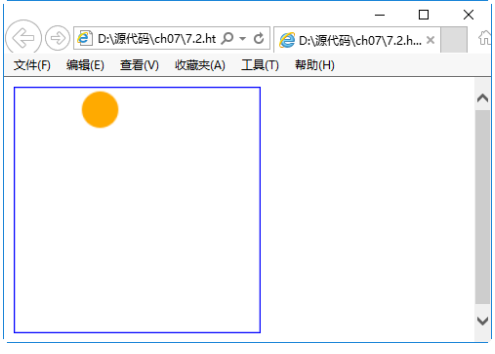

图 7-2　绘制圆形

7.2.3　使用 moveTo 与 lineTo 绘制直线

绘制直线常用的方法是 moveTo 和 lineTo，其含义如表 7-3 所示。

表 7-3　绘制直线的方法

方法或属性	功　能
moveTo(x,y)	不绘制，只是将当前位置移动到新目标位置(x,y)，并作为线条的开始点
lineTo(x,y)	绘制线条到指定的目标位置(x,y)，并且在两个坐标之间画一条直线。不管调用它们哪一个，都不会真正画出图形，因为还没有调用 stroke 和 fill 函数。当前，只是在定义路径的位置，以便后面绘制时使用
strokeStyle	指定线条的颜色
lineWidth	设置线条的粗细

【例 7.3】绘制直线(示例文件 ch07\7.3.html)。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="200"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var cxt = c.getContext("2d");
cxt.beginPath();
cxt.strokeStyle = "rgb(0,182,0)";
cxt.moveTo(10,10);
cxt.lineTo(150,50);
cxt.lineTo(10,50);
cxt.lineWidth = 14;
cxt.stroke();
cxt.closePath();
</script>
</body>
</html>
```

在上面的代码中，使用 moveTo 方法定义一个位置坐标为(10,10)，然后以此位置坐标为起点，绘制两条不同的直线，并用 lineWidth 设置了直线的宽度，用 strokeStyle 设置了直线的颜色，用 lineTo 设置了两条不同直线的结束位置。

在 IE 11.0 中浏览，效果如图 7-3 所示，可以看到，网页中绘制了两条直线，这两条直线在某一点交叉。

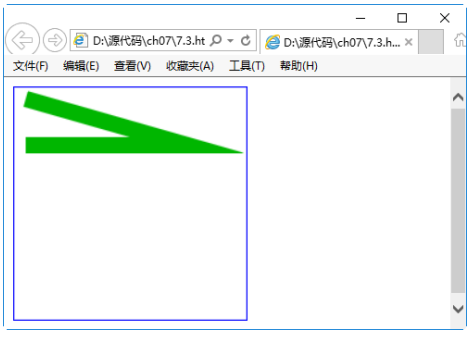

图 7-3　绘制直线

7.2.4 使用 bezierCurveTo 绘制贝塞尔曲线

在数学的数值分析领域中，贝塞尔(Bézier)曲线是电脑图形学中相当重要的参数曲线。更高维度的广泛化贝塞尔曲线就称作贝塞尔曲面，其中贝塞尔三角是一种特殊的实例。

bezierCurveTo()表示为一个画布的当前子路径添加一条三次贝塞尔曲线。这条曲线的开始点是画布的当前点，而结束点是(x, y)。两条贝塞尔曲线的控制点(cpX1, cpY1)和(cpX2, cpY2)定义了曲线的形状。当这个方法返回的时候，当前的位置为(x, y)。

方法 bezierCurveTo 的具体格式如下：

```
bezierCurveTo(cpX1, cpY1, cpX2, cpY2, x, y)
```

其参数的含义如表 7-4 所示。

表 7-4 绘制贝塞尔曲线的参数

参 数	描 述
cpX1, cpY1	与曲线的开始点(当前位置)相关联的控制点的坐标
cpX2, cpY2	与曲线的结束点相关联的控制点的坐标
x, y	曲线的结束点的坐标

【例 7.4】绘制贝塞尔曲线(示例文件 ch07\7.4.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>贝塞尔曲线</title>
<script>
function draw(id)
{
    var canvas = document.getElementById(id);
    if(canvas==null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#eeeeff";
    context.fillRect(0,0,400,300);
    var n = 0;
    var dx = 150;
    var dy = 150;
    var s = 100;
    context.beginPath();
    context.globalCompositeOperation = 'and';
    context.fillStyle = 'rgb(100,255,100)';
    context.strokeStyle = 'rgb(0,0,100)';
    var x = Math.sin(0);
    var y = Math.cos(0);
    var dig = Math.PI/15*11;
    for(var i=0; i<30; i++)
    {
        var x = Math.sin(i*dig);
        var y = Math.cos(i*dig);
```

```
        context.bezierCurveTo(
          dx+x*s,dy+y*s-100,dx+x*s+100,dy+y*s,dx+x*s,dy+y*s);
    }
    context.closePath();
    context.fill();
    context.stroke();
}
</script>
</head>
<body onload="draw('canvas');">
<h1>绘制元素</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在上面的 draw 函数代码中，首先使用 fillRect(0,0,400,300)语句绘制一个矩形，其大小与画布相同，填充颜色为浅青色。然后定义了几个变量，用于设定曲线的坐标位置，在 for 循环中使用 bezierCurveTo 绘制贝塞尔曲线。在 IE 11.0 中浏览，效果如图 7-4 所示，可以看到，网页中显示了一条贝塞尔曲线。

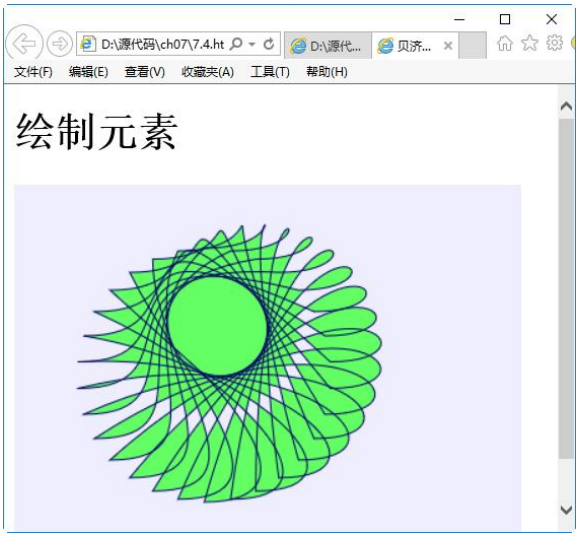

图 7-4　贝塞尔曲线

7.3　绘制渐变图形

渐变是两种或更多颜色的平滑过渡，是指在颜色集上使用逐步抽样算法，并将结果应用于描边样式和填充样式中。canvas 的绘图上下文支持两种类型的渐变：线性渐变和放射性渐变，其中，放射性渐变也称为径向渐变。

7.3.1　绘制线性渐变

使用渐变需要 3 个步骤。

step 01 创建渐变对象：

```
var gradient = cxt.createLinearGradient(0,0,0,canvas.height);
```

step 02 为渐变对象设置颜色，指明过渡方式：

```
gradient.addColorStop(0,'#fff');
gradient.addColorStop(1,'#000');
```

step 03 在 context 上为填充样式或者描边样式设置渐变：

