# Canvas 常用操作有哪些

canvas 是一种抽象概念，是 2D 图形系统中的重要部分。Canvas API（画布）用于在网页实时生成图像，并且可以操作图像内容，基本上它是一个可以用 JavaScript 操作的位图（bitmap）。

## 开始使用

一、首先新建一个<canvas>网页元素。

```
<canvas id="myCanvas" width="400" height="200">
    你的浏览器不支持 canvas!
</canvas>
```

上面代码中，如果浏览器不支持这个 API，则就会显示<canvas>标签中间的文字——“您 de 浏览器不支持 canvas！”。

每个 canvas 节点都有一个对应的 context 对象（上下文对象），Canvas API 定义在这个 context 对象上面，所以需要获取这个对象，方法是使用 getContext 方法。

```
var canvas = document.getElementById('myCanvas');
if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
}
```

上面代码中，getContext 方法指定参数 2d，表示该 canvas 节点用于生成 2D 图案（即平面图案）。如果参数是 webgl，就表示用于生成 3D 图像（即立体图案），这部分实际上单独叫做 WebGL API。

## 绘图方法

canvas 画布提供了一个用来作图的平面空间，该空间的每个点都有自己的坐标，x 表示横坐标，y 表示竖坐标。原点(0, 0)位于图像左上角，x 轴的正向是原点向右，y 轴的正向是原点向下

（1）绘制路径

beginPath 方法表示开始绘制路径，moveTo(x, y)方法设置线段的起点，lineTo(x, y)方法设置线段的终点，stroke 方法用来给透明的线段着色。

```
ctx.beginPath(); // 开始路径绘制
```

```
ctx.moveTo(20, 20); // 设置路径起点，坐标为(20,20)
ctx.lineTo(200, 20); // 绘制一条到(200,20)的直线
ctx.lineWidth = 1.0; // 设置线宽
ctx.strokeStyle = '#CC0000'; // 设置线的颜色
ctx.stroke(); // 进行线的着色，这时整条线才变得可见
```

moveto 和 lineto 方法可以多次使用。最后，还可以使用 closePath 方法，自动绘制一条当前点到起点的直线，形成一个封闭图形，省却使用一次 lineto 方法。

（2）绘制矩形

fillRect(x, y, width, height)方法用来绘制矩形，它的四个参数分别为矩形左上角顶点的 x 坐标、y 坐标，以及矩形的宽和高。fillStyle 属性用来设置矩形的填充色。

```
ctx.fillStyle = 'yellow';
ctx.fillRect(50, 50, 200, 100);
```

strokeRect 方法与 fillRect 类似，用来绘制空心矩形。

```
ctx.strokeRect(10,10,200,100);
```

clearRect 方法用来清除某个矩形区域的内容。

```
ctx.clearRect(100,50,50,50);
```

（3）绘制文本

fillText(string, x, y) 用来绘制文本，它的三个参数分别为文本内容、起点的 x 坐标、y 坐标。使用之前，需用 font 设置字体、大小、样式（写法类似与 CSS 的 font 属性）。与此类似的还有 strokeText 方法，用来添加空心字。

```
// 设置字体
ctx.font = "Bold 20px Arial";
// 设置对齐方式
ctx.textAlign = "left";
// 设置填充颜色
ctx.fillStyle = "#008600";
// 设置字体内容，以及在画布上的位置
ctx.fillText("Hello!", 10, 50);
// 绘制空心字
ctx.strokeText("Hello!", 10, 100);
```

fillText 方法不支持文本断行，即所有文本出现在一行内。所以，如果要生成多行文本，只有调用多次 fillText 方法。

（4）绘制圆形和扇形

arc 方法用来绘制扇形

ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);

arc 方法的 x 和 y 参数是圆心坐标，radius 是半径，startAngle 和 endAngle 则是扇形的起始角度和终止角度（以弧度表示），anticlockwise 表示做图时应该逆时针画（true）还是顺时针画（false）。

下面是如何绘制实心的圆形。

```
ctx.beginPath();
ctx.arc(60, 60, 50, 0, Math.PI * 2, true);
```

```
        ctx.fillStyle = "#000000";
        ctx.fill();
```

绘制空心圆形的例子。

```
        ctx.beginPath();
        ctx.arc(60, 60, 50, 0, Math.PI*2, true);
        ctx.lineWidth = 1.0;
        ctx.strokeStyle = "#000";
        ctx.stroke();
```

（5）设置渐变色

createLinearGradient 方法用来设置渐变色。

```
var myGradient = ctx.createLinearGradient(0, 0, 0, 160);
myGradient.addColorStop(0, "#BABABA");
myGradient.addColorStop(1, "#636363");
```

createLinearGradient 方法的参数是(x1, y1, x2, y2)，其中 x1 和 y1 是起点坐标，x2 和 y2 是终点坐标。通过不同的坐标值，可以生成从上至下、从左到右的渐变等等。

