Canvas作为HTML5的新功能不是所有浏览器都支持，需要做支持判断。

<canvas>

你的浏览器不支持canvas标签

</canvas>

canvas支持点、线段、形状的绘制，也支持渐变色、绘制外部图像。

canvas自带绘制规则矩形图形的api

//js代码

var c= document.getElementById("canv");

var m=c.getContext("2d");

m.fillStyle="#f00";

m.fillReact(0,0,300,300);

m.fillStyle="rgba(0,0,255,0.5)";

m.fillReact(200,200,500,500);

绘制矩形rect()、fillRect()和strokeRect()

1. context.rect( x , y , width , height )：只定义矩形的路径；
2. context.fillRect( x , y , width , height )：直接绘制出填充的矩形；
3. context.strokeRect( x , y , width , height )：直接绘制出矩形边框

这里需要说明两点：第一点就是stroke()和fill()绘制的前后顺序，如果fill()后面绘制，那么当stroke边框较大时，会明显的把stroke()绘制出的边框遮住一半；第二点：设置fillStyle或strokeStyle属性时，可以通过“rgba(255,0,0,0.2)”的设置方式来设置，这个设置的最后一个参数是透明度。

自带线段api

m.lineWidth=5;//定义线条的宽度

m.strokeStyle="red";//定义线条的样式

m.moveTo(10,10);//起始位置

m.lineTo(150,150);终止位置

m.lineTo(10,50);如果没有再次设置起始位置将从结束位置开始画

m.stroke() 开始划线。

注意：如果没有stroke()函数，表示还没有画图终止，会不给你显示！！

所以，画点完成必须stroke()

绘制线段moveTo()和lineTo()，如果没有通过moveTo()特别指定，lineTo()的起始点是以上一个点为准。因此，如果需要重新选择起始点，则需要通过moveTo()方法。如果需要对不同的线段设置样式，则需要通过context.beginPath()重新开启一条路径。

绘制圆形

var cObj=document.getElementById("myCanvas");

var cnt = cObj.getContext("2d");

cnt.beginPath();//从新开始画，防止冲突重叠

cnt.fillStyle="#0ff";

cnt.arc(100,100,50,0,Math.PI\*2,true);

cnt.closePath();//结束画布，防止冲突重叠

cnt.fill();//开始渲染

arc()里面有6个参数，第一个参数和第二个参数表示所要画弧(圆)的圆心，第三个参数表示半径，第四个参数0表示从3点钟的方向开始画(x轴哪里)，第五个参数表示所画圆弧的大小，第六个参数为true表示逆时针，false为顺时针.

渐变色

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

var cnt = c.getContext("2d");

var m=cnt.createLinearGradient(0,0,175,50);

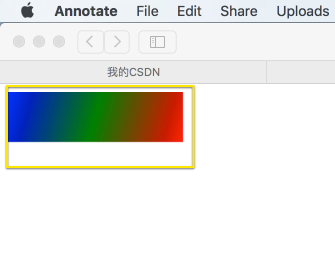
m.addColorStop(0,"blue");

m.addColorStop(0.5,"green");

m.addColorStop(1,"red");

cnt.fillStyle = m;

cnt.fillRect(0,0,175,50);



也可以载入外部图像来填充

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

var cnt = c.getContext("2d");

var img = new Image();

img.src = "04.jpg";

cnt.drawImage(img,0,0);

需要注意的是，只载入的话图片可以是任意网络资源，但是要对canvas图片做编辑的话必须是同源图片否则会有安全限制。

对于canvas里的图型可以做区域擦除。

ctn.clearRect(0,0,100,50);//从(0,0)点开始擦除长100，宽50的面积

可以通过擦除像素点来达到手工橡皮差的清除效果。

**线条属性**

1、lineWidth设置绘制线段的宽度，单位为px：

2、lineCap 属性设置或返回线条末端线帽的样式，可以取以下几个值：

“butt” 向线条的每个末端添加平直的边缘（默认）；

“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形线帽。

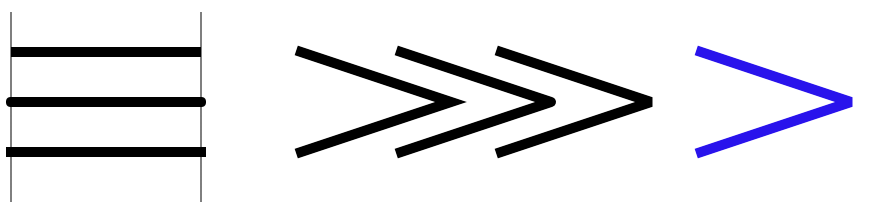
3、lineJoin 属性当两条线交汇时设置或返回所创建边角的类型，可以取以下几个值：