```
cxt.fillStyle = gradient;
```

要设置显示颜色，在渐变对象上使用 addColorStop 函数即可。除了可以变换成其他颜色外，还可以为颜色设置 alpha 值，并且 alpha 值也是可以变化的。为了达到这样的效果，需要使用颜色值的另一种表示方法，如内置 alpha 组件的 CSSrgba 函数。绘制线性渐变时，会用到下面的方法，如表 7-5 所示。

表 7-5 绘制线性渐变的方法

方 法	功 能
addColorStop	允许指定两个参数：颜色和偏移量。颜色参数是指开发人员希望在偏移位置描边或填充时所使用的颜色。偏移量是一个 0.0～1.0 的数值，代表沿着渐变线渐变的距离有多远
createLinearGradient(x0,y0,x1,y1)	沿着直线从(x0,y0)至(x1,y1)绘制渐变

【例 7.5】绘制线性渐变(示例文件 ch07\7.5.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>线性渐变</title>
</head>
<body>

<h1>绘制线性渐变</h1>
<canvas id="canvas" width="400" height="300"
  style="border:1px solid red"/>

<script type="text/javascript">
var c = document.getElementById("canvas");
var cxt = c.getContext("2d");
var gradient = cxt.createLinearGradient(0,0,0,canvas.height);
gradient.addColorStop(0,'#fff');
gradient.addColorStop(1,'#000');
cxt.fillStyle = gradient;
cxt.fillRect(0,0,400,400);
</script>
</body>
</html>
```

上面的代码使用 2D 环境对象产生了一个线性渐变对象，渐变的起始点是(0,0)，渐变的结

束点是(0,canvas.height)，然后使用 addColorStop 函数设置渐变颜色，最后将渐变填充到上下文环境的样式中。

在 IE 11.0 中浏览，效果如图 7-5 所示，可以看到，在网页中创建了一个垂直方向上的渐变，从上到下颜色逐渐变深。

图 7-5　线性渐变

7.3.2　绘制径向渐变

径向渐变即放射性渐变。所谓放射性渐变，就是颜色在两个指定圆之间的锥形区域平滑变化。放射性渐变与线性渐变使用的颜色终止点是一样的。如果要实现放射性渐变，即径向渐变，需要使用 createRadialGradient 方法。

createRadialGradient(x0,y0,r0,x1,y1,r1)方法表示沿着两个圆之间的锥面绘制渐变。其中前 3 个参数代表开始的圆，圆心为(x0,y0)，半径为 r0。后 3 个参数代表结束的圆，圆心为(x1,y1)，半径为 r1。

【例 7.6】绘制径向渐变(示例文件 ch07\7.6.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>径向渐变</title>
</head>
<body>
<h1>绘制径向渐变</h1>
<canvas id="canvas" width="400" height="300" style="border:1px solid red"/>
<script type="text/javascript">
var c = document.getElementById("canvas");
var cxt = c.getContext("2d");
var gradient = cxt.createRadialGradient(
  canvas.width/2,canvas.height/2,0,canvas.width/2,canvas.height/2,150);
gradient.addColorStop(0,'#fff');
gradient.addColorStop(1,'#000');
cxt.fillStyle = gradient;
cxt.fillRect(0,0,400,400);
</script>
</body>
</html>
```

在上面的代码中，首先创建渐变对象 gradient，此处使用 createRadialGradient 方法创建了一个径向渐变，然后使用 addColorStop 添加颜色，最后将渐变填充到上下文环境中。

在 IE 11.0 中浏览，效果如图 7-6 所示，可以看到，在网页中，从圆的中心亮点开始，向外逐步发散，形成了一个径向渐变。

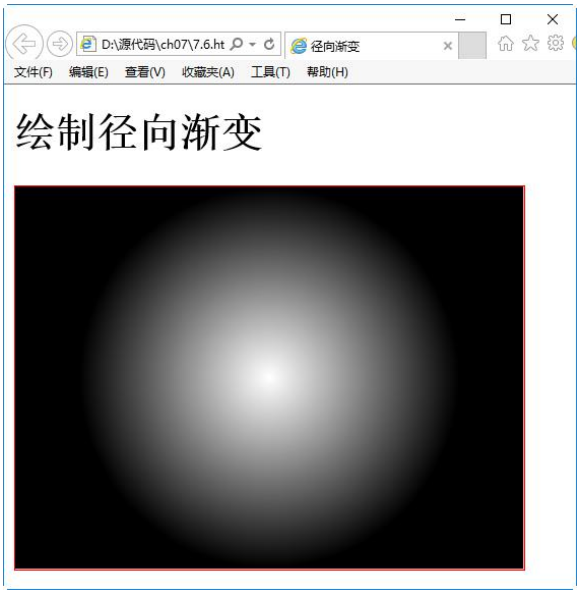

图 7-6　径向渐变

7.4　绘制变形图形

画布 canvas 不但可以使用 moveTo 这样的方法移动画笔，绘制图形和线条，还可以使用变换调整画笔下的画布，变换的方法包括平移、缩放、旋转等。

7.4.1　绘制平移效果的图形

如果要对图形实现平移，需要使用 translate(x,y)方法，该方法表示在平面上平移，即以原来的原点为参考，然后以偏移后的位置作为坐标原点。也就是说，原来在(100,100)，然后translate(1,1)，则新的坐标原点为(101,101)，而不是(1,1)。

【例 7.7】绘制坐标变换(示例文件 ch07\7.7.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制坐标变换</title>
<script>
function draw(id)
{
    var canvas = document.getElementById(id);
    if(canvas==null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#eeeeff";
    context.fillRect(0,0,400,300);
    context.translate(200,50);
    context.fillStyle = 'rgba(255,0,0,0.25)';
```

```
    for(var i=0; i<50; i++){
        context.translate(25,25);
        context.fillRect(0,0,100,50);
    }
}
</script>
</head>
<body onload="draw('canvas');">
<h1>变换原点坐标</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在 draw 函数中，使用 fillRect 方法绘制了一个矩形，然后使用 translate 方法平移到一个新位置，并从新位置开始，使用 for 循环，连续移动多次坐标原点，即多次绘制矩形。

在 IE 11.0 中浏览，效果如图 7-7 所示，可以看到，网页中从坐标位置(200,50)开始绘制矩形，并每次以指定的平移距离绘制矩形。

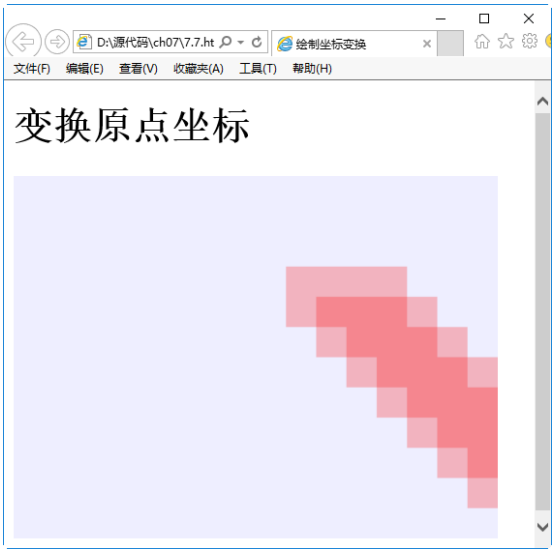

图 7-7　变换原点坐标

7.4.2　绘制缩放效果的图形

对变形图形来说，最常用的方式就是对图形进行缩放，即以原来的图形为参考，放大或者缩小图形，从而增加效果。

如果要实现图形缩放，需要使用 scale(x,y)函数，该函数带有 2 个参数，分别代表在 x，y 两个方向上的值。每个参数在 canvas 显示图像的时候，向其传递在本方向轴上图像要放大(或者缩小)的量。如果 x 值为 2，就代表所绘制的图像的全部元素都会变成 2 倍宽。如果 y 值为 0.5，表示绘制出来的图像的全部元素都会变成先前的一半高。

【例 7.8】绘制图形缩放(示例文件 ch07\7.8.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制图形缩放</title>
<script>
function draw(id)
{
    var canvas = document.getElementById(id);
    if(canvas==null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#eeeeff";
```

```
    context.fillRect(0,0,400,300);
    context.translate(200,50);
    context.fillStyle = 'rgba(255,0,0,0.25)';
    for(var i=0; i<50; i++){
        context.scale(3,0.5);
        context.fillRect(0,0,100,50);
    }
}
</script>
</head>
<body onload="draw('canvas');">
<h1>图形缩放</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在上面的代码中，缩放操作是在 for 循环中完成的，在此循环中，以原来的图形为参考物，使其在 x 轴方向增加到 3 倍宽，y 轴方向上变为原来的一半。

在 IE 11.0 中浏览，效果如图 7-8 所示，可以看到，在一个指定方向上绘制了多个矩形。

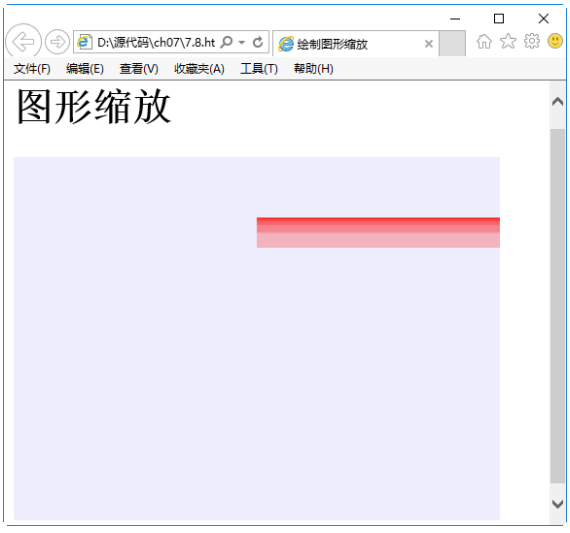

图 7-8 图形缩放

7.4.3 绘制旋转效果的图形

变换操作并不限于平移和缩放，还可以使用函数 context.rotate(angle)来旋转图像，甚至可以直接修改底层变换矩阵以完成一些高级操作，如剪裁图像的绘制路径。

例如，context.rotate(1.57)表示旋转角度参数以弧度为单位。rotate()方法默认从左上端的(0,0)开始旋转，通过指定一个角度，改变画布坐标和 Web 浏览器中的<canvas>元素的像素之间的映射，使得任意后续绘图在画布中都显示为旋转效果。

【例 7.9】绘制旋转图形(示例文件 ch07\7.9.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制旋转图像</title>
<script>
function draw(id)
{
    var canvas = document.getElementById(id);
    if(canvas==null)
        return false;
    var context = canvas.getContext('2d');
```

```
    context.fillStyle = "#eeeeff";
    context.fillRect(0,0,400,300);
    context.translate(200,50);
    context.fillStyle = 'rgba(255,0,0,0.25)';
    for(var i=0; i<50; i++){
        context.rotate(Math.PI/10);
        context.fillRect(0,0,100,50);
    }
}
</script>
</head>
<body onload="draw('canvas');">
<h1>旋转图形</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在上面的代码中，使用 rotate 方法，在 for 循环中对多个图形进行了旋转，其旋转角度相同。在 IE 11.0 中浏览，效果如图 7-9 所示，在显示页面上，多个矩形以中心弧度为原点进行旋转。

这个操作并没有旋转<canvas>元素本身。而且，旋转的角度是用弧度指定的。

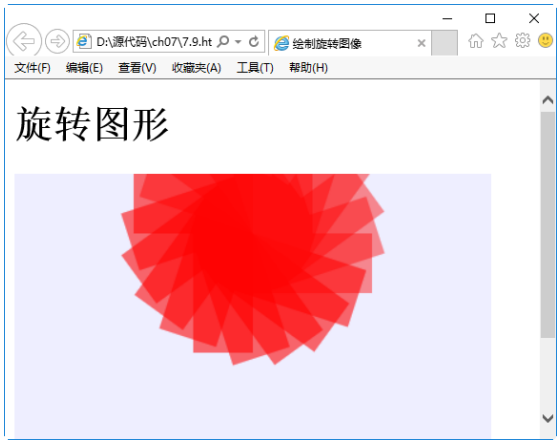

图 7-9 旋转图形

7.4.4 绘制组合效果的图形

在前面介绍的知识中，可以将一个图形画在另一个图形之上。但大多数情况下，这样做是不够的。比如，这样会受制于图形的绘制顺序。我们可以利用 globalCompositeOperation 属性来改变这些做法，不仅可以在已有图形之上再画新图形，还可以用来遮盖、清除(比 clearRect 方法强劲得多)某些区域。

其语法格式如下：

