源码下载

第5章 使用 Canvas 画图

HTML4 的画图能力很弱，通常只能在网页中显示指定的图像文件。HTML5 提供了 Canvas 元素，可以在网页中定义一个画布，然后使用 Canvas 绘图方法在画布中画图。在游戏开发中大量使用 Canvas 画图。本章介绍在 HTML5 中如何使用 Canvas 画图。

5.1 Canvas 元素

Canvas 就是画布，可以进行画任何的线、图形、填充等一系列的操作。Canvas 是 HTML5 出现的新元素，它有自己的属性、方法和事件，其中就有绘图的方法，JavaScript 能够调用 Canvas 绘图方法来进行绘图。另外 Canvas 不仅仅提供简单的二维矢量绘图，也提供了三维的绘图，以及图片处理等一系列的 API 支持。

5.1.1 Canvas 元素的定义语法

Canvas 元素的定义语法如下：

```
<canvas id="xxx" height=… width=…>…</canvas>
```

Canvas 元素的常用属性如下：

id 是 Canvas 元素的标识；height 是 Canvas 画布的高度，单位为像素；width 是 Canvas 画布的宽度，单位为像素。

例如在 HTML 文件中定义一个 Canvas 画布，id 为 myCanvas，高和宽各为 100 像素，代码如下：

```
<canvas id="myCanvas" height=100 width=100>
您的浏览器不支持 canvas。
</canvas>
```

<canvas>和之间的字符串</canvas>指定当浏览器不支持 Canvas 时显示的字符串。

注意：Internet Explorer 9、Firefox、Opera、Chrome 和 Safari 支持 Canvas 元素。Internet Explorer 8 及其之前版本不支持 Canvas 元素。

5.1.2 使用 JavaScript 获取网页中的 Canvas 对象

在 JavaScript 中，可以使用 document.getElementById()方法获取网页中的对象，语法如下：

```
document.getElementById(对象 id)
```

例如，获取定义的 myCanvas 对象的代码如下：

```
<canvas id="myCanvas" height=100 width=100>
您的浏览器不支持 canvas。
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
</script>
```

得到的对象 c 即为 myCanvas 对象。要在其中绘图还需要获得 myCanvas 对象的 2d 上下文对象，代码如下：

```
var ctx=c.getContext("2d");//获得 myCanvas 对象的 2d 上下文对象
```

Canvas 绘制图形都是依靠 Canvas 对象的上下文对象。上下文对象用于定义如何在画布上绘图。顾名思义，2d 上下文支持在画布上绘制 2D 图形、图像和文本。

5.2 坐标与颜色

5.2.1 坐标系统

在实际的绘图中，我们所关注的一般都是指设备坐标系，此坐标系以像素为单位，像素指的是屏幕上的亮点。每个像素都有一个坐标点与之对应，左上角的坐标设为（0，0），向右为 X 正轴，向下为 Y 正轴。一般情况下以（x，y）代表屏幕上某个像素的坐标点，其中水平以 X 坐标值表示，垂直以 Y 坐标值表示。例如，在图 5-1 所示的坐标系统中画一个点，该点的坐标（*x*，*y*）是（4，3）。

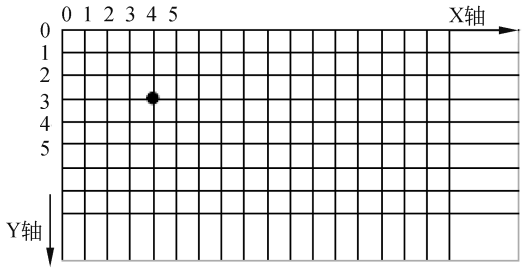

图 5-1　Canvas 坐标的示意图

计算机作图是在一个事先定义好的坐标系统中进行的，这与日常生活中的绘图方式有着很大的区别。图形的大小、位置等都与绘图区或容器的坐标有关。

5.2.2 颜色的表示方法

1．颜色关键字

W3C 的 HTML 4.0 标准仅支持 16 种颜色名，它们是 aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white、yellow。如果需要使用其他的颜色，需要使用十六进制的颜色值。

2．十六进制颜色值

可以使用一个十六进制的字符串表示颜色，格式为#RGB。其中，R 表示红色分量，G 表示绿色分量，B 表示蓝色分量。每种颜色的最小值是 0（十六进制：#00）。最大值是 255（十六进制：#FF）。例如#FF0000 表示红色，#00FF00 表示绿色，#0000FF 表示蓝色，#A020F0 表示紫色，#FFFFFF 表示白色，#000000 表示黑色。