“miter” 创建尖角(默认)；

“bevel” 创建斜角；

“round” 创建圆角。

4、miterLimit 属性设置或返回最大斜接长度（默认为10）。斜接长度指的是在两条线交汇处内角和外角之间的距离。只有当 lineJoin 属性为 “miter” 时，miterLimit 才有效。



**填充样式**

前面用到的fillStyle和strokeStyle除了设置颜色外，还能设置其他填充样式，这里以fillStyle为例：

**线性渐变**

使用步骤

（1）var grd = context.createLinearGradient( xstart , ystart, xend , yend )创建一个线性渐变，设置起始坐标和终点坐标；

（2）grd.addColorStop( stop , color )为线性渐变添加颜色，stop为0~1的值；

（3）context.fillStyle=grd将赋值给context。

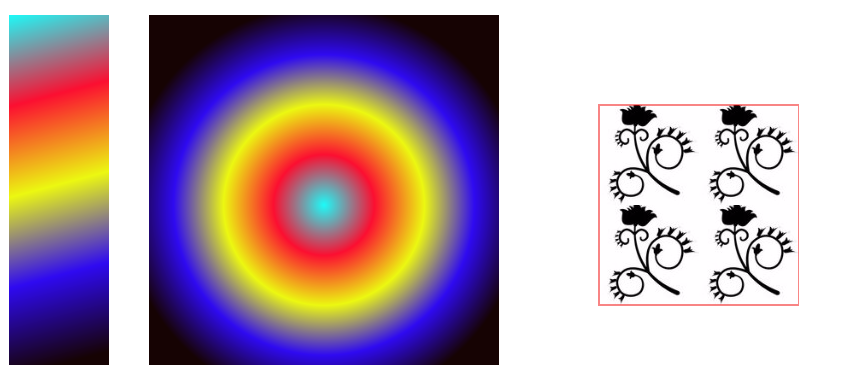
**径向渐变**

该方法与线性渐变使用方法类似，只是第一步接收的参数不一样

var grd = context.createRadialGradient(x0 , y0, r0 , x1 , y1 , r1 );接收起始圆心的坐标和圆半径以及终点圆心的坐标和圆的半径。

**位图填充**

createPattern( img , repeat-style )使用图片填充，repeat-style可以取repeat、repeat-x、repeat-y、no-repeat。



**图形变换**

平移：context.translate(x,y)，接收参数分别为原点在x轴方向平移x，在y轴方向平移y。

缩放：context.scale(x,y)，接收参数分别为x坐标轴按x比例缩放，y坐标轴按y比例缩放。

旋转：context.rotate(angle)，接收参数是坐标轴旋转的角度。

需要说明的是，对图形进行变化后，接下来的一次绘图是紧接着上一次的状态的，所以如果需要回到初始状态，要用到context.save();和context.restore();来保存和恢复当前状态:

另外一个跟图形变换相关的是：矩阵变换 ：context.transform(a, b, c, d, e, f, g)。参数的含义如下：

a 水平缩放 ( 默认为1 )

b 水平倾斜 ( 默认为 0 )

c 垂直倾斜 ( 默认为 0 )

d 垂直缩放 ( 默认为1 )

e 水平位移 ( 默认为 0 )

f 垂直位移 ( 默认为 0 )

**绘制曲线**

跟绘制曲线的有四个函数，分别是：

context.arc(x,y,r,sAngle,eAngle,counterclockwise);用于创建弧/曲线（用于创建圆或部分圆）。接收的参数含义：

| 参数 | 含义 |

| :————- |:————-|

| 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|

下面是几个arc()函数的几个示例：

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

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

context.strokeStyle = "#F22D0D";

context.lineWidth = "2";

//绘制圆

context.beginPath();

context.arc(100,100,40,0,2\*Math.PI);

context.stroke();

//绘制半圆

context.beginPath();

context.arc(200,100,40,0,Math.PI);

context.stroke();