```
globalCompositeOperation = type
```

这表示设置不同形状的组合类型，其中 type 表示方的图形是已经存在的 canvas 内容，圆的图形是新的形状，其默认值为 source-over，表示在 canvas 内容上面画新的形状。

type 具有 12 个属性值，具体说明如表 7-6 所示。

表 7-6 type 的属性值

属 性 值	说 明
source-over(default)	这是默认设置，新图形会覆盖在原有内容之上
destination-over	在原有内容之下绘制新图形

续表

属 性 值	说 明
source-in	新图形仅仅出现在与原有内容重叠的部分，其他区域都变为透明
destination-in	原有内容中与新图形重叠的部分会被保留，其他区域都变为透明
source-out	新图形中只有与原有内容不重叠的部分会被绘制出来
destination-out	原有内容中与新图形不重叠的部分会被保留
source-atop	新图形中与原有内容重叠的部分会被绘制，并覆盖于原有内容之上
destination-atop	原有内容中与新内容重叠的部分会被保留，并在原有内容之下绘制新图形
lighter	两图形中的重叠部分做加色处理
darker	两图形中的重叠部分做减色处理
xor	重叠的部分会变成透明
copy	只有新图形会被保留，其他都被清除

【例 7.10】绘制图形组合(示例文件 ch07\7.10.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制图形组合</title>
<script>
function draw(id)
{
    var canvas = document.getElementById(id);
    if(canvas==null)
        return false;
    var context = canvas.getContext('2d');
    var oprtns = new Array(
        "source-atop",
        "source-in",
        "source-out",
        "source-over",
        "destination-atop",
        "destination-in",
        "destination-out",
        "destination-over",
        "lighter",
        "copy",
        "xor"
    );
    var i = 10;
    context.fillStyle = "blue";
    context.fillRect(10,10,60,60);
    context.globalCompositeOperation = oprtns[i];
    context.beginPath();
    context.fillStyle = "red";
    context.arc(60,60,30,0,Math.PI*2,false);
    context.fill();
}
</script>
```

```
</head>
<body onload="draw('canvas');">
<h1>图形组合</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

在上述代码中，首先创建了一个 oprtns 数组，用于存储 type 的 12 个值，然后绘制了一个矩形，并使用 content 上下文对象设置了图形的组合方式，即采用新图形显示，其他被清除的方式，最后使用 arc 绘制了一个圆。

在 IE 11.0 中浏览，效果如图 7-10 所示，在显示页面上绘制了一个矩形和圆，但矩形和圆重叠的地方以空白显示。

图 7-10　图形组合

7.4.5　绘制带阴影的图形

在画布 canvas 上绘制带有阴影效果的图形非常简单，只需要设置几个属性即可。这些属性分别为 shadowOffsetX、shadowOffsetY、shadowBlur 和 shadowColor。

属性 shadowColor 表示阴影的颜色，其值与 CSS 颜色值一致。shadowBlur 表示设置阴影模糊程度，此值越大，阴影越模糊。shadowOffsetX 和 shadowOffsetY 属性表示阴影的 x 和 y 偏移量，单位是像素。

【例 7.11】绘制带阴影的图形(示例文件 ch07\7.11.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制阴影效果图形</title>
</head>
<body>
<canvas id="my_canvas" width="200" height="200"
  style="border:1px solid #ff0000">
</canvas>
<script type="text/javascript">
var elem = document.getElementById("my canvas");
if (elem && elem.getContext)  {
    var context = elem.getContext("2d");
    //shadowOffsetX 和 shadowOffsetY：阴影的 x 和 y 偏移量，单位是像素
    context.shadowOffsetX = 15;
    context.shadowOffsetY = 15;
    //shadowBlur：设置阴影模糊程度。此值越大，阴影越模糊
    //其效果与 Photoshop 的高斯模糊滤镜相同
    context.shadowBlur = 10;
    //shadowColor：阴影颜色。其值与 CSS 颜色值一致
    //context.shadowColor = 'rgba(255, 0, 0, 0.5)';  或下面的十六进制表示法
    context.shadowColor = '#f00';
```

```
    context.fillStyle = '#00f';
    context.fillRect(20, 20, 150, 100);
}
</script>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 7-11 所示，在显示页面上显示了一个蓝色矩形，其阴影为红色矩形。

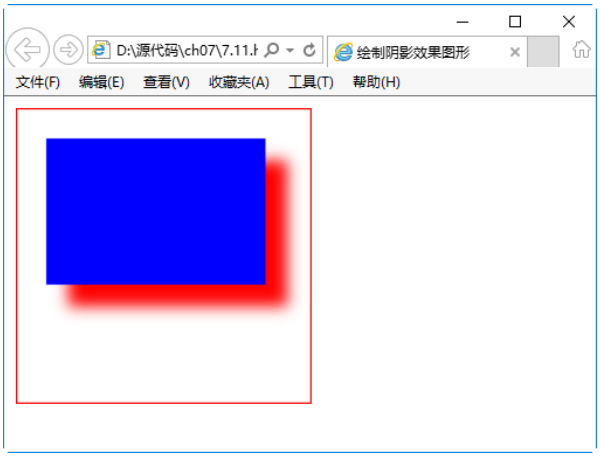

图 7-11　带有阴影的图形

7.5　使 用 图 像

画布 canvas 有一项功能就是可以引入图像，可以用于图片合成或者制作背景等。而目前仅可以在图像中加入文字。只要是 Geck 支持的图像(如 PNG、GIF、JPEG 等)都可以引入 canvas 中，而且其他 canvas 元素也可以作为图像的来源。

7.5.1　绘制图像

要在画布 canvas 上绘制图像，需要先有一张图片。这张图片可以是已经存在的<img>元素，或者通过 JavaScript 创建。

无论采用哪种方式，都需要在绘制 canvas 之前完全加载这张图片。浏览器通常会在页面脚本执行的同时异步加载图片。如果试图在图片未完全加载之前就将其呈现到 canvas 上，那么 canvas 将不会显示任何图片。

捕获和绘制图像完全是通过 drawImage 方法完成的，它可以接收不同的 HTML 参数，具体含义如表 7-7 所示。

表 7-7 绘制图像的方法

方 法	说 明
drawImage(image,dx,dy)	接收一张图片，并将其画到 canvas 中。给出的坐标(dx,dy)代表图片的左上角。例如，坐标(0,0)将把图片画到 canvas 的左上角
drawImage(image,dx,dy,dw,dh)	接收一张图片，将其缩放到宽度为 dw 和高度为 dh，然后把它画到 canvas 上的(dx,dy)位置
drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)	接收一张图片，通过参数(sx,sy,sw,sh)指定图片裁剪的范围，缩放到(dw,dh)的大小，最后把它画到 canvas 上的(dx,dy)位置

【例 7.12】绘制图像(示例文件 ch07\7.12.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制图像</title>
</head>
<body>
<canvas id="canvas" width="300" height="200" style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
window.onload=function(){
    var ctx = document.getElementById("canvas").getContext("2d");
    var img = new Image();
    img.src = "01.jpg";
    img.onload=function(){
        ctx.drawImage(img,0,0);
    }
}
</script>
</body>
</html>
```