3．RGB 颜色值

RGB 颜色值可以使用如 rgb(红色分量,绿色分量,蓝色分量)形式表示颜色。表 5-1 是十六进制字符串表示颜色与 RGB 颜色值的对照表。

表 5-1　十六进制颜色值与 RGB 颜色值对照表

Color HEX	Color RGB	颜色	Color HEX	Color RGB	颜色
#000000	rgb(0,0,0)	黑色	#00FFFF	rgb(0,255,255)	青色
#FF0000	rgb(255,0,0)	红色	#FF00FF	rgb(255,0,255)	深红
#00FF00	rgb(0,255,0)	绿色	#C0C0C0	rgb(192,192,192)	灰色
#0000FF	rgb(0,0,255)	蓝色	#FFFFFF	rgb(255,255,255)	白色
#FFFF00	rgb(255,255,0)	黄色	#FF8000	rgb(255,128,0)	桔黄

5.3　绘制图形

5.3.1　绘制直线

在 JavaScript 中可以使用 Canvas API 绘制直线，具体过程如下：

（1）在网页中使用 Canvas 元素定义一个 Canvas 画布，用于绘画。

```
var c=document.getElementById("myCanvas"); //获取网页中的 canvas 对象
```

（2）使用 JavaScript 获取网页中的 Canvas 对象，并获取 Canvas 对象的 2d 上下文 ctx。使用 2d 上下文可以调用 Canvas API 绘制图形。

```
var ctx=c.getContext("2d");                    //获取 canvas 对象的上下文
```

（3）调用 beginPath()方法，指示开始绘图路径，即开始绘图。语法如下：

```
ctx.beginPath();
```

（4）调用 moveTo()方法将坐标移至直线起点。moveTo()方法的语法如下：

```
ctx.moveTo(x, y);
```

x 和 y 为要移动至的坐标。

（5）调用 lineTo()方法绘制直线。lineTo()方法的语法如下：

```
ctx.lineTo(x, y);
```

x 和 y 为直线的终点坐标。

（6）调用 stroke()方法，绘制图形的边界轮廓。语法如下：

```
ctx. stroke();
```

【例 5-1】 使用连续画线的方法绘制一个三角形，代码如下：

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" height=200 width=200>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function drawtriangle()
{
 var c=document.getElementById("myCanvas");     //获取网页中的 canvas 对象
 var ctx=c.getContext("2d");                    //获取 canvas 对象的上下文
ctx.beginPath();                                //开始绘图路径
 ctx.moveTo(100,0);      //将坐标移至直线起点
 ctx.lineTo(50,100);     //绘制直线
 ctx.lineTo(150,100);    //绘制直线
 ctx.lineTo(100,0);      //绘制直线
 ctx.closePath();        //闭合路径，不是必需的，如果线的终点跟起点一样会自动闭合
 ctx.stroke();           //通过线条绘制轮廓（边框）
}
window.addEventListener("load", drawtriangle, true);
</script>
</body>
</html>
```

浏览例 5-1 的结果如图 5-2 所示。

图 5-2 Canvas 绘制一个三角形

【例 5-2】 一个通过画线绘制复杂菊花图形的例子。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" height=1000 width=1000>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function drawline()
{
  var c=document.getElementById("myCanvas");    //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");                   //获取 canvas 对象的上下文
  var dx=150;
  var dy=150;
  var s = 100;
```

```
    ctx.beginPath();                                    //开始绘图路径
    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);
      //用三角函数计算顶点
       ctx.lineTo(dx+x*s,dy+y*s);
    }
    ctx.closePath();
    ctx.stroke();
}
window.addEventListener("load", drawline, true);
</script>
</body>
</html>
```

浏览例 5-2 的结果如图 5-3 所示。

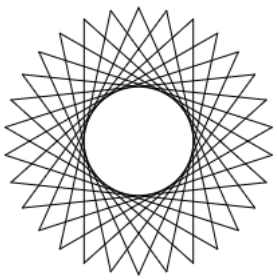

图 5-3　Canvas 绘制复杂图形

5.3.2　绘制矩形

可以通过调用 rect()、strokeRect()、fillRect()和 clearRect()等 4 个 API 方法在 Canvas 画布中绘制矩形。其中，前两个方法用于绘制矩形边框，调用 fillRect()可以填充指定的矩形区域，调用 clearRect ()可以擦除指定的矩形区域。

1．rect()

rect()用于创建矩形，rect()方法的语法如下：

```
rect (x, y, width, height)
```

参数说明如下：

x 是矩形的左上角的 X 坐标；y 是矩形的左上角的 Y 坐标；width 是矩形的宽度；height 是矩形的高度。

【例 5-3】 使用 rect()方法绘制矩形边框的例子。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
```

```
<script type="text/javascript">
function drawRect()
{
  var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");   //获取 canvas 对象的上下文
  ctx.beginPath();              // 开始绘图路径，绘制起始点
  ctx.rect(20,20, 100, 50);
  ctx.stroke();                 //通过线条绘制轮廓（边框）
}
window.addEventListener("load", drawRect, true);
</script>
```

2．strokeRect()

strokeRect()绘制矩形（无填充），strokeRect()方法的语法如下：

