第7讲

Canvas游戏

信息学院 孙辉

内容

- Canvas基础
- Canvas基本动画

Canvas基础

- canvas元素是HTML5中新增的一个重要元素，专门用来绘制图形。
- 在页面上放置一个canvas元素就相当于在页面上放置了一块“画布”，它是一个矩形区域，您可以控制其每一像素。可以在其中进行图形的描绘。
- canvas 拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。
- canvas 元素本身是没有绘图能力的。所有的绘制工作必须在 JavaScript 内部完成

Canvas元素的基础知识

- <canvas>看起来很像<img>，唯一不同就是它不含 src 和 alt 属性。
- 它只有两个属性，width 和 height，两个都是可选的，并且都可以 DOM 或者 CSS 来设置。
- 如果不指定width 和 height，默认的是宽300像素，高150像素。
- 虽然可以通过 CSS 来调整canvas的大小，但渲染图像会缩放来适应布局的 。
- 结束标签 </canvas> 是必须的

放置Canvas

```
<style type="text/css">
.canvas {  width:150px; height:150px;}
</style>

<body onload="draw();">
   <div class="canvas">
          <canvas id="canvas" width="150" height="150">
             <p>写在这里面的内容将展示给不兼容canvas的浏览器
  </p>
          </canvas>
  </div>
 </body>
```

演示

创建canvans元素

- Canvan标签
  - 规定元素的id、宽度和高度

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

  - 默认宽300像素，高150像素

- 当web浏览器不支持Canvans时的反馈信息

```
<canvas id="canvas" width=“200" height="100">
        写在这里面的内容将展示给不兼容canvas的浏览器
</canvas>
```

实例：在canvas中绘制矩形

```
<body>
<canvas id="myCanvas">
        your browser does not support the canvas tag
</canvas>

<script type="text/javascript">

var canvas=document.getElementById('myCanvas');
var ctx=canvas.getContext('2d');
ctx.fillStyle='#FF0000';
ctx.fillRect(0,0,80,100);
</script>
</body>
```

得到画布

创建 context 对象；
getContext("2d") 对象是内建的 HTML5 对象，拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法

fillStyle 方法将context对象染成红色，
fillRect 方法规定了形状、位置和尺寸。

fillRect 方法拥有参数 (0,0,80,100)。
意思是：在画布上绘制 80x100 的矩形，从左上角开始 (0,0)。

理解坐标

HTML DOM getContext() 方法

定义和用法

getContext() 方法返回一个用于在画布上绘图的环境。

语法

Canvas.getContext(*contextID*)

参数

指定了您想要在画布上绘制的类型。当前唯一的合法值是"2d"，它指定了二维绘图，并且导致这个方法返回一个环境对象，该对象导出一个二维绘图 API。

提示: 未来如果 <canvas> 标签扩展到支持3D绘图，getContext()方法可能允许传递一个"3d"字符串参数。

返回值

一个 CanvasRenderingContext2D 对象，用它可以绘制到 Canvas 元素中，实现了一个画布所使用的大多数方法

设置绘图样式

- 填充样式
  - fillStyle() 颜色值支持半透明
- 描边样式
  - strokeStyle() 颜色值支持半透明
- 设定线条样式
  - 设定线宽 lineWidth()
  - 设定图线帽样式 lineCap

    'butt','round','square'
  - 线的链接样式 lineJoin

    'round','bevel',**'miter'**
- 绘制渐变
  - 绘制线性渐变
  - 绘制径向渐变

颜色、样式和阴影

颜色、样式和阴影

属性	描述
fillStyle	设置或返回用于填充绘画的颜色、渐变或模式
strokeStyle	设置或返回用于笔触的颜色、渐变或模式
shadowColor	设置或返回用于阴影的颜色
shadowBlur	设置或返回用于阴影的模糊级别
shadowOffsetX	设置或返回阴影距形状的水平距离
shadowOffsetY	设置或返回阴影距形状的垂直距离

线条样式

线条样式

属性	描述
lineCap	设置或返回线条的结束端点样式
lineJoin	设置或返回两条线相交时，所创建的拐角类型
lineWidth	设置或返回当前的线条宽度
miterLimit	设置或返回最大斜接长度

矩形

矩形

方法	描述
rect()	创建矩形
fillRect()	绘制"被填充"的矩形
strokeRect()	绘制矩形（无填充）
clearRect()	在给定的矩形内清除指定的像素