使用方法如下：

```
ctx.fillStyle = myGradient;
ctx.fillRect(10,10,200,100);
```

（6）设置阴影

一系列与阴影相关的方法，可以用来设置阴影。

```
ctx.shadowOffsetX = 10; // 设置水平位移
ctx.shadowOffsetY = 10; // 设置垂直位移
ctx.shadowBlur = 5; // 设置模糊度
ctx.shadowColor = "rgba(0,0,0,0.5)"; // 设置阴影颜色
ctx.fillStyle = "#CC0000";
ctx.fillRect(10,10,200,100);
```

图像处理方法

## drawImage 方法

Canvas API 允许将图像文件插入画布，做法是读取图片后，使用 drawImage 方法在画布内进行重绘。

```
var img = new Image();
img.src = 'image.png';
ctx.drawImage(img, 0, 0); // 设置对应的图像对象，以及它在画布上的位置
```

上面代码将一个 PNG 图像载入画布。drawImage()方法接受三个参数，第一个参数是图像文件的 DOM 元素（即<img>节点），第二个和第三个参数是图像左上角在画布中的坐标，上例中的(0, 0)就表示将图像左上角放置在画布的左上角。

由于图像的载入需要时间，drawImage 方法只能在图像完全载入后才能调用，因此上面的代码需要改写。

```
var image = new Image();

image.onload = function() {
  var canvas = document.createElement('canvas');
  canvas.width = image.width;
  canvas.height = image.height;
  canvas.getContext('2d').drawImage(image, 0, 0);
  // 插入页面底部
  document.body.appendChild(image);
  return canvas;
}

image.src = 'image.png';
```

## getImageData 方法，putImageData 方法

getImageData 方法可以用来读取 Canvas 的内容，返回一个对象，包含了每个像素的信息。

```
var imageData = context.getImageData(0, 0, canvas.width, canvas.height);
```

imageData 对象有一个 data 属性，它的值是一个一维数组。该数组的值，依次是每个像素的红、绿、蓝、alpha 通道值，因此该数组的长度等于 图像的像素宽度 x 图像的像素高度 x 4，每个值的范围是 0 - 255。这个数组不仅可读，而且可写，因此通过操作这个数组的值，就可以达到操作图像的目的。修改这个数组以后，使用 putImageData 方法将数组内容重新绘制在 Canvas 上。

```
context.putImageData(imageData, 0, 0);
```

## toDataURL 方法

对图像数据做出修改以后，可以使用 toDataURL 方法，将 Canvas 数据重新转化成一般的图像文件形式。

```
function convertCanvasToImage(canvas) {
  var image = new Image();
  image.src = canvas.toDataURL('image/png');
  return image;
}
```

上面的代码将 Canvas 数据，转化成 PNG data URI。

## save 方法，restore 方法

save 方法用于保存上下文环境，restore 方法用于恢复到上一次保存的上下文环境。

```
ctx.save();
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur = 5;
ctx.shadowColor = 'rgba(0,0,0,0.5)';
ctx.fillStyle = '#CC0000';
ctx.fillRect(10, 10, 150, 100);
ctx.restore();
ctx.fillStyle = '#000000';
ctx.fillRect(180, 10, 150, 100);
```

上面代码先用 save 方法，保存了当前设置，然后绘制了一个有阴影的矩形。接着，使用 restore 方法，恢复了保存前的设置，绘制了一个没有阴影的矩形。

## 动画

利用 JavaScript，可以在 canvas 元素上很容易地产生动画效果。

```
var posX = 20,
    posY = 100;
setInterval(function() {
    context.fillStyle = "black";
    context.fillRect(0,0,canvas.width, canvas.height);
    posX += 1;
    posY += 0.25;
    context.beginPath();
    context.fillStyle = "white";
    context.arc(posX, posY, 10, 0, Math.PI*2, true);
    context.closePath();
    context.fill();
}, 30);
```

上面代码会产生一个小圆点，每隔 30 毫秒就向右下方移动的效果。setInterval 函数的一开

始，之所以要将画布重新渲染黑色底色，是为了抹去上一步的小圆点。
通过设置圆心坐标，可以产生各种运动轨迹。
先上升后下降。

```
var vx = 10,
    vy = -10,
    gravity = 1;

setInterval(function() {
    posX += vx;
    posY += vy;
    vy += gravity;
    // ...
});
```

上面代码中，x 坐标始终增大，表示持续向右运动。y 坐标先变小，然后在重力作用下，不断增大，表示先上升后下降。
小球不断反弹后，逐步趋于静止。

```
var vx = 10,
        vy = -10,
        gravity = 1;
    setInterval(function () {
        posX += vx;
        posY += vy;
        if (posY > canvas.height * 0.75) {
            vy *= -0.6;
            vx *= 0.75;
            posY = canvas.height * 0.75;
        }
        vy += gravity;
        // ...
    });
```

上面代码表示，一旦小球的 y 坐标处于屏幕下方 75%的位置，向 x 轴移动的速度变为原来的 75%，而向 y 轴反弹上一次反弹高度的 40%。