在上述代码中，使用窗口的 onload 加载事件，即在页面被加载时执行函数。在函数中，创建上下文对象 ctx，并创建 Image 对象 img；然后使用 img 对象的 src 属性设置图片来源，最后使用 drawImage 画出当前的图像。

在 IE 11.0 中浏览，效果如图 7-12 所示。

图 7-12 绘制图像

7.5.2 平铺图像

使用画布 canvas 绘制图像有多种用处，其中一个用处就是将绘制的图像作为背景图片

使用。在做背景图片时，如果显示图片的区域大小不能直接设定，通常将图片以平铺的方式显示。

HTML 5 Canvas API 支持图片平铺，此时需要调用 createPattern 函数，即调用 createPattern 函数来替代先前的 drawImage 函数。函数 createPattern 的语法格式如下：

```
createPattern(image,type)
```

其中 image 表示要绘制的图像，type 表示平铺的类型，其具体含义如表 7-8 所示。

表 7-8 type 参数含义

参 数 值	说 明
no-repeat	不平铺
repeat-x	横方向平铺
repeat-y	纵方向平铺
repeat	全方向平铺

【例 7.13】平铺图像(示例文件 ch07\7.13.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制图像平铺</title>
</head>
<body onload="draw('canvas');">
<h1>图形平铺</h1>
<canvas id="canvas" width="800" height="600"></canvas>
<script>
function draw(id){
    var canvas = document.getElementById(id);
    if(canvas==null){
        return false;
    }
    var context = canvas.getContext('2d');
    context.fillStyle = "#eeeeff";
    context.fillRect(0,0,800,600);
    image = new Image();
    image.src = "02.jpg";
    image.onload = function(){
        var ptrn = context.createPattern(image,'repeat');
        context.fillStyle = ptrn;
        context.fillRect(0,0,800,600);
    }
}
</script>
</body>
</html>
```

在上述代码中，首先用 fillRect 创建了一个宽度为 800、高度为 600，左上角坐标位置为(0,0)的矩形，接着创建了一个 Image 对象，src 表示链接一个图像源，然后使用 createPattern 绘制一幅图像，其方式是完全平铺，并将这幅图像作为一个模式填充到矩形中。最后绘制这个矩形，此矩形的大小完全覆盖原来的图形。

在 IE 11.0 中浏览，效果如图 7-13 所示，即在显示页面上绘制一幅图像，其图像以平铺的方式充满整个矩形。

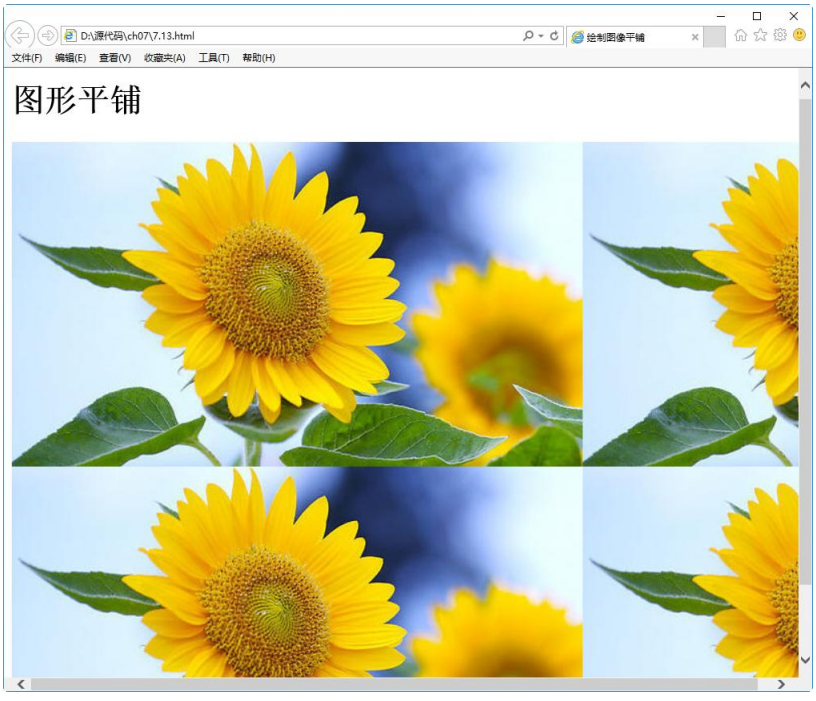

图 7-13　图像平铺

7.5.3　裁剪图像

要完成对图像的裁剪，需要用到 clip 方法。clip 方法表示给 canvas 设置一个剪辑区域，在调用 clip 方法之后，所有代码只对这个设定的剪辑区域有效，不会影响其他地方，这个方法在要进行局部更新时很有用。在默认情况下，剪辑区域是一个左上角在(0,0)，宽和高分别等于 canvas 元素的宽和高的矩形。

【例 7.14】裁剪图像(示例文件 ch07\7.14.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>绘制图像裁剪</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body onload="draw('canvas');">
<h1>图像裁剪实例</h1>
<canvas id="canvas" width="400" height="300"></canvas>
<script>
function draw(id){
    var canvas = document.getElementById(id);
    if(canvas==null){
        return false;
    }
```

```
    var context = canvas.getContext('2d');
    var gr = context.createLinearGradient(0,400,300,0);
    gr.addColorStop(0,'rgb(255,255,0)');
    gr.addColorStop(1,'rgb(0,255,255)');
    context.fillStyle = gr;
    context.fillRect(0,0,400,300);
    image = new Image();
    image.onload=function(){
        drawImg(context,image);
    };
    image.src = "02.jpg";
}
function drawImg(context,image){
    create8StarClip(context);
    context.drawImage(image,-50,-150,300,300);
}
function create8StarClip(context){
    var n = 0;
    var dx = 100;
    var dy = 0;
    var s = 150;
    context.beginPath();
    context.translate(100,150);
    var x = Math.sin(0);
    var y = Math.cos(0);
    var dig = Math.PI/5*4;
    for(var i=0; i<8; i++){
        var x = Math.sin(i*dig);
        var y = Math.cos(i*dig);
        context.lineTo(dx+x*s,dy+y*s);
    }
    context.clip();
}
</script>
</body>
</html>
```

