## **概述**

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）。

使用前，首先需要新建一个<canvas>网页元素。

<canvas id="myCanvas" width="400" height="200">

您的浏览器不支持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——“您的浏览器不支持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（本书不涉及）。

## **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.lineCap = "butt"; //设置端点样式:butt(默认),round围绕,square方形

ctx.lineJoin = "miter"; //设置连接样式:miter(默认),bevel斜边,round圆

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

ctx.fill(); //填充当前的图像（路径）。默认颜色是黑色。

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);

## **图像变换**

### **平移、旋转、缩放**

ctx.translate( x, y )//位移：把图像原点位移到(x， y)的位置ctx.rotate( deg )//旋转：旋转 deg 度数ctx.scale( sx, sy )//缩放：在横向进行 sx 倍的缩放，在纵向进行 sy 倍的缩放

缩放出现的问题 1.如果有lineWith，宽度也会缩放 2.如果起始点不是0 0，起始点也会缩放

### **变换矩阵**

ctx.transform(a, b, c, d, e, f);/\*

a:水平缩放(默认值1)

b:水平倾斜(默认值0)

c:垂直倾斜(默认值0)

d:垂直缩放(默认值1)

e:水平位移(默认值0)

f:垂直位移(默认值0)

\*/

context.transform()可以叠加使用，如果需要重新初始化矩阵变换的值，可以用:context.setTransform(a, b, c, d, e, f) 它会使得之前设置的context.transform()失效，恢复为单位矩阵然后再transform

## **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%。

## **像素处理**

通过getImageData方法和putImageData方法，可以处理每个像素，进而操作图像内容。

假定filter是一个处理像素的函数，那么整个对Canvas的处理流程，可以用下面的代码表示。

if (canvas.width > 0 && canvas.height > 0) {

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

filter(imageData);

context.putImageData(imageData, 0, 0);

}

以下是几种常见的处理方法。

### **灰度效果**

灰度图（grayscale）就是取红、绿、蓝三个像素值的算术平均值，这实际上将图像转成了黑白形式。假定d[i]是像素数组中一个象素的红色值，则d[i+1]为绿色值，d[i+2]为蓝色值，d[i+3]就是alpha通道值。转成灰度的算法，就是将红、绿、蓝三个值相加后除以3，再将结果写回数组。

grayscale = function (pixels) {

var d = pixels.data;

for (var i = 0; i < d.length; i += 4) {

var r = d[i];

var g = d[i + 1];

var b = d[i + 2];

d[i] = d[i + 1] = d[i + 2] = (r+g+b)/3;

}

return pixels;

};

### **复古效果**

复古效果（sepia）则是将红、绿、蓝三个像素，分别取这三个值的某种加权平均值，使得图像有一种古旧的效果。

sepia = function (pixels) {

var d = pixels.data;

for (var i = 0; i < d.length; i += 4) {

var r = d[i];

var g = d[i + 1];

var b = d[i + 2];

d[i] = (r \* 0.393)+(g \* 0.769)+(b \* 0.189); // red

d[i + 1] = (r \* 0.349)+(g \* 0.686)+(b \* 0.168); // green

d[i + 2] = (r \* 0.272)+(g \* 0.534)+(b \* 0.131); // blue

}

return pixels;

};

### **红色蒙版效果**

红色蒙版指的是，让图像呈现一种偏红的效果。算法是将红色通道设为红、绿、蓝三个值的平均值，而将绿色通道和蓝色通道都设为0。

red = function (pixels) {

var d = pixels.data;

for (var i = 0; i < d.length; i += 4) {

var r = d[i];

var g = d[i + 1];

var b = d[i + 2];

d[i] = (r+g+b)/3; // 红色通道取平均值

d[i + 1] = d[i + 2] = 0; // 绿色通道和蓝色通道都设为0

}

return pixels;

};

### **亮度效果**

亮度效果（brightness）是指让图像变得更亮或更暗。算法将红色通道、绿色通道、蓝色通道，同时加上一个正值或负值。

brightness = function (pixels, delta) {

var d = pixels.data;

for (var i = 0; i < d.length; i += 4) {

d[i] += delta; // red

d[i + 1] += delta; // green

d[i + 2] += delta; // blue

}

return pixels;

};

### **反转效果**

反转效果（invert）是指图片呈现一种色彩颠倒的效果。算法为红、绿、蓝通道都取各自的相反值（255-原值）。

invert = function (pixels) {

var d = pixels.data;

for (var i = 0; i < d.length; i += 4) {

d[i] = 255 - d[i];

d[i+1] = 255 - d[i + 1];

d[i+2] = 255 - d[i + 2];

}

return pixels;

};