```
strokeRect(x, y, width, height)
```

参数的含义与 rect()方法的参数相同。strokeRect()方法与 rect()方法的区别在于调用 strokeRect()方法时不需要使用 beginPath()和 stroke()即可绘图。

3．fillRect() 和 clearRect()

fillRect()绘制“被填充”的矩形，fillRect()方法的语法如下：

```
fillRect(x, y, width, height)
```

参数的含义与 rect()方法的参数相同。

clearRect()清除给定矩形内的图像：

```
clearRect(x, y, width, height)
```

参数的含义与 rect()方法的参数相同。

【例 5-4】 Canvas 绘制一个矩形和一个填充矩形的例子。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="demoCanvas" width="500" height="500">您的浏览器不支持 canvas。
</canvas>
<!---下面将演示一种绘制矩形的 demo--->
<script type="text/javascript">
       var c = document.getElementById("demoCanvas"); //获取网页中的 canvas 对象
       var context = c.getContext('2d');        //获取上下文
       context.strokeStyle = "red";             //指定绘制线样式、颜色
       context.strokeRect(10, 10, 190, 100);    //绘制矩形线条，内容是空的
       //以下填充矩形
       context.fillStyle = "blue";
       context.fillRect(110,110,100,100);       //绘制填充矩形
</script>
</body>
```

5.3.3　绘制圆弧

可以调用 arc()方法绘制圆弧，语法如下：

```
arc(centerX, centerY, radius, startingAngle, endingAngle, antiClockwise);
```

参数说明如下：

centerX，圆弧圆心的 X 坐标；centerY，圆弧圆心的 Y 坐标；radius，圆弧的半径；startingAngle，圆弧的起始角度；endingAngle，圆弧的结束角度；antiClockwise，是否按逆时针方向绘图。

例如：使用 arc()方法绘制圆心为（50，50），半径为 100 的圆弧。圆弧的起始角度为 60°，圆弧的结束角度为 180°。

```
ctx.beginPath();  // 开始绘图路径
ctx.arc(50, 50, 100, 1/3 * Math.PI, 1 * Math.PI, false);
ctx.stroke();
```

【例 5-5】 使用 arc()方法画圆的例子。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
  var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");   //获取 canvas 对象的上下文
  var radius = 100;
  var startingAngle = 0;
  var endingAngle = 2 * Math.PI;
  ctx.beginPath();              // 开始绘图路径
  ctx.arc(150, 150, radius, startingAngle, endingAngle, false);
  ctx.stroke();
}
window.addEventListener("load", draw, true);
</script>
```

5.4　描边和填充

5.4.1　描边

通过设置 Canvas 的上下文 2D 对象的 strokeStyle 属性可以指定描边的颜色，通过设置上下文 2D 对象的 lineWidth 属性可以指定描边的宽度。

例如：通过设置描边颜色和宽度来绘制红色线条宽度为 10 的圆。

```
ctx.lineWidth = 10;             //描边宽度为 10
ctx.strokeStyle = "red";        //描边颜色红色
ctx.arc(50, 50, 100, 0, 2 * Math.PI, false);
ctx.stroke();
```

5.4.2 填充图形内部

通过设置 Canvas 的上下文 2D 对象的 fillStyle 属性可以指定填充图形内部的颜色。

【例 5-6】 填充图形内部的例子。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
  var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");        //获取 canvas 对象的上下文
  ctx.fillStyle = "yellow";          //填充图形内部的颜色为黄色
  ctx.fillRect(65,65, 100, 100);     //矩形的宽度和高度为 100，内部填充黄色
}
window.addEventListener("load", draw, true);
</script>
```

5.4.3 渐变颜色

1．创建 CanvasGradient 对象

CanvasGradient 是用于定义画布中的一个渐变颜色的对象。如果要使用渐变颜色，首先需要创建一个 CanvasGradient 对象。可以通过下面两种方法创建 CanvasGradient 对象。

（1）以线性颜色渐变方式创建 CanvasGradient 对象

使用 Canvas 的上下文 2D 对象 createLinearGradient()方法可以线性颜色渐变方式创建 CanvasGradient 对象。createLinearGradient() 方法的语法如下：

```
createLinearGradient(xStart, yStart, xEnd, yEnd)
```

参数 xStart 和 yStart 是渐变的起始点的坐标，参数 xEnd 和 yEnd 是渐变的结束点的坐标。例如：

```
var g1 = ctx.createLinearGradient(0, 0, 300, 100);
```