在上面的代码中，创建了 3 个 JavaScript 函数，其中 create8StarClip 函数完成多边的图形创建，以此图形作为裁剪的依据。drawImg 函数表示绘制一个图形，其图形带有裁剪区域。draw 函数完成对画布对象的获取，并定义一个线性渐变，然后创建了一个 Image 对象。

在 IE 11.0 中浏览，效果如图 7-14 所示，即在显示页面上绘制了一个五边形，图像作为五边形的背景显示，从而实现了对图像的裁剪。

图 7-14　图像裁剪

7.5.4　图像的像素化处理

在画布中，可以使用 ImageData 对象保存图像的像素值，它有 width、height 和 data 这 3 个

属性，其中 data 属性就是一个连续数组，图像的所有像素值就保存在 data 中。

data 属性保存像素值的方法如下：

```
imageData.data[index*4+0]
imageData.data[index*4+1]
imageData.data[index*4+2]
imageData.data[index*4+3]
```

上面取出了 data 数组中相邻的 4 个值，这 4 个值分别代表图像中第 index+1 像素的红色、绿色、蓝色和透明度值的大小。需要注意的是，index 从 0 开始，图像中共有 width*height 像素，数组中总共保存 width*height*4 个数值。

画布对象有 3 个方法，用来创建、读取和设置 ImageData 对象，如表 7-9 所示。

表 7-9　创建画布对象的方法

方　法	说　明
createImageData(width, height)	在内存中创建一个指定大小的 ImageData 对象(即像素数组)，对象中的像素点都是黑色透明的，即 rgba(0,0,0,0)
getImageData(x, y, width, height)	返回一个 ImageData 对象，这个 ImageData 对象中包含指定区域的像素数组
putImageData(data, x, y)	将 ImageData 对象绘制到屏幕的指定区域

【例 7.15】图像像素处理(示例文件 ch07\7.15.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>图像像素处理</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body onload="draw('canvas');">
<h1>像素处理示例</h1>
<canvas id="canvas" width="400" height="300"></canvas>
<script>
function draw(id){
    var canvas = document.getElementById(id);
    if(canvas==null){
        return false;
    }
    var context = canvas.getContext('2d');
    image = new Image();
    image.src = "01.jpg";
    image.onload=function(){
        context.drawImage(image,0,0);
        var imagedata = context.getImageData(0,0,image.width,image.height);
        for(var i=0,n=imagedata.data.length; i<n; i+=4){
            imagedata.data[i+0] = 255-imagedata.data[i+0];
            imagedata.data[i+1] = 255-imagedata.data[i+2];
            imagedata.data[i+2] = 255-imagedata.data[i+1];
        }
        context.putImageData(imagedata,0,0);
    };
```

```
}
</script>
</body>
</html>
```

在上面的代码中，使用 getImageData 方法获取一个 ImageData 对象，并包含相关的像素数组。在 for 循环中，对像素值重新赋值，最后使用 putImageData 将处理过的图像在画布上绘制出来。

在 IE 11.0 中浏览，效果如图 7-15 所示，在页面上显示了一幅图像，其图像明显经过像素处理，没有原来清晰。

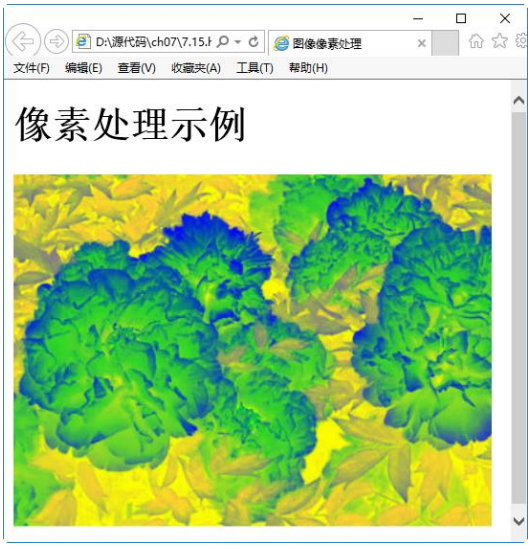

图 7-15　像素处理

7.6　绘 制 文 字

在画布中绘制字符串(文字)的方式，与操作其他路径对象的方式相同，可以描绘文本轮廓和填充文本内部。同时，所有能够应用于其他图形的变换和样式都能够应用于文本。

文本绘制功能涉及的方法如表 7-10 所示。

表 7-10　绘制文本的方法

方　法	说　明
fillText(text,x,y,maxwidth)	绘制带 fillStyle 填充的文字，拥有文本参数以及用于指定文本位置的坐标的参数。maxwidth 是可选参数，用于限制字体大小，它会将文本字体强制收缩到指定尺寸
strokeText(text,x,y,maxwidth)	绘制只有 strokeStyle 边框的文字，其参数含义与上一个方法相同
measureText	该函数会返回一个度量对象，它包含在当前 context 环境下指定文本的实际显示宽度

为了保证文本在各浏览器下都能正常显示，在绘制上下文里有以下字体属性。

- font：可以是 CSS 字体规则中的任何值。包括字体样式、字体变种、字体大小与粗细、行高和字体名称。
- textAlign：控制文本的对齐方式。它类似于(但不完全等同于)CSS 中的 text-align，可能的取值为 start、end、left、right 和 center。
- textBaseline：控制文本相对于起点的位置。可以取值为 top、hanging、middle、alphabetic、ideographic 和 bottom。对于简单的英文字母，可以放心地使用 top、middle 或 bottom 作为文本基线。

【例 7.16】绘制文字(示例文件 ch07\7.16.html)。