路径

方法	描述
fill()	填充当前绘图（路径）
stroke()	绘制已定义的路径
beginPath()	起始一条路径，或重置当前路径
moveTo()	把路径移动到画布中的指定点，不创建线条
closePath()	创建从当前点回到起始点的路径
lineTo()	添加一个新点，然后在画布中创建从该点到最后指定点的线条
clip()	从原始画布剪切任意形状和尺寸的区域
quadraticCurveTo()	创建二次贝塞尔曲线
bezierCurveTo()	创建三次方贝塞尔曲线
arc()	创建弧/曲线（用于创建圆形或部分圆）
arcTo()	创建两切线之间的弧/曲线
isPointInPath()	如果指定的点位于当前路径中，则返回 true，否则返回 false

举例：绘制带阴影的心型

- shadowOffsetX——阴影的横向移动
- shadowOffsetY——阴影的纵向移动
- shadowColor——阴影的颜色
- shadowBlur——阴影的模糊范围

```
ctx.strokeStyle="rgb(200,0,0)";
ctx.fillStyle="rgb(255,0,0)";
ctx.shadowOffsetX=3;
ctx.shadowOffsetY=3;
ctx.shadowColor="rgba(150,0,0,0.2)";
ctx.shadowBlur=3;
ctx.beginPath();
ctx.moveTo(75,60);
ctx.bezierCurveTo(125,20,125,100,75,115);
ctx.bezierCurveTo(25,100,25,20,75,60);
ctx.stroke();
ctx.fill();
```

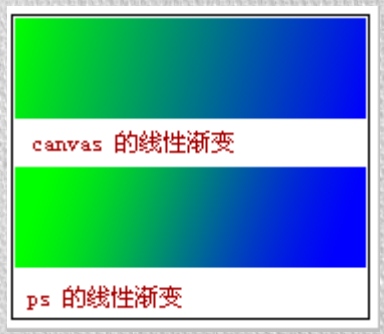
渐变效果

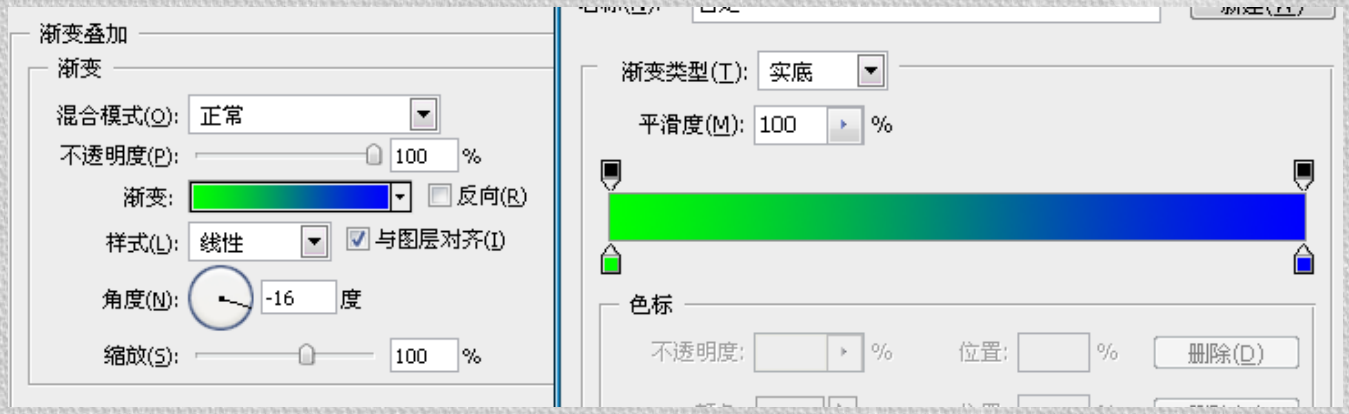
方法	描述
createLinearGradient()	创建线性渐变（用在画布内容上）
createPattern()	在指定的方向上重复指定的元素
createRadialGradient()	创建放射状/环形的渐变（用在画布内容上）
addColorStop()	规定渐变对象中的颜色和停止位置

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

创建并返回了一个新的 CanvasGradient 对象，它在指定的起始点和结束点之间线性地内插颜色值。注意，这个方法并没有为渐变指定任何颜色。使用返回对象的 addColorStop() 来做到这一点。要使用一个渐变来勾勒线条或填充区域，只需要把 CanvasGradient 对象赋给 strokeStyle 属性或 fillStyle 属性即可。

canvas线性渐变示例






```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createLinearGradient(1,1,175,50);
//(开始坐标x，开始坐标y，结束点X，结束点Y)
grd.addColorStop(1,"#0000ff");//addColorStop(stop, color)/*参数：百分比，颜色值*/
grd.addColorStop(0,"#00FF00");
ctx.fillStyle=grd;
ctx.fillRect(1,1,175,50);
```