（2）以放射颜色渐变方式创建 CanvasGradient 对象

使用 Canvas 的上下文 2D 对象 createRadialGradient()方法可以放射颜色渐变方式创建 CanvasGradient 对象。createRadialGradient()方法的语法如下：

```
createRadialGradient(xStart, yStart, radiusStart, xEnd, yEnd, radiusEnd)
```

参数 xStart 和 yStart 是开始圆的圆心的坐标，radiusStart 是开始圆的半径；参数 xEnd 和 yEnd 是结束圆的圆心的坐标，radiusEnd 是结束圆的半径。

2．为渐变对象设置颜色

创建 CanvasGradient 对象后，还需要为其设置颜色基准，可以通过 CanvasGradient 对象的 addColorStop()方法在渐变中的某一点添加一个颜色变化。渐变中其他点的颜色将以此为基准。addColorStop()方法的语法如下：

```
addColorStop(offset, color)
```

参数 offset 是一个范围在 0.0 到 1.0 之间的浮点值，表示渐变的开始点和结束点之间的一部分。offset 为 0 对应开始点，offset 为 1 对应结束点。Color 指定 offset 显示的颜色。沿着渐变某一点的颜色是根据这个值以及任何其他的颜色色标来插值的。

```
var canvas=document.getElementById("myCanvas");//获取网页中的 canvas 对象
var ctx = canvas.getContext('2d');
var g1 = ctx.createLinearGradient(0, 0, 300, 100);
g1.addColorStop(0, 'rgb(0,0,255)');              //蓝
g1.addColorStop(0.4, 'rgb(255,255,255)');        //白
g1.addColorStop(1, 'rgb(255,0,0)');              //红
```

程序代码的示意图如图 5-4 所示。

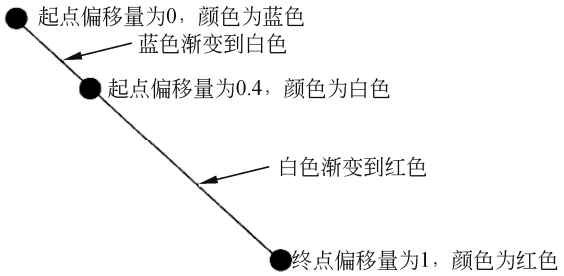

图 5-4　程序代码的运行结果

3．设置描边样式为渐变颜色

只要将前面创建的 CanvasGradient 对象赋值给用于绘图的 Canvas 的上下文 2D 对象的 strokeStyle 属性，即可使用渐变颜色进行描边。例如：

```
var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
var ctx=c.getContext("2d");  //获取 canvas 对象的上下文
var Colordiagonal = ctx.createLinearGradient(10,10, 100,10);
ctx.strokeStyle = Colordiagonal;
ctx.stroke();  //关闭绘图路径
```

【例 5-7】 使用黄、绿、红的放射渐变颜色填充一个圆。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
  var c=document.getElementById("myCanvas");    //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");                   //获取 canvas 对象的上下文
  //对角线上的渐变
  var Colordiagonal = ctx.createRadialGradient(100,100, 0, 100,100, 100);
  Colordiagonal.addColorStop(0, "red");
  Colordiagonal.addColorStop(0.5, "green");
  Colordiagonal.addColorStop(1, "yellow");
```

```
    var centerX = 100;
    var centerY = 100;
    var radius = 100;
    var startingAngle = 0;
    var endingAngle = 2 * Math.PI;
    ctx.beginPath();  // 开始绘图路径
    ctx.arc(centerX, centerY, radius, startingAngle, endingAngle, false);
    ctx.fillStyle = Colordiagonal;
    ctx.stroke();
    ctx.fill();
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-7 的结果如图 5-5 所示。

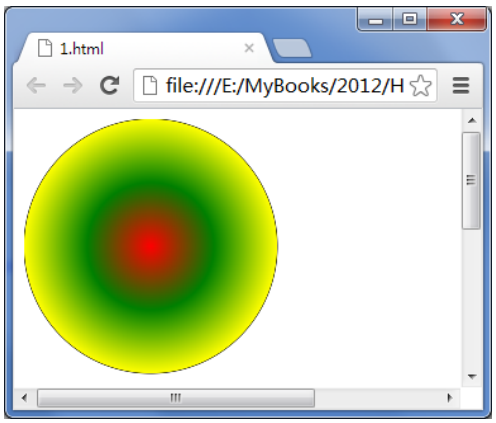

图 5-5　使用黄、绿、红的放射渐变颜色填充一个圆

5.4.4　透明颜色

在指定颜色时，可以使用 rgba()方法定义透明颜色，格式如下：

```
rgba(r,g,b, alpha)
```

其中 r 表示红色集合，g 表示绿色集合，b 表示蓝色集合。r、g、b 都是十进制数，取值范围为 0～255。alpha 的取值范围为 0～1，用于指定透明度，0 表示完全透明，1 表示不透明。