```
<!DOCTYPE html>
<html>
<head>
<title>Canvas</title>
</head>
<body>
<canvas id="my canvas" width="200" height="200"
  style="border:1px solid #ff0000">
</canvas>
<script type="text/javascript">
var elem = document.getElementById("my_canvas");
if (elem && elem.getContext) {
    var context = elem.getContext("2d");
    context.fillStyle = '#00f';
    //font: 文字字体，同 CSSfont-family 属性
    context.font = 'italic  30px 微软雅黑';          //斜体 30 像素 微软雅黑字体
    //textAlign: 文字水平对齐方式
    //可取属性值: start, end, left,right, center。默认值:start
    context.textAlign = 'left';
    //文字竖直对齐方式
    //可取属性值: top, hanging, middle,alphabetic,ideographic, bottom
    //默认值: alphabetic
    context.textBaseline = 'top';
    //要输出的文字内容，文字位置坐标，第 4 个参数为可选选项——最大宽度
    //如果需要的话，浏览器会缩减文字，以让它适应指定宽度
    context.fillText('生日快乐!', 0, 0,50);       //有填充
    context.font = 'bold 30px sans-serif';
    context.strokeText('生日快乐!', 0, 50,100);  //只有文字边框
}
</script>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 7-16 所示。在页面上显示了一个画布边框，画布中显示了两个不同的字符串，第一个字符串以斜体显示，其颜色为蓝色。第二个字符串的文本颜色为浅黑色，加粗显示。

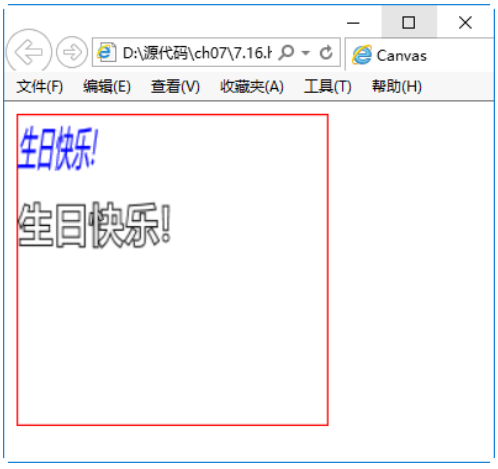

图 7-16　绘制文字

7.7　图形的保存与恢复

在画布对象中绘制图形或图像时，可以将这些图形或者图形的状态进行改变，即永久保存图形或图像。

7.7.1　保存与恢复状态

在画布对象中，由两个方法管理绘制状态的当前栈，save 方法把当前状态压入栈中，而 restore 方法从栈顶弹出状态。绘制状态不会覆盖对画布所做的每件事情。其中，save 方法用来保存 canvas 的状态。save 之后，可以调用 canvas 的平移、缩放、旋转、裁剪等操作。restore 方法用来恢复 canvas 先前保存的状态，防止 save 后对 canvas 执行的操作对后续的绘制有影响。save 和 restore 要配对使用(restore 可以比 save 少，但不能多)，如果 restore 调用的次数比 save 多，会引发 Error。

【例 7.17】保存与恢复图形的状态(示例文件 ch07\7.17.html)。

```
<!DOCTYPE html>
<html>
<head><title>保存与恢复</title></head>
<body>
<canvas id="myCanvas" width="500" height="400"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "rgb(0,0,255)";
ctx.save();
ctx.fillRect(50,50,100,100);
```

```
ctx.fillStyle = "rgb(255,0,0)";
ctx.save();
ctx.fillRect(200,50,100,100);
ctx.restore();
ctx.fillRect(350,50,100,100);
ctx.restore();
ctx.fillRect(50, 200, 100, 100);
</script>
</body>
</html>
```

在上面的代码中，绘制了 4 个矩形，在绘制第 1 个矩形之前，定义当前矩形的显示颜色，并将此样式加入到栈中，然后创建矩形。在绘制第 2 个矩形之前，重新定义了矩形的显示颜色，并使用 save 将此样式压入到栈中，然后创建矩形。在绘制第 3 个矩形之前，使用 restore 恢复当前显示颜色，即调用栈中最上层的颜色，之后绘制矩形。在绘制第 4 个矩形之前，继续使用 restore 方法，调用最后一个栈中的元素来定义矩形颜色。

在 IE 11.0 中浏览，效果如图 7-17 所示，在显示页面上绘制了 4 个矩形，第 1 个和第 4 个矩形显示为蓝色，第 2 个和第 3 个矩形显示为红色。

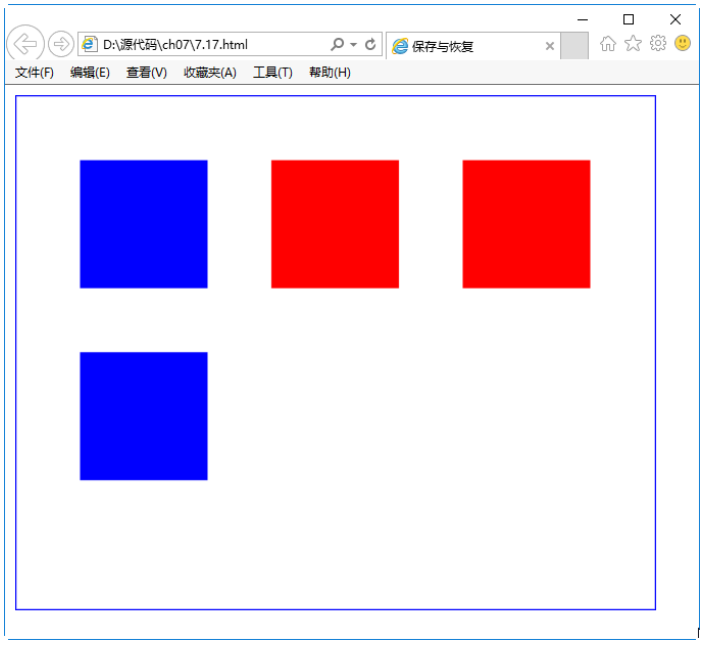

图 7-17　保存和恢复

7.7.2　保存文件

绘制出漂亮的图形后，有时需要保存这些劳动成果。这时可以将当前的画布元素(而不是 2D 环境)的状态导出到数据 URL。导出很简单，可以利用 toDataURL 方法来完成，它可以调用不同的图片格式。目前 Firefox 和 Opera 浏览器只支持 PNG 格式，Safari 支持 GIF、PNG 和 JPG 格式。大多数浏览器支持读取 base64 编码内容，例如一幅图像。URL 的格式如下：

```

```

它以一个 data 开始，然后是 mine 类型，之后是编码和 base64，最后是原始数据。这些原始数据就是画布元素所要导出的内容，并且浏览器能够将数据编码为真正的资源。