通过JS在画布上绘制图形

- 使用路径
  - beginPath() stopPath()
  - moveTo() lineTo()
  - 绘制矩形 Rect() fillRect() strokeRect() clearRect()
  - 绘制圆形 arc()
  - 绘制贝塞尔曲线
    - **quadraticCurveTo()**
    - **bezierCurveTo()**
  - Fill() strock()

开始与闭合路径

- beginPath()
  - 起始一条路径，或重置当前路径
  - 该方法不使用参数。通过调用该方法，开始路径的创建。
  - 在几次循环地创建路径的过程中，每次开始创建时都要调用beginPath函数
- closePath()
  - 如果当前子路径是打开的，就关闭它。
  - 路径创建完成后，使用图形上下文对图像的closePath方法将路径关闭。将路径关闭后，路径的创建工作就完成了，但是注意，这时只是路径创建完毕而已，还没有真正的绘制任何图形。

moveTo、lineTo

- moveTo
  - 把路径移动到画布中的指定点，不创建线条

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.moveTo(0,0);
ctx.lineTo(300,150);
ctx.stroke();
```

HTML DOM rect() 方法

定义和用法

- rect() 方法为当前路径添加一条矩形子路径。

语法

- quadraticCurveTo(x, y, *width*, *height*)
- **参数描述**x, y矩形的左上角的坐标。width, height矩形的大小。

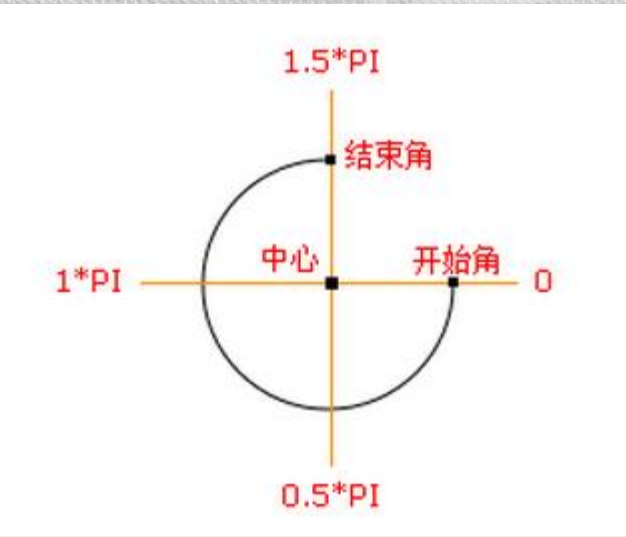
描述

- rect() 方法为路径添加了一个矩形。这个矩形是路径的一个子路径并且没有和路径中的任何其他子路径相连。

HTML DOM arc() 方法

- 为一个画布的当前子路径添加一条弧。

  context.arc(x,y,r,sAngle,eAngle,counterclockwise);

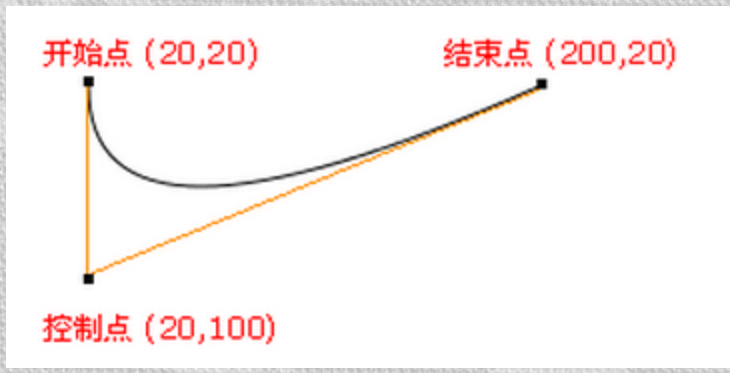


参数	描述
x	圆的中心的 x 坐标。
y	圆的中心的 y 坐标。
r	圆的半径。
sAngle	起始角，以弧度计。（弧的圆形的三点钟位置是 0 度）。
eAngle	结束角，以弧度计。
counterclockwise	可选。规定应该逆时针还是顺时针绘图。False = 顺时针，true = 逆时针。

quadraticCurveTo() 方法

- 为当前路径添加一条贝塞尔曲线
  - quadraticCurveTo(*cpX*, *cpY*, *x*, *y*)

参数	描述
cpx	贝塞尔控制点的 x 坐标
cpy	贝塞尔控制点的 y 坐标
x	结束点的 x 坐标
y	结束点的 y 坐标



提示：二次贝塞尔曲线需要两个点。第一个点是用于二次贝塞尔计算中的控制点，第二个点是曲线的结束点。曲线的开始点是当前路径中最后一个点。如果路径不存在，那么请使用 beginPath()和moveTo() 方法来定义开始点。

HTML DOM bezierCurveTo() 方法

- 定义和用法

  bezierCurveTo() 方法在一个画布中开始子路径的一个新的集合。

- 语法：bezierCurveTo(*cpX1*, *cpY1*, *cpX2*, *cpY2*, *x*, *y*)

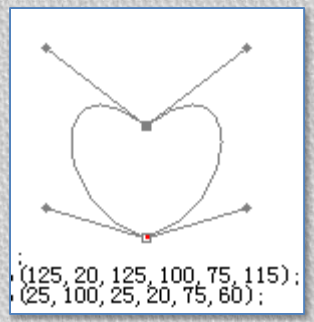
- bezierCurveTo() 为一个画布的当前子路径添加一条三次贝塞尔曲线。这条曲线的开始点是画布的当前点，而结束点是 (*x*, *y*)。两条贝塞尔曲线控制点 (*cpX1*, *cpY1*) 和 (*cpX2*, *cpY2*) 定义了曲线的形状。当这个方法返回的时候，当前的位置为 (*x*, *y*)。

参数	描述
cpX1, cpY1	和曲线的开始点（当前位置）相关联的控制点的坐标。
cpX2, cpY2	和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。

示例在下页

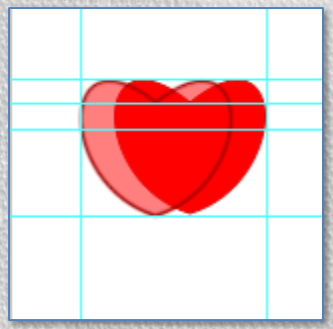
二维贝塞尔曲线示例

Js写出来的



PS下的路径

PS填充描边



两者可以完全重合