//绘制半圆,逆时针

context.beginPath();

context.arc(300,100,40,0,Math.PI,true);

context.stroke();

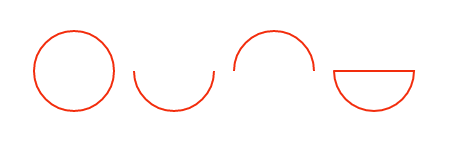
//绘制封闭半圆

context.beginPath();

context.arc(400,100,40,0,Math.PI);

context.closePath();

context.stroke();



context.arcTo(x1,y1,x2,y2,r); 在画布上创建介于两个切线之间的弧/曲线。接收的参数含义：



这里需要注意的是arcTo函数绘制的曲线的起始点需要通过moveTo(）函数来设置

context.quadraticCurveTo(cpx,cpy,x,y);绘制二次贝塞曲线，参数含义如下：



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。

context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y); 绘制三次贝塞尔曲线，参数如下：



**文字渲染**

与文本渲染有关的主要有三个属性以及三个方法：



上述的属性和方法的基本用法如下：

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

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

context.font="bold 30px Arial"; //设置样式

context.strokeStyle = "#1712F4";

context.strokeText("欢迎来到我的博客！",30,100);

context.font="bold 50px Arial";

var grd = context.createLinearGradient( 30 , 200, 400 , 300 );//设置渐变填充样式

grd.addColorStop(0,"#1EF9F7");

grd.addColorStop(0.25,"#FC0F31");

grd.addColorStop(0.5,"#ECF811");

grd.addColorStop(0.75,"#2F0AF1");

grd.addColorStop(1,"#160303");

context.fillStyle = grd;

context.fillText("欢迎来到我的博客！",30,200);

context.save();

context.moveTo(200,280);

context.lineTo(200,420);

context.stroke();

context.font="bold 20px Arial";

context.fillStyle = "#F80707";

context.textAlign="left";

context.fillText("文本在指定的位置开始",200,300);

context.textAlign="center";

context.fillText("文本的中心被放置在指定的位置",200,350);

context.textAlign="right";

context.fillText("文本在指定的位置结束",200,400);

context.restore();

context.save();

context.moveTo(10,500);

context.lineTo(500,500);

context.stroke();

context.fillStyle="#F60D0D";

context.font="bold 20px Arial";

context.textBaseline="top";

context.fillText("指定位置在上面",10,500);

context.textBaseline="bottom";

context.fillText("指定位置在下面",150,500);

context.textBaseline="middle";

context.fillText("指定位置居中",300,500);

context.restore();

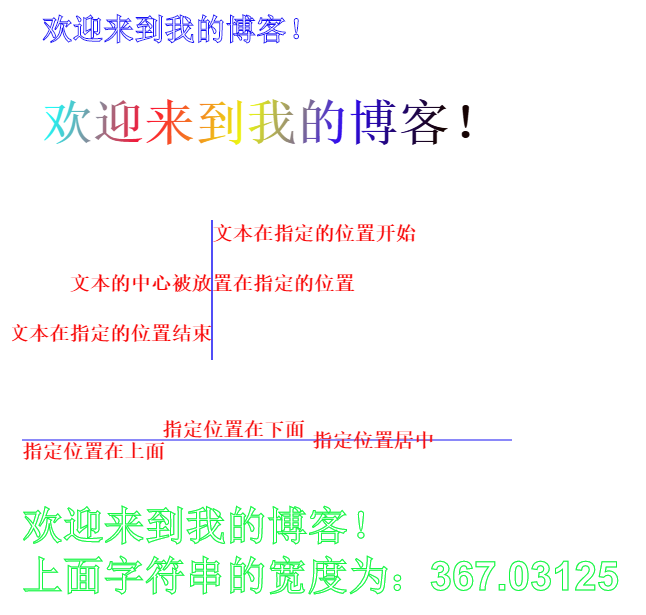
context.font="bold 40px Arial";

context.strokeStyle = "#16F643";

var text = "欢迎来到我的博客！";

context.strokeText("欢迎来到我的博客！",10,600);

context.strokeText("上面字符串的宽度为："+context.measureText(text).width,10,650);



**其他属性和方法**

**阴影绘制：**

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（数值越大越模糊）。

shadowOffsetX 设置或返回阴影与形状的水平距离。

shadowOffsetY 设置或返回阴影与形状的垂直距离。

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

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

context.beginPath();