【例 7.18】保存图形(示例文件 ch07\7.18.html)。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="500" height="500"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var cxt = c.getContext("2d");
cxt.fillStyle = 'rgb(0,0,255)';
cxt.fillRect(0,0,cxt.canvas.width,cxt.canvas.height);
cxt.fillStyle = "rgb(0,255,0)";
cxt.fillRect(10,20,50,50);
window.location = cxt.canvas.toDataURL('image/png');
</script>
</body>
</html>
```

在上面的代码中，使用 canvas.toDataURL 语句将当前绘制的图像保存到 URL 数据中。在 Firefox 53.0 中浏览，效果如图 7-18 所示。浏览器的地址栏中显示的是 URL 数据。

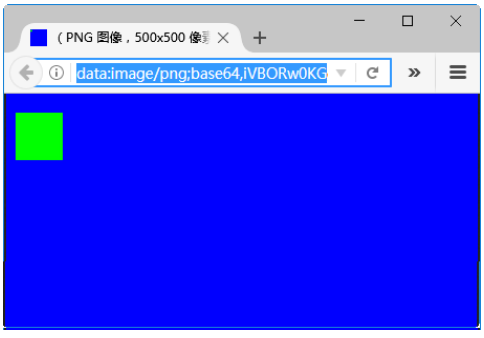

图 7-18　保存图形

7.8　综合案例——绘制火柴棒人物

漫画中最常见的一种图形，就是火柴棒人，通过简单的几个笔画，就可以绘制一个传神的动漫人物。使用 canvas 和 JavaScript 同样可以绘制一个火柴棒人物。其具体操作步骤如下。

step 01 分析需求。

一个火柴棒人，由一个脸部和一个身躯组成。脸部是一个圆形，其中包括眼睛和嘴；身

躯由几条直线组成，包括手、腿等。实际上此案例就是绘制圆形、弧形和直线的组合。示例效果如图 7-19 所示。

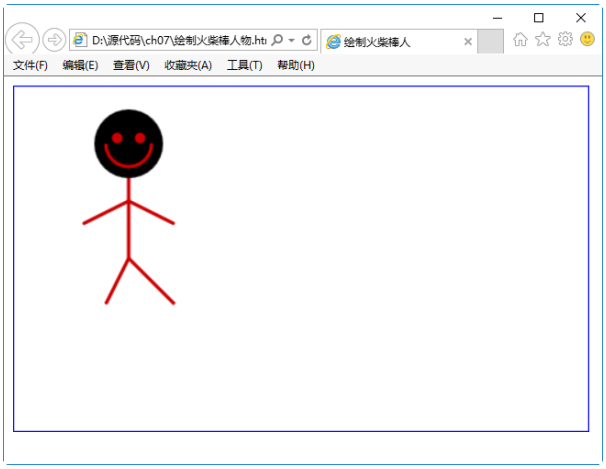

图 7-19　火柴棒人

step 02 实现 HTML 页面，定义画布 canvas：

```
<!DOCTYPE html>
<html>
<title>绘制火柴棒人</title>
<body>
<canvas id="myCanvas" width="500" height="300"
  style="border:1px solid blue">
    Your browser does not support the canvas element.
</canvas>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 7-20 所示，页面显示了一个画布边框。

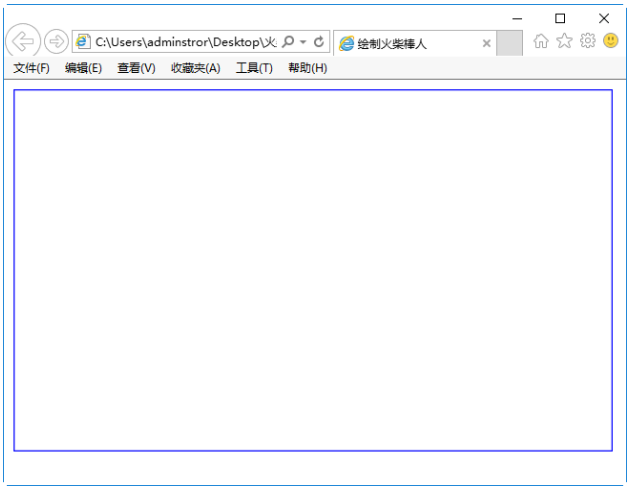

图 7-20　定义画布边框

step 03 实现头部轮廓绘制：

```
<script type="text/javascript">
var c = document.getElementById("myCanvas");
var cxt = c.getContext("2d");
cxt.beginPath();
cxt.arc(100,50,30,0,Math.PI*2,true);
cxt.fill();
</script>
```

上述代码会产生一个实心的、填充的头部，即圆形。在 arc 函数中，x 和 y 的坐标为(100,50)，半径为 30 像素，另两个参数为弧形的开始和结束，第 6 个参数表示绘制弧形的方向，即顺时针和逆时针方向。

在 IE 11.0 中浏览，效果如图 7-21 所示，页面显示实心圆，其颜色为黑色。

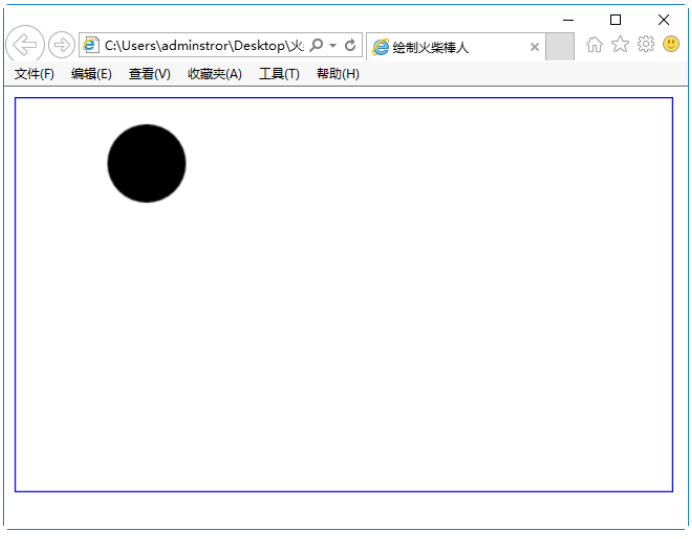

图 7-21 绘制头部轮廓

step 04 用 JavaScript 绘制笑脸：

```
cxt.beginPath();
cxt.strokeStyle = '#c00';
cxt.lineWidth = 3;
cxt.arc(100,50,20,0,Math.PI,false);
cxt.stroke();
```

此处使用 beginPath 方法，表示重新绘制，并设定线条宽度，然后绘制一个弧形。

在 IE 11.0 中浏览，效果如图 7-22 所示，页面上显示了一个漂亮的半圆式笑脸。

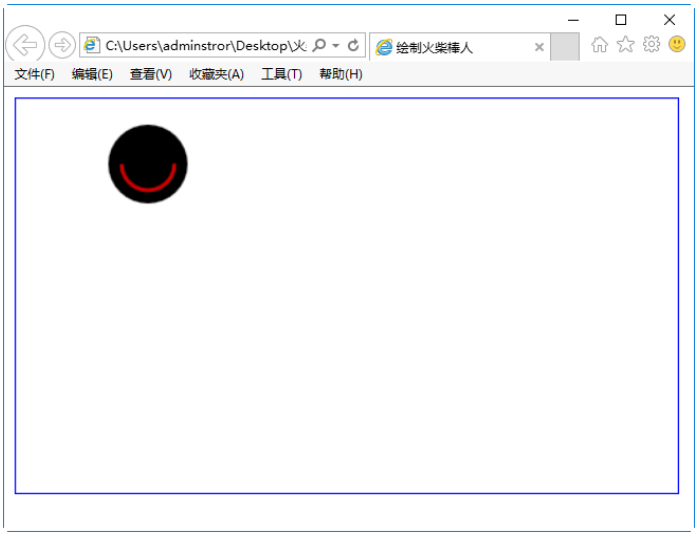

图 7-22　绘制笑脸

step 05 绘制眼睛：

```
cxt.beginPath();
cxt.fillStyle = "#c00";
cxt.arc(90,45,3,0,Math.PI*2,true);
cxt.fill();
cxt.moveTo(113,45);
cxt.arc(110,45,3,0,Math.PI*2,true);
cxt.fill();
cxt.stroke();
```

首先填充弧线，创建一个实体样式的眼睛，用 arc 绘制左眼，然后使用 moveTo 绘制右眼。在 IE 11.0 中浏览，效果如图 7-23 所示，页面显示了一双眼睛。