```
ctx.strokeStyle="rgb(200,0,0)";  //描边颜色
ctx.fillStyle="rgb(255,0,0)";// 填充颜色
ctx.beginPath();//开始路径

ctx.moveTo(75,60);//曲线起点
ctx.bezierCurveTo(125,20,125,100,75,115);// 右边曲线，（75,115）将作为下条曲线起点
ctx.bezierCurveTo(25,100,25,20,75,60);//  两条曲线的 cp点其实是对称的，合并终点。
ctx.closePath();//闭合路径
ctx.stroke();//描边
ctx.fill();//填充
```

在Canvas中引入图像

- canvas绘制图像基础
- drawImage方法（三种用法）
- 图像平铺createPattern方法

canvas绘制图像

Canvas 相当有趣的一项功能就是可以引入图像，它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（标准说明中并没有包含绘制文字的功能）。只要是 Gecko 支持的图像（如 PNG，GIF，JPEG等）都可以引入到 canvas 中，而且其它的 canvas 元素也可以作为图像的来源。

在 Canvas API 中，图像通过表示 HTML <img> 元素的 Image 对象来指定，或者通过使用 Image() 构造函数所创建的屏幕外图像来指定。一个 Canvas 对象也可以用作图像来源。

可以使用 drawImage() 方法在一个画布上绘制图像；而更为常见的形式是，允许源图像的任意矩形区域缩放或绘制到画布上。

drawImage()

- 3参数 ctx.drawImage(image,x,y)

  其中第一个参数可以使一个IMG元素，可以是一个video元素，或者JS创建的Image对象。XY分别表示图像在画布中的起始点坐标。

- 5参数 ctx.drawImage(image,x,y,w,h)

  由于新引入了两个参数，这种方法可以对图像进行缩放。

- 9参数 ctx.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)

  这种方法可以进行裁切，我们把图像从（sx，sy）开始宽高为sw，sh的这一部分像素放置到画布的（dx,dy）处，宽高分别为dw，dh。

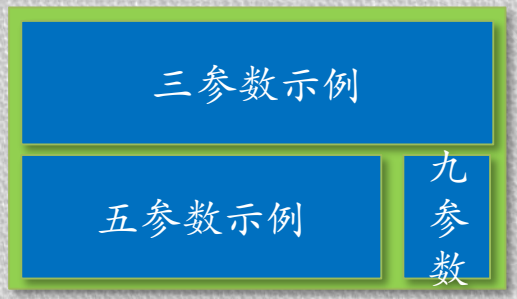
drawImage 示例

```
var myImage= new Image();   // Create new Image object
        myImage.src = "backdrop.png";

        function draw() {

                var canvas_1 = document.getElementById('canvas_1');
                if (canvas_1.getContext) {
                var ctx_1 = canvas_1.getContext("2d");
                ctx_1.drawImage(myImage,4,4); //三参数
                ctx_1.drawImage(myImage,3,93,177,110);// 五参数
                ctx_1.drawImage(myImage,80,3,40,40,182,90,60,60);
                ctx_1.drawImage(myImage,117,3,40,40,182,150,60,60);// 九参