【例 5-8】 使用透明颜色填充 10 个连串的圆，模拟太阳光照射的光环。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
   var canvas=document.getElementById("myCanvas");
   if(canvas == null)
```

```
        return false;
    var context = canvas.getContext("2d");
     //先绘制画布的底图
     context.fillStyle="yellow";
     context.fillRect(0,0,400,350);
     //用循环绘制 10 个圆形
     var n = 0;
     for(var i=0 ;i<10;i++){
         //开始创建路径，因为圆本质上也是一个路径，这里向 canvas 说明要开始画了，这是起点
        context.beginPath();
        context.arc(i*25,i*25,i*10,0,Math.PI*2,true);
        context.fillStyle="rgba(255,0,0,0.25)";
        context.fill();        //填充刚才所画的圆形
     }
  }
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-8 的结果如图 5-6 所示。

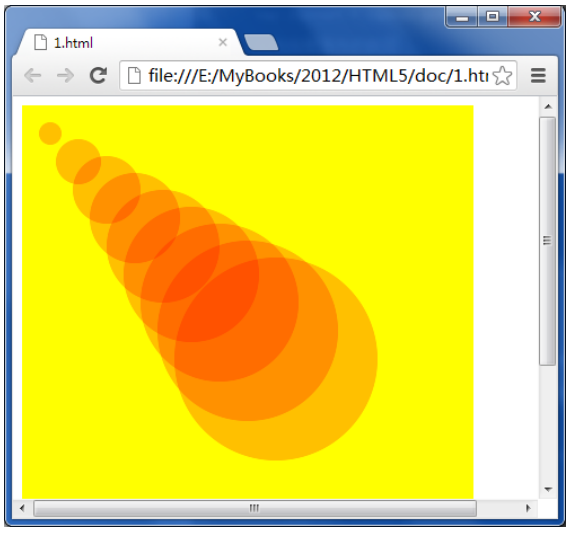

图 5-6　透明颜色填充 10 个连串的圆

5.5 绘制图像与文字

5.5.1 绘制图像

在画布上绘制图像的 Canvas API 是 drawImage()，语法如下：

```
drawImage(image, x, y)
drawImage(image, x, y, width, height)
drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight,destX, destY,
```

```
    destWidth, destHeight)
```

参数说明：

image，所要绘制的图像，必须是表示<img>标记或者图像文件的 Image 对象，或者是 Canvas 元素。

x 和 y，要绘制的图像的左上角位置；width 和 height，绘制图像的宽度和高度。

sourceX 和 sourceY，图像将要被绘制的区域的左上角；destX 和 destY，所要绘制的图像区域的左上角的画布坐标；sourceWidth 和 sourceHeight，被绘制的原图像区域；destWidth 和 destHeight，图像区域在画布上要绘制成的大小。

【例 5-9】 不同形式显示一本图书封面 cover.jpg。

```
<canvas id="myCanvas" height=1000 width=1000>您的浏览器不支持 canvas。
</canvas>
<script type="text/javascript">
function draw()
{
  var c=document.getElementById("myCanvas");     //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");                    //获取 canvas 对象的上下文
  var imageObj = new Image();                    //创建图像对象
  imageObj.src = "cover.jpg";
  imageObj.onload = function(){
      ctx.drawImage(imageObj, 0, 0);             //原图大小显示
      ctx.drawImage(imageObj, 250, 0, 120, 160);//原图一半大小显示
//从原图(0,100)位置开始截取中间一块宽 240*高 160 的区域，原大小显示在屏幕(400,0)处
      ctx.drawImage(imageObj, 0, 100, 240, 160, 400, 0, 240, 160);
   };
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-9 的结果如图 5-7 所示。

图 5-7　不同形式显示一本图书的封面

5.5.2　组合图形

在绘制图形时，如果画布上已经有图形，就涉及一个问题：两个图形如何组合。可以通过 Canvas 的上下文 2D 对象的 globalCompositeOperation 属性来设置组合方式。globalCompositeOperation 属性的可选值如表 5-2 所示。

表 5-2　globalCompositeOperation 属性可选值

可选值	描述
source-over	默认值，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
source-in	新图形会仅仅出现与原有内容重叠的部分，其他区域都变成透明的
destination-in	原有内容中与新图形重叠的部分会被保留，其他区域都变成透明的
source-out	只有新图形中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，并会在原有内容之下绘制新图形
lighter	两图形中重叠部分作加色处理
darker	两图形中重叠部分作减色处理
xor	重叠的部分会变成透明
copy	只有新图形会被保留，其他都被清除掉

【例 5-10】 一个矩形和圆的重叠效果。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
  var c=document.getElementById("myCanvas");    //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");                   //获取 canvas 对象的上下文
  ctx.fillStyle = "blue";
  ctx.fillRect(0,0, 100, 100);                  //填充蓝色的矩形
  ctx.fillStyle = "red";
  ctx.globalCompositeOperation = "source-over";
  var centerX = 100;
  var centerY = 100;
  var radius = 50;
  var startingAngle = 0;
  var endingAngle = 2 * Math.PI;
  ctx.beginPath();                              //开始绘图路径
  ctx.arc(centerX, centerY, radius, startingAngle, endingAngle, false);
                                                //绘制圆
  ctx.fill();
}
window.addEventListener("load", draw, true);
</script>
```

