画布

* canvas元素默认具有高宽width： 300px/ height：150px

* <canvas> 元素只是创造了一个固定大小的画布，要想在它上面去绘制内容，

我们需要找到它的渲染上下文

* <canvas> 元素有一个叫做 getContext() 的方法，这个方法是用来获得渲染上下文和它的绘画功能。

使用的基本格式

**套路：**

在body：

<canvas width="400" height="400">//**画布的宽高只能在这里设置**

您的浏览器不支持canvas画布 请你换成萌萌的谷歌浏览器</canvas>

**在js中：**

var canvas = document.getElementById('tutorial');

if (canvas.getContext){// **getContext() 的方法，这个方法是用来获得渲染上下文和它的绘画功能。**

var ctx = canvas.getContext('2d');//**舞台是2d的**

这三个都要写

ctx.save();//

**这里面写画笔的样式**

**画图的方法**

ctx.beginPath();//清除之前的轨迹

**这里面画图的轨迹**

ctx.restore();

**清除压栈的样式**

}

**save()** 是 Canvas 2D API 通过将当前状态放入栈中，保存 canvas 全部状态的方法

保存到栈中的绘制状态有下面部分组成：

当前的变换矩阵。/当前的剪切区域。/ 当前的虚线列表.

以下属性当前的值： strokeStyle,

fillStyle,

lineWidth,

lineCap,

lineJoin...

**restore()**是 Canvas 2D API 通过在绘图状态栈中弹出顶端的状态，将 canvas 恢复到最近的保存状态的方法。如果没有保存状态，此方法不做任何改变。

**画笔的样式**

设置图形的填充颜色。**画笔方法对应好颜色**

**strokeStyle** : 设置图形轮廓的颜色。**ctx. strokeStyle = ‘颜色’**

默认情况下，线条和填充颜色都是黑色（CSS 颜色值 #000000）

**lineWidth :** 这个属性设置当前绘线的粗细。属性值必须为正数。

**ctx. lineWidth = 数字’**

描述线段宽度的数字。 0、 负数、 Infinity 和 NaN 会被忽略。

默认值是1.0。

**lineCap：**设定线条与线条间接合处的样式（默认是 miter）

**round :** 圆角 四角圆润 在两边加半圆

**bevel :** 斜角 四角一刀切

**miter :** 直角 矩形

**ctx.globalAlpha=0-1；** :绘制出来的是全局透明度设置，在0-1之间

**路径画图：**

**beginPath()**

* 新建一条路径，生成之后，图形绘制命令被指向到路径上准备生成路径。
* 生成路径的第一步叫做beginPath()。本质上，路径是由很多子路径构成，这些子路径都是在一个列表中，
* 所有的子路径（线、弧形、等等）构成图形。而每次这个方法调用之后，列表清空重置，

然后我们就可以重新绘制新的图形。

* **画直线：**

**moveTo(x, y)** 起点：将笔触移动到指定的坐标x以及y上

**lineTo(x, y)** 终点：绘制一条从当前位置到指定x以及y位置的直线。

* **画矩形**

**rect(x, y, width, height)+绘制的两种方法**:X和Y是偏移量，后面两个是长宽

**context.strokeRect(10,10,50,50)**//直接画，是一个实心的矩形

**context.strokeRect(10.5,10.5,50,**50)//直接画，是一个有2px的边框的空心矩形

**备：**这里实际上是1px的变宽，但是到浏览器不认0.5会画成1，所以就会变成2个，要是想要1px的边框，就让前面的偏移量变成x+0.5

* **画圆：如果有宽度，长度非常小，加一个lineCap=‘round’也行**

arc(x, y, radius, startAngle, endAngle, ture：逆时针/false:顺时针)

* **画弧形**

arcTo(x1, y1, x2, y2, radius)