                }
        }
```



图像平铺

HTML DOM createPattern() 方法

定义和用法

- createPattern() 方法为贴图图像创建一个模式。

语法

- createPattern(*image*, *repetitionStyle*)

参数

Image：需要贴图的图像。这个参数通常是一个 Image 对象，但是也可以使用一个 Canvas 元素。

repetitionStyle说明图像如何贴图。可能的值如下所示：

"repeat" - 在各个方向上都对图像贴图。默认值。

"repeat-x" - 只在 X 方向上贴图。

"repeat-y" - 只在 Y 方向上贴图。

"no-repeat" - 不贴图，只使用它一次。

注：
1.使用createPattern之后并**不是**在底部进行平铺。
2.火狐只支持“**repeat**”方式平铺，其他方式也只按repeat来。
3.与 drawImage 有点不同，你需要确认 image 对象**已经装载完毕**，否则图案可能效果不对的。

返回值

表示模式的一个 CanvasPattern 对象。

描述

createPattern() 方法创建并返回一个 CanvasPattern 对象，该对象表示一个贴图图像所定义的模式。要使用一个模式来勾勒线条或填充区域，可以把一个 CanvasPattern 对象用作 strokeStyle 属性或 fillStyle 属性的值。

Canvas的画布处理

- 图像裁剪（画布）
- 坐标变换
  - 平移 translate
  - 扩大 scale
  - 旋转 rotate
- 矩阵变换

图像裁剪

HTML DOM clip() 方法

使用Canvas绘制图像的时候，我们经常会想要只保留图像的一部分，这是我们可以使用canvas API再带的图像裁剪功能来实现这一想法。

Canvas API的图像裁剪功能是指，在画布内使用路径，只绘制该路径内所包含区域的图像，不绘制路径外的图像。

使用图形上下文的不带参数的Clip方法来实现Canvas的图像裁剪功能。该方法使用路径来对Canvas话不设置一个裁剪区域。因此，必须先创建好路径。创建完整后，调用clip方法来设置裁剪区域

图像裁剪示例

裁切后的图（部分代码未给）

素材图
（原图很大）

注*：裁剪是对画布进行的，裁切后的画布不能恢复到原来的大小，也就是说画布是越切越小的，要想保证最后仍然能在canvas最初定义的大小下绘图需要注意save和restore，后面会说到。
画布是先裁切完了再进行绘图。
并不一定非要是图片，路径也可以放进去~