例 5-10 中蓝色正方形先画，红色圆形后画，source-over 取值效果如图 5-8 所示。

图 5-8　source-over 取值效果

其余取值效果如图 5-9 所示。

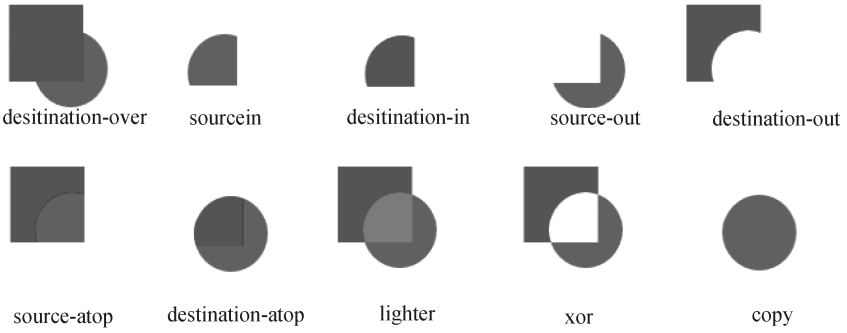

图 5-9　globalCompositeOperation 属性的不同值效果

5.5.3　输出文字

可以使用 strokeText()方法在画布的指定位置输出文字，语法如下：

```
strokeText(string text, float x, float y)
```

参数说明如下：

string 为所要输出的字符串；x 和 y 是要输出的字符串的位置坐标。

例如：

```
var c=document.getElementById("myCanvas"); //获取网页中的 canvas 对象
var ctx=c.getContext("2d");                 //获取 canvas 对象的上下文
ctx.strokeText("中原工学院", 100, 100);      //在(100, 100)处显示"中原工学院"
```

1．设置字体

可以通过 Context.font 属性来设置输出字符串的字体，格式如下：

Context.font = "字体大小字体名称"

例如：

```
var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
var ctx=c.getContext("2d");  //获取 canvas 对象的上下文
ctx.font = "40 隶书";
ctx.strokeText("中原工学院", 100, 100);
```

2．设置对齐方式

可以通过 Context.TextAlign 属性来设置输出字符串的对齐方式。可选值为"left"（左对

齐)、"center"（居中对齐）和"right"（右对齐）。

例如：

```
ctx.TextAlign = "center";
```

3．设置边框宽度和颜色

可以通过设置 Canvas 的上下文 2D 对象的 strokeStyle 属性指定输出文字的颜色。

```
ctx.strokeStyle = "blue";
ctx.font = "40pt 隶书";
ctx.strokeText("中原工学院", 100, 100);
```

4．填充字体内部

使用 strokeText()方法输出的文字是中空的，只绘制了边框。如果要填充文字内部，可以使用 fillText()方法，语法如下：

```
fillText (string text, float x, float y)
```

可以使用 Context.fillStyle 属性指定填充的颜色。

```
ctx.fillStyle = "blue";
```

【例 5-11】 渐变填充文字。

```
<canvas id="myCanvas" height=500 width=500>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw(){
  var c=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var ctx=c.getContext("2d");  //获取 canvas 对象的上下文
  var Colordiagonal = ctx.createLinearGradient(100,100, 300,100);
  Colordiagonal.addColorStop(0, "yellow");
  Colordiagonal.addColorStop(0.5, "green");
  Colordiagonal.addColorStop(1, "red");
  ctx.fillStyle = Colordiagonal;
  ctx.font = "60pt 隶书";
  ctx.fillText("中原工学院", 100, 100);
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-11 的结果如图 5-10 所示。

图 5-10　渐变填充文字

5.6 图形的操作

5.6.1 保存和恢复绘图状态

调用 Context.save()方法可以保存当前的绘图状态。Canvas 状态是以堆（stack）的方式保存绘图状态的，绘图状态包括：

- 当前应用的操作（比如移动、旋转、缩放或变形，具体方法将在本节稍后介绍）。
- strokeStyle、fillStyle、globalAlpha、lineWidth、lineCap、lineJoin、miterLimit、shadowOffsetX、shadowOffsetY、shadowBlur、shadowColor、globalCompositeOperation 等属性的值。
- 当前的裁切路径（clipping path）。

调用 Context.restoe()方法可以从堆中弹出之前保存的绘图状态。

Context.save()方法和 Context.restoe()方法都没有参数。

【例 5-12】 保存和恢复绘图状态。