//设置是个顶点的坐标，根据顶点制定路径

for (var i = 0; i < 5; i++) {

context.lineTo(Math.cos((18+i\*72)/180\*Math.PI)\*200+200,

-Math.sin((18+i\*72)/180\*Math.PI)\*200+200);

context.lineTo(Math.cos((54+i\*72)/180\*Math.PI)\*80+200,

-Math.sin((54+i\*72)/180\*Math.PI)\*80+200);

}

context.closePath();

//设置边框样式以及填充颜色

context.lineWidth="3";

context.fillStyle = "#F6F152";

context.strokeStyle = "#F5270B";

context.shadowColor = "#F7F2B4";

context.shadowOffsetX = 30;

context.shadowOffsetY = 30;

context.shadowBlur = 2;

context.fill();

context.stroke();



**图形组合：**

globalAlpha： 设置或返回绘图的当前 alpha 或透明值

该方法主要是设置图形的透明度，这里就不具体介绍。

globalCompositeOperation： 设置或返回新图像如何绘制到已有的图像上，该方法有以下属性值：



**剪辑区域：**

clip()方法从原始画布中剪切任意形状和尺寸。

提示：一旦剪切了某个区域，则所有之后的绘图都会被限制在被剪切的区域内（不能访问画布上的其他区域）。您也可以在使用 clip() 方法前通过使用 save() 方法对当前画布区域进行保存，并在以后的任意时间对其进行恢复（通过 restore() 方法）

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

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

context.beginPath();

context.fillStyle = "#0C0101";

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

context.beginPath();

context.fillStyle = "#FFFDFD";

context.arc(400,400,100,0,2\*Math.PI);

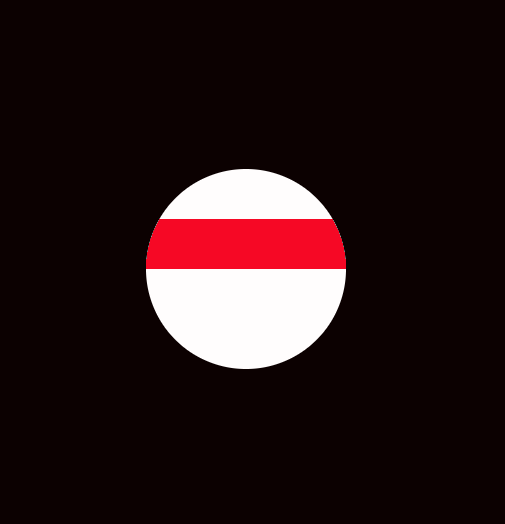
context.fill();

context.clip();

context.beginPath();

context.fillStyle = "#F60825";

context.fillRect(200, 350, 400,50);



除了上述的属性的和方法，还有以下等方法：

drawImage()： 向画布上绘制图像、画布或视频。

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

var cnt = c.getContext("2d");

var img = new Image();

img.src = "04.jpg";

cnt.drawImage(img,0,0);

toDataURL() ：保存图形

canvas.toDataURL(fileType, rate)

dataURL=canvas.toDataURL('image/jpeg'); //转换图片为dataURL Base64格式

dataURL=canvas.toDataURL('image/jpeg',0.6)

转成jpeg时加上图片品质，可以简单实现jpeg的图片压缩.

isPointInPath()： 如果指定的点位于当前路径中，则返回 true，否则返回 false。

context.isPointInPath(30, 30) //true or false

由于isPointInPath方法仅判断当前上下文环境中的路径，所以当Canvas里已经绘制了多个图形时，仅能以最后一个图形的上下文环境来判断事件。例如，如果我们最后画的是右腿，那么我们只能判断是否点击了右腿，而不能判断身体的其他部分——右腿最后画。 这种问题的解决方法是：当点击事件发生时，重绘所有图形，每绘制一个就使用isPointInPath方法，判断事件坐标是否在该图形覆盖范围内。也就是说你不能画了10条路径后突然跑去判断点是不是在第1条路径之中。为什么要这样呢?因为canvas中路径是不能保存的,不能像一个变量那样保留起来供后面再用.这样一来你想要重现这个路径的话,惟一的办法就是:按照之前的值,之前的函数,重新画一遍.