```
ctx.beginPath();
ctx.moveTo(75,60);
ctx.bezierCurveTo(125,20,125,100,75,115);
ctx.bezierCurveTo(25,100,25,20,75,60);
ctx.clip();
ctx.closePath();
ctx.drawImage(myImage,20,30,100,100);
```

示例

Canvas的坐标变换

- 平移 translate
- 缩放 scale
- 旋转 rotate

注*：由于canvas的操作是直接对画布进行处理，例如在将画布放大2倍之后再往画布上画路径加图像都会出现新的图形图像是原图像的两倍，在旋转一次之后，再次画上的图形图像也会是在原基础上进行了旋转的，所以在一次进行变换操作之前应该适时的进行保存是很有必要的。

平移 translate
HTML DOM translate() 方法

定义和用法

- translate() 方法转换画布的用户坐标系统。

语法

- translate(*dx*, *dy*)

参数

- 参数描述dx, dy转换的量的 X 和 Y 大小。

描述

- translate() 方法为画布的变换矩阵添加水平的和垂直的偏移。参数 *dx* 和 *dy* 添加给后续定义路径中的所有点。

缩放 scale
HTML DOM scale() 方法

定义和用法

- scale() 方法标注画布的用户坐标系统。

语法

- scale(sx, sy)

描述

- scale() 方法为画布的当前变换矩阵添加一个缩放变换。缩放通过独立的水平和垂直缩放因子来完成。例如，传递一个值 2.0 和 0.5 将会导致绘图路径宽度变为原来的两倍，而高度变为原来的 1/2。指定一个负的 sx 值，会导致 X 坐标沿 Y 轴对折，而指定一个负的 sy 会导致 Y 坐标沿着 X 轴对折。

参数

参数	描述
sx, sy	水平和垂直的缩放因子。

旋转 rotate
HTML DOM rotate() 方法

定义和用法

- rotate() 方法旋转画布的坐标系统。

语法

- rotate(*angle*)

参数描述

- angle旋转的量，用弧度表示。正值表示顺时针方向旋转，负值表示逆时针方向旋转。

描述

- rotate() 方法通过指定一个角度，改变了画布坐标和 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的。它并没有旋转 <Canvas> 元素本身。注意，这个角度是用弧度指定的。

提示：

- 如需把角度转换为弧度，请乘以 Math.PI 并除以 180。

Canvas中图形图像的组合

HTML DOM globalCompositeOperation 属性

定义和用法

- globalCompositeOperation 属性说明如何在画布上组合颜色。

语法

- CanvasRenderingContext2D.globalCompositeOperation

描述

- globalCompositeOperation 属性说明了绘制到画布上的颜色是如何与画布上已有的颜色组合（或“合成”）的。
- 下面的表格列出了可能的值及其含义。这些值中的 "source" 一词，指的是将要绘制到画布上的颜色，而 "destination" 指的是画布上已经存在的颜色。默认值是 "source-over"。

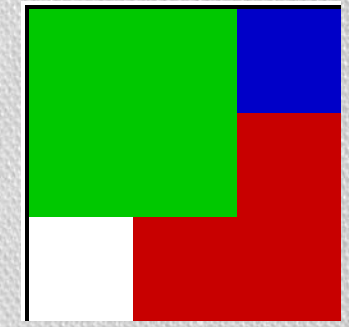
注：测试时发现火狐下兼容的效果出入很大，某些属性chrome下显示也并非其描述样式。

Canvas中图形图像的组合

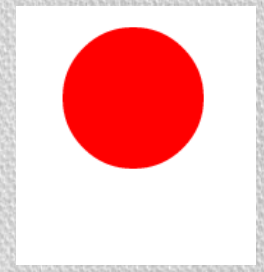
HTML DOM globalCompositeOperation 属性

值	含义
"copy"	只绘制新图形，删除其他所有内容。
"darker"	在图形重叠的地方，颜色由两个颜色值相减后决定。
"destination-atop"	已有的内容只有在它和新的图形重叠的地方保留。新图形绘制于内容之后。
"destination-in"	在新图形以及已有画布重叠的地方，已有内容都保留。所有其他内容成为透明的。
"destination-out"	在已有内容和新图形不重叠的地方，已有内容保留。所有其他内容成为透明。
"destination-over"	新图形绘制于已有内容的后面。
"lighter"	在图形重叠的地方，颜色由两种颜色值的加值来决定。
"source-atop"	只有在新图形和已有内容重叠的地方，才绘制新图形。
"source-in"	在新图形以及已有内容重叠的地方，新图形才绘制。所有其他内容成为透明。！！
"source-out"	只有在和已有图形不重叠的地方，才绘制新图形。！！
"source-over"	新图形绘制于已有图形的顶部。这是默认的行为。
"xor"	在重叠和正常绘制的其他地方，图形都成为透明的。

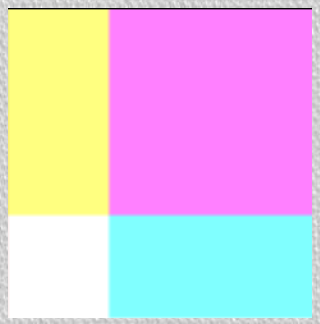
Canvas图形组合示例1



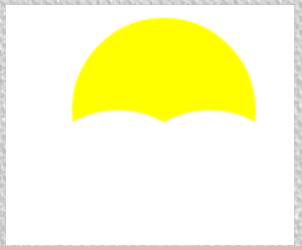
destination-over



destination-in 画了三个图形，红色为最先画上的，之后才加了该属性，所以以后的都不会显示了

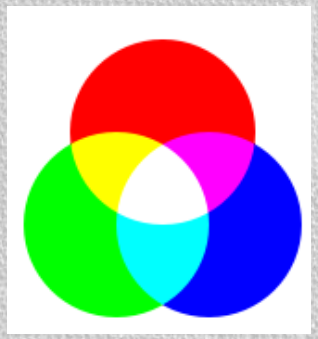


Copy透明度为0.5，尽管半透明但是也未显示下面得



destination-out 画了三个图形，黄色为最先画上的，之后才加了该属性，所以以后的和黄色相交的部分都切掉了。

Darker



Lighter

destination-atop

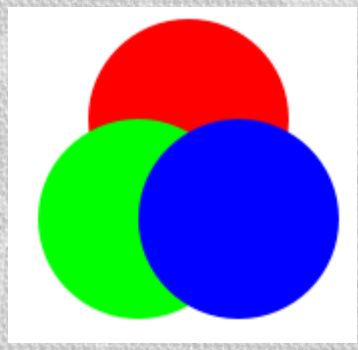
source-atop

Canvas组合示例2

source-in 只显示了最后相交的那一小块，chrome下显示与atop一样，该图为ff下截图

source-out 只显在与所有已有图形不重合的地方才显示
此图为chrome下的解释

source-out 只显在与所有已有图形不重合的地方才显示
此图为ff下的解释



source-over 默认样式
Ff下显示一样

Xor 没看懂！！！

Canvas的绘制文字

- 填充文字与文字描边
  - fillText ——填充字体
  - strokeText——轮廓描边
- 文字属性设置
  - font——字体
  - textAlign——水平对齐
  - textBaseline——垂直对齐

注：之所以说文字的绘制是因为文字在这里面是以图形的方式存在的，所以我们可以以处理图形的方式对文字处理，例如做渐变的文字，对画布进行缩放的时候尽量不要存在文字。

绘制文字 fillText()、strokeText()

fillText方法用于填充方式绘制字符串该方法定义如下所示：

- ctx.fillText(text,x,y,[maxWidth]);
- 该方法接受四个参数，第一个参数text表示绘制文字的内容，第二个参数x表示绘制文字的起点横坐标，第三个参数y表示绘制文字的起点纵坐标，第四个参数maxWidth为可选参数表示显示文字的最大宽度，可以防止溢出。

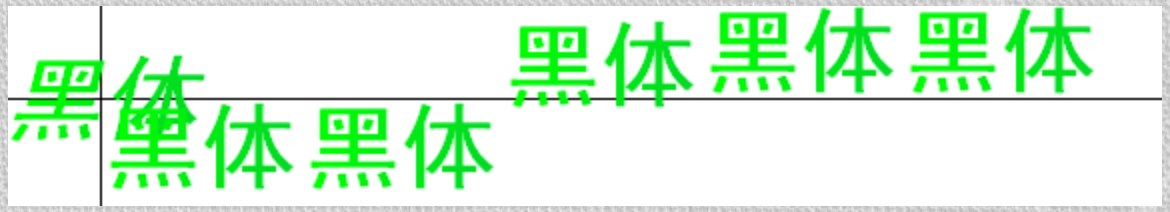
strokeText方法用于轮廓方式绘制字符串该方法定义如下所示：

- ctx.strokeText(text,x,y,[maxWidth]);
- 该方法的参数部分的解释与fillText方法相同；

文字样式

font、textAlign、textBaseline

- ctx.font='italic bolder 48px 黑体';
  - *经过测试：粗体斜体可以不加，粗体斜体顺序可变，但是不可以放在字体大小和字体后 大小和字体不可变。
- textAlign、textBaseline
  - 分别用来确定文字的水平对齐方式和垂直对齐方式
  - 下图为垂直方向对齐的各种展示 图中横线为文字的y值。



示例

Canvas其他知识

- 保存与恢复状态
  - Save Restore
- 保存文件
  - toDataURL(“image/jpeg”)
- 获取像素信息
  - getImageData putImageData
- 基本动画
  - clearRect()
  - setInterval

Canvas中状态的保存与恢复

Save和restore是一对操作，分别对应进栈和出栈的操作，canvas把当前的状态通过save放入堆栈然后在需要退回到之前的某种状态时就通过restore返回到之前的状态。

前面已经提到很多地方用到了save和restore，因为这些操作都会影响到未来的绘图操作，如果不还原到最初的状态就会出现我们所不想要的效果，例如在时钟动画中如果我们在画完钟表表盘之后未回退那么很有可能看到的是整个画面都在旋转。

可以保存的状态现在我所知道的有对画布的操作包括裁剪、旋转、缩放、平移，有定义的各种样式包括填充样式、描边样式、字体样式等。

另外当我们进行填充和描边的时候所用的样式字体样式，也会被保存为状态，当然你可以随时重新给他们新的样式，如果你不希望出现你预想之外的事情，或者你想用之前的样式，那么你就需要使用保存和恢复状态了。

canvas保存文件

通过保存文件可以将canvas内的内容保存为图片，保存文件的原理其实是把当前的绘画状态输出到一个dataURL地址指向的数据中的过程，所谓dataURL，是指目前达到或数浏览器能够识别的一种base64位编码的URL，主要用于小型的、可以在网页中直接嵌入而不需要从外部文件嵌入的数据，譬如img元素中的图像文件等。

<img src="...36KgiobUjiXRQhP4yf53+5r55JQxEDUMSZZ3onSmI+ijiDUQx4IdfoJ6ohi4HIYjaSGIks/B5nd///Ql89taYfc7JONwifVWf9hCzW/dfbBn/T/umrR2wYhVEvg/QwcVePJopYoy+L/V/0dT6N9k9fPWJDESPR9qge9uCBh5FlaCEgI1/09T3N9k9fPULTQ1em6wF9T6PO5SL9s6Y/103Q59D+6atHbHgLZxSxCFmcfaKtC+298QSIhXgL5xd9Xc+i/dNXj9j0E6TeRwtC2/QYnbUJ7eSE9/4nDNC/aP/01WM0JNFftyMOPNTyWgtkO4AsziGJfi/69z/P9k9fPWJDFjXIDbHdUWCnac/t0F+zFlnUeNG/+0W0f/rqEZueCPCa3lnxKOIntDjuVfrjeL2zXnvilKL/z9s/ffUYzbbZK4sv0bJNTiGLLwkQ9f4nVzJP0P7pq8dseAvn/+kVzFNs//TVP+2fZtf+H7wWySg/aOLgAAAAAElFTkSuQmCC“>

右边的这段放进html就可以直接得到左边的图片

Canvas获得像素

在html5中使用Canvas API所能够做到的图像处理技术中还包括像素处理技术。使用CanvasAPI能够获取图像中的每一个像素，然后得到该像素颜色的rgb或rgba值。使用图形上下文对象的getImageData来获取图像中的像素。

imageData变量是一个CanvasPixeArray对象，具有height，width，data等属性。Data属性是一个保存像素数据的数组，内容类似[r1,g1,b1,a1,r2,g2,b2,a2…]就是每个像素点的rgba值，data.length为索取的像素的数量。

取得了像素信息之后就能够对每个像素进行处理。实现各种复杂的操作，包括人像识别，图像蒙版等，示例中可以通过该方法获得canvas内图像的RGB通道和Alpha通道。

通过putImageData可以讲处理后的图像呈现到 画布上。