```
<canvas id="myCanvas" height=500 width=500></canvas>
<script type="text/javascript">
function draw() {
var ctx = document.getElementById('myCanvas').getContext('2d');
ctx.fillStyle = 'red'
ctx.fillRect(0,0,150,150);  //使用红色填充矩形
ctx.save();                 //保存当前的绘图状态
ctx.fillStyle = 'green'
ctx.fillRect(45,45,60,60);  //使用绿色填充矩形
ctx.restore();              // 恢复之前保存的绘图状态，即 ctx.fillStyle = 'red'
ctx.fillRect(60,60,30,30);  // 使用红色填充矩形
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-12 的结果如图 5-11 所示。

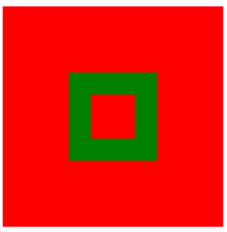

图 5-11 渐变填充文字

5.6.2 图形的变换

1．平移 translate(x,y)

参数 x 是坐标原点向 x 轴方向平移的位移，参数 y 是坐标原点向 y 轴方向平移的位移。

2．缩放 scale(x,y)

参数 x 是 x 坐标轴缩放比例，参数 y 是 y 坐标轴缩放比例。

3．旋转 rotate(angle)

参数 angle 是坐标轴旋转的角度（角度变化模型和画圆的模型一样）。

4．变形 setTransform()

可以调用 setTransform()方法对绘制的 canvas 图形进行变形，语法如下：

```
context.setTransform(m11, m12, m21, m22, dx, dy);
```

假定点(x, y)经过变形后变成了(X, Y)，则变形的转换公式如下：

$X = m11 \times x + m21 \times y + dx$

$Y = m12 \times x + m22 \times y + dy$

【例 5-13】 图形的变换例子。

```
<canvas id="myCanvas" height=250 width=250>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw(){
  var canvas=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var context = canvas.getContext("2d");           //获取 canvas 对象的上下文
  context.save();                                  //保存了当前 context 的状态
  context.fillStyle = "#EEEEFF";
  context.fillRect(0, 0, 400, 300);
  context.fillStyle = "rgba(255,0,0,0.1)";
  context.fillRect(0, 0, 100, 100);//正方形
  //平移 1 缩放 2 旋转 3
  context.translate(100, 100);          //坐标原点平移(100, 100)
  context.scale(0.5, 0.5);              //x,y 轴是原来一半
  context.rotate(Math.PI / 4);          //旋转 45 度
  context.fillRect(0, 0, 100, 100);     //平移缩放旋转后的正方形
  context.restore();                    //恢复之前保存的绘图状态
  context.beginPath();                  //开始绘图路径
  context.arc(200, 50, 50, 0,  2 * Math.PI, false); //绘制圆
  context.stroke();
  context.fill();
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-13 的结果如图 5-12 所示。

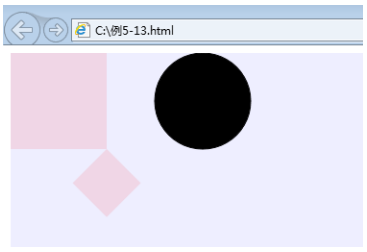

图 5-12　渐变填充文字

5.7 HTML5 Canvas 动画实例

在开发在线游戏时，绘制动画是非常重要的。本节介绍一个使用 Canvas API 实现的动画实例——游戏人物的跑步动画。

5.7.1 动画的概念及原理

1．动画

动画是通过一幅幅静止的，内容不同的画面（即帧）快速播放使人们在视觉上产生运动的感觉。这是利用了人类眼睛的视觉暂留原理。利用人的这种生理特性可制作出具有高度想象力和表现力的动画影片。

2．原理

人们在看画面时，画面会在大脑视觉神经中停留大约 1/24 秒，如果每秒更替 24 个画面或更多，那么前一个画面还没在人脑中消失之前，下一个画面进入人脑，人们就会觉得画面动起来了，它的基本原理与电影、电视一样，都是视觉原理。

在计算机上要实现动画效果，除了绘图外，还需要解决下面两个问题：

（1）定期绘图，也就是每隔一段时间就调用绘图函数进行绘图。动画是通过多次绘图实现的，一次绘图只能实现静态图像。

可以使用 setInterval()方法设置一个定时器，语法如下：