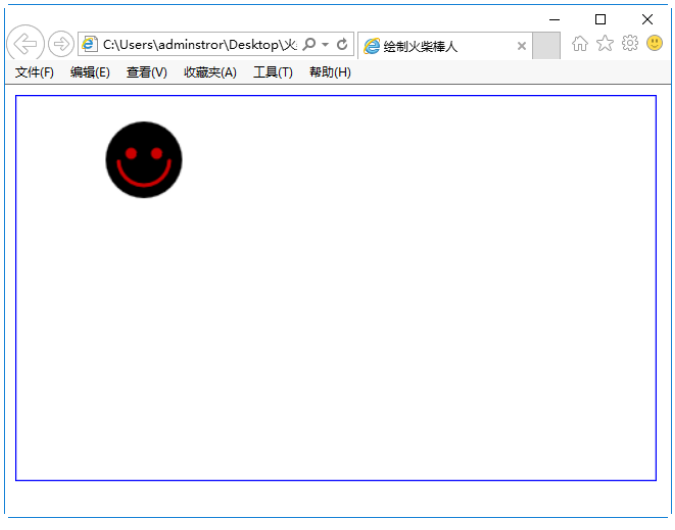

图 7-23　绘制眼睛

step 06 绘制身躯：

```
cxt.moveTo(100,80);
cxt.lineTo(100,150);
cxt.moveTo(100,100),
cxt.lineTo(60,120);
cxt.moveTo(100,100);
cxt.lineTo(140,120);
cxt.moveTo(100,150);
cxt.lineTo(80,190);
cxt.moveTo(100,150);
cxt.lineTo(140,190);
cxt.stroke();
```

上面的代码以 moveTo 作为起始坐标，以 lineTo 作为终点坐标，绘制不同的直线，这些直线需要在不同地方汇集，两只手在坐标位置(100,100)处交叉，两只脚在坐标位置(100,150)处交叉。

在 IE 11.0 中浏览，效果如图 7-24 所示，页面显示一个火柴棒人，与上一个图形相比，多了一个身躯。

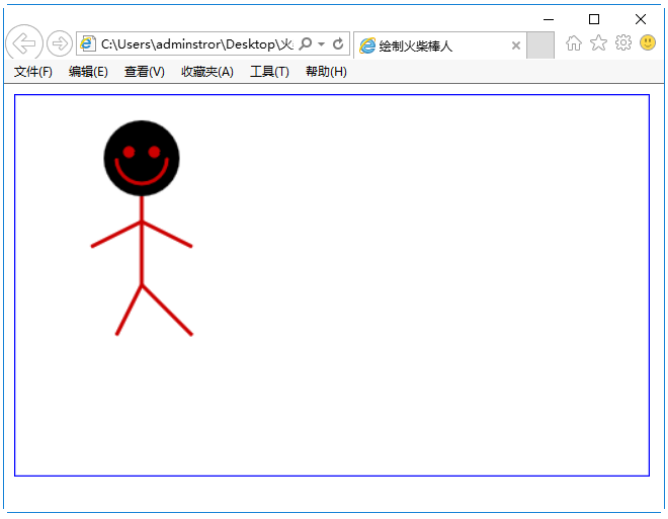

图 7-24 绘制身躯

7.9 跟我学上机——绘制商标

绘制商标是 canvas 画布的用途之一，如绘制 Adidas 和 Nike 商标。Nike 的图标比 Adidas 的复杂得多，Adidas 的图标都是直线组成的，而 Nike 多了曲线。实现本例的具体操作步骤如下。

step 01 分析需求。

要绘制两条曲线，需要找到曲线的参考点(参考点决定了曲线的曲率)，这需要慢慢地移动，然后再看效果，反反复复。quadraticCurveTo(30,79,99,78)函数有两组坐标：第一组坐标为控制点，决定曲线的曲率；第二组坐标为终点。

step 02 构建 HTML，实现 canvas 画布：

```
<!DOCTYPE html>
<html>
<head>
<title>绘制商标</title>
</head>
<body>
<canvas id="Nike" width="375px" height="132px"
  style="border:1px solid #000;">
</canvas>
</body>
</html>
```

在 IE 11.0 中浏览，效果如图 7-25 所示，只显示了一个画布边框，其内容还没有绘制。

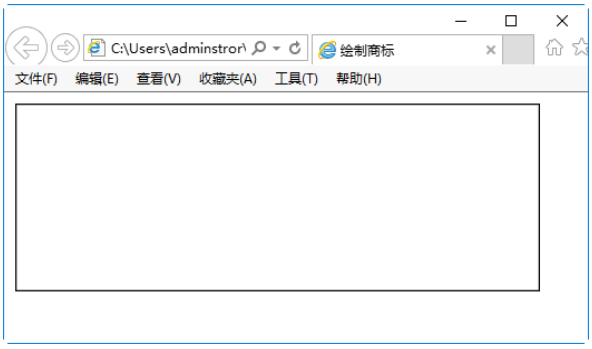

图 7-25　定义画布边框

step 03 用 JavaScript 实现基本图形：

```
<script>
function drawNike(){
    //取得 canvas 元素及其绘图上下文
    var canvas = document.getElementById('nike');
    var context = canvas.getContext('2d');
    //保存当前的绘图状态
    context.save();
    //开始绘制打钩的轮廓
    context.beginPath();
    context.moveTo(53,0);
    //绘制上半部分曲线，第一组坐标为控制点，决定曲线的曲率；第二组坐标为终点
    context.quadraticCurveTo(30,79,99,78);
    context.lineTo(371,2);
    context.lineTo(74,134);
    context.quadraticCurveTo(-55,124,53,0);
    //用红色填充
    context.fillStyle = "#da251c";
    context.fill();
    //用 3 像素深红线条描边
    context.lineWidth = 3;
    //连接处平滑
    context.lineJoin = 'round';
    context.strokeStyle = "#d40000";
```

```
    context.stroke();
    //恢复原有的绘图状态
    context.restore();
}
window.addEventListener("load",drawNike,true);
</script>
```

在 IE 11.0 中浏览，效果如图 7-26 所示，显示了一个商标图案，颜色为红色。

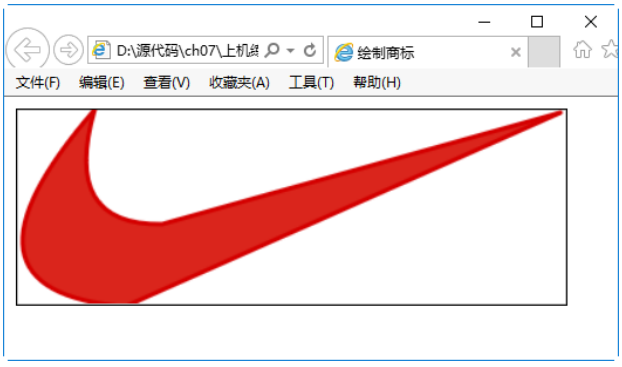

图 7-26　绘制商标

7.10　疑 难 解 惑

疑问 1：canvas 的宽度和高度是否可以在 CSS 属性中定义呢？

答：添加 canvas 标签时，会在 canvas 的属性里填写要初始化的 canvas 的高度和宽度：

```
<canvas width="500" height="400">Not Supported!</canvas>
```

如果把高度和宽度写在 CSS 中，结果会发现，在绘图时坐标获取出现差异，canvas.width 和 canvas.height 分别是 300 和 150，与预期的不一样。这是因为 canvas 要求这两个属性必须随 canvas 标记一起出现。

疑问 2：画布中 Stroke 和 Fill 二者的区别是什么？

答：在 HTML 5 中，图形分为两大类：第一类称作 Stroke，就是轮廓、勾勒或者线条，总之，图形是由线条组成的；第二类称作 Fill，就是填充区域。上下文对象中有两个绘制矩形的方法，可以让我们很好地理解这两大类图形的区别：一个是 strokeRect，还有一个是 fillRect。