```
var imagedata =ctx.getImageData(x,y,w,h);
for(var i=0,n =imagedata.data.length;i<n;i+=4){
if (a==2)  imagedata.data[i+0] =imagedata.data[i+1] = imagedata.data[i+2];
if(a==0)   imagedata.data[i+1] =imagedata.data[i+2] = imagedata.data[i+0];
if(a==1)   imagedata.data[i+0] =imagedata.data[i+2] = imagedata.data[i+1];
if(a==3)   imagedata.data[i+1]=imagedata.data[i+2]=imagedata.data[i+0]=imagedata.data[i+3];
}
ctx.putImageData(imagedata,x,y);
}
```

Js 存在安全性问题 本地不能实现需要上传到本地服务器

Canvas基本动画

- Canvas的动画是通过不停的擦除、重绘来实现的。具体步骤如下：
  ① 预先编写好用来绘图的函数，在该函数中先用clearRect方法将画布整体或局部擦除。
  ② 用setInterval方法设置动画时间间隔。
  ③ 在比较复杂的情况下，我们也可以灾情处于重回动画的当中插入当前绘制状态的保存与恢复，变成擦除、保存绘制状态、进行绘制、恢复状态的过程。

动画实例1跳动的小球（坐标改变）

给小球X方向和Y方向各三个状态，向下（右）、向下（右）撞到下（右）面“墙壁”后、以及再次撞到上（左）面“墙壁”，当然一定有更好的实现方式。

效果是小球简单的弹动。动画很简单，不牵扯画布的操作，所以不存在保存状态和恢复状态。