```
setInterval(函数名,时间间隔)
```

时间间隔的单位是毫秒（ms），每经过指定的时间间隔系统都会自动调用指定的函数完成绘画。

（2）清除先前绘制的所有图形。物体已经移动开来，可原来的位置上还保留先前绘制的图形，这样当然不行。解决这个问题最简单的方法是使用 clearRect(x, y, width, height)方法清除画布中指定区域的内容。

图 5-13 是一个方向（一般都是 4 个方向）的跑步动作序列图。假如想获取一个姿态的位图，可利用 Canvas 的上下文 2D 对象的 drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight,destX, destY, destWidth, destHeight)将源位图上某个区域（sourceX, sourceY, sourceWidth, sourceHeight）拷贝到目标区域的（destX, destY）坐标点处，显示大小为（宽 destWidth, 高 destHeight）。

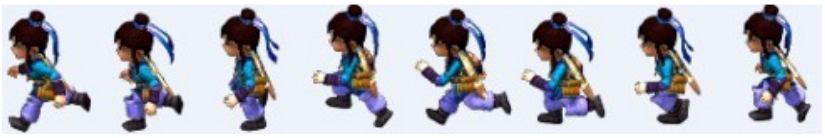

图 5-13 一个方向的跑步动作序列

【例 5-14】 实现从跑步动作序列 Snap1.jpg 文件中截取的第 3 个动作（帧）。

分析：在 Snap1.jpg 文件中，每个人物动作的大小为 60 像素×80 像素，所以截取源位图的 sourceX=120，sourceY=0，sourceWidth=60, sourceHeight=80 就是第 3 个动作（帧）。

```
<canvas id="myCanvas" height=250 width=250>您的浏览器不支持 canvas。</canvas>
<script type="text/javascript">
function draw()
{
  var canvas=document.getElementById("myCanvas");  //获取网页中的 canvas 对象
  var context = canvas.getContext("2d");    //获取 canvas 对象的上下文
  var imageObj = new Image();               //创建图像对象
  imageObj.src = "Snap1.jpg";
  imageObj.onload = function(){
      //从原图(120, 0)位置开始截取中间一块宽 60*高 80 的区域，原大小显示在屏幕(0,0)处
      ctx.drawImage(imageObj, 120, 0, 60, 80, 0, 0, 60, 80);
     };
}
window.addEventListener("load", draw, true);
</script>
```

浏览例 5-14 的结果如图 5-14 所示，在页面上仅仅显示第 3 个动作。

图 5-14　静态显示第 3 个动作

5.7.2　游戏人物的跑步动画

【例 5-15】 实现游戏人物的跑步动画。

首先定义一个 Canvas 元素，画布的长和宽都是 300，代码如下：

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Canvas 实现游戏人物的跑步动画</title>
</head>
<body>
<canvas id="canvasId" width="300" height="300"></canvas>
</body>
</html>
```

在 JavaScript 代码中定义一个 Image 对象，用于显示 Snap1.jpg。然后定义一个 init() 函数，初始化 Image 对象，并设置定时器，代码如下：

```
<script type="text/javascript">
    var imageObj = new Image();
    var x =300;
    var n =0;                          //计数器
```

```
    function init(){
      imageObj.src = 'Snap1.jpg';
      imageObj.onload = function(){  //图片加载成功
        setInterval(draw,100);       //定时器，每 0.1 秒执行一次 draw()函数
    };
//此处省略 draw()函数的代码
    window.addEventListener("load", init, true);
</script>
```

使用了定时器，每隔 100 毫秒就会在 Snap1.jpg 图片截取一张 60 像素×80 像素大小的小图并绘制出来，且每次向左移 15 像素，直到最左端时重新从右侧开始，不停循环，就可见游戏人物在屏幕上不停地奔跑。

下面分析 draw()函数的实现。例 5-14 中仅仅显示人物的第三个动作，而为了实现动画需要 clearRect(x, y, width, height)不断清除先前绘制的动作图形，再绘制后续的动作。所以需要一个计数器 n，记录当前绘制到第几动作（帧）了。

```
function draw()
{
      var canvas=document.getElementById("myCanvas"); //获取网页中的 canvas
                                                        对象
      var ctx = canvas.getContext("2d");    //获取 canvas 对象的上下文
      ctx.clearRect(0,0,300,300);           //清除 canvas 画布
      //从原图(60*n)位置开始截取中间一块宽 60*高 80 的区域，显示在屏幕(x,0)处
      ctx.drawImage(imageObj, 60*n, 0, 60, 80, x, 0, 60, 80);
      if(n>=8){
        n=0;
      }else{
        n++;
      }
      if(x>=0){
        x=x-30;                             //前移 30 像素
      }else{
        x=300;                              //回到右侧
      }
}
```

浏览例 5-15 的结果是一个游戏人物不停重复地从右侧跑到左侧的动画。