```
setInterval(ball,20);
ctx.clearRect(0,0,300,230);
ctx.drawImage(Ball,p,i);
 //ctx.fillRect(p,i,50,50);
```

用两种方式实现了小球，黄色为直接用的png图片，绿色为用渐变填充的。两种方式实现的方式是相同的，都是通过不停在不同位置删除、绘制图形或图像来达到动画的效果。
另外我们也可以通过移动画布来实现小球的移动。

动画实例2跳动的小球（移动画布）

我们上次给了小球两组状态，现在这两组状态我们要给画布，所谓“撞墙”也就变成了画布在x或y方向上是否已经移动了画布的宽或高。

所实现效果与上例完全一样，但是实现的时候原理却完全不一样，这个动画我们通过移动画布来让小球移动了，图中的黑色线条便是画布的边缘。

```
ctx.clearRect(-300,-230,600,600);
/*相比移动小球我们每次要擦除更多的画布，因为上次小球出现的位置现在已经不在我们的画布上了。*/
ctx.drawImage(Ball,0,0);/*我们把球放在了(0,0)位置，之后便没有再对他进行处理。*/
ctx.translate(movex,movey);/*但是，我们对画布进行了移动，注意，我们用了两个参数但是这两个参数我们只修改参数的正负，不进行累加，因为画布每次平移之后，只要不进行保存恢复操作，那么他的移动是基于移动后的位置的。*/
p+=speedx;
i+=speedy;
/*但是我们要引入两个变量来记录画布已经相对原点的位置，因为我们要据此来检测碰撞*/
```