根据给定的控制点和半径画一段圆弧

肯定会从(x1 y1) 但不一定经过(x2 y2);(x2 y2)只是控制一个方向

**根据轨迹画图的方法：**

**ctx.stroke()** 通过线条来绘制图形轮廓。

**ctx.fill()**  通过填充路径的内容区域生成实心的图形。

**转化**

**ctx.translate(x, y)**

我们先介绍 translate 方法，它用来移动 canvas的原点到一个不同的位置。

translate 方法接受两个参数。x 是左右偏移量，y 是上下偏移量，

**在canvas中translate是累加的**

**ctx.rotate(angle)**

这个方法只接受一个参数：旋转的角度(angle)，它是顺时针方向的，以弧度为单位的值。

旋转的中心点始终是 canvas 的原点，如果要改变它，我们需要用到 translate 方法

**在canvas中rotate是累加的**

**ctx.scale(x, y)**

scale 方法接受两个参数。x,y 分别是横轴和纵轴的缩放因子，它们都必须是正值。

值比 1.0 小表示缩小，比 1.0 大则表示放大，值为 1.0 时什么效果都没有。

缩放一般我们用它来增减图形在 canvas 中的像素数目，对形状，位图进行缩小或者放大。

**使用图片/设置背景**

**用图片的套路：**

**此处省略之前假装一万字的获得画笔的套路。。。。**

ctx.beginPath();

var img = new Image();

img.src = 'img/q\_r1.jpg';//要放的图片的路径

img.onload = function () {//要在图片全部加载完了再动

draw(this);//在这里添加一个函数来写图片的操作

//这个this可以指代上面那个图片，也可以传给下一个函数 }

function draw(img){//在这个函数里面写图片位置和动作

**ctx.drawImage**(img,value,0)

}

**解释：**

**ctx.drawImage(image, x, y, width, height)**

imgage是对像，x，y是图片的起始放置位置

width，height是图片的宽高

**ctx.createPattern(image, repetition)**

**image:图像源**

repetition: "repeat"/"repeat-x"/"repeat-y"/"no-repeat"

一般情况下，我们都会将createPattern返回的对象作为fillstyle的值

**图片的合成：**

**ctx.globalCompositeOperation：可取值（destination-4种/source-4种）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **写法：**ctx.save();  ctx.fillStyle="pink";  ctx.beginPath();  ctx.fillRect(0,0,100,100);//之前的画的图，成图**destination:已经绘制过的图形(目标)**  **ctx.globalCompositeOperation**="destination-atop";  ctx.fillStyle="deeppink";  ctx.fillRect(50,50,100,100);//后面画的图，成图叫做**source:新的图像(源)**  ctx.restore(); | | | | | | | |
| **source：只要出现这个的，就是后面画的在最上面，覆盖之前** | | | | **destination：最先画的在最上面，覆盖之后画的** | | | |
| source-over(默) | source-in | source-out | source-atop | destination-over | destination-in | destination-out | destination-atop |
| 源在上面,新的图像层级比较高 | 只留下源与目标的**重叠**部分(源的那一部分) | 只留下源超过目标的部分，  只保**留不重叠**的 | 砍掉源溢出的部分，不要**源不重叠**的部分 | 目标在上面,旧的图像层级比较高 | 只留下源与目标的**重叠**部分(目标的那一部分) | 只留下目标超过源的部分，后面的图片和重叠的部分一起消失。 | 砍掉目标溢出的部分 |
|  |  |  |  |  |  |  |  |

**像素的操作：**

getImageData():获得一个包含画布场景像素数据的ImageData对像,它代表了画布区域的对象数据

例如：**ctx.getImageData(sx, sy, sw, sh)**

**（起始位置x，y，要取的宽度，要取的高度）**

**ImageData**对象中存储着canvas对象真实的像素数据，它包含以下几个只读属性：

获得所有的像素点：

var allPx = imgData.width \* imgData.height;

**操作单个像素：**

ImageData function getPxInfo(imgData,x,y){

var color =[];

var data = imgData.data;

//存放的是里面的像素的r gba的各个值的数组

var w = imgData.width;

var h = imgData.height;

**//宽\*高+x的值\*4 获得像素点的值**

**//(x,y)之前有多少个像素点: y\*imgData.width + x**

color[0]= data[(y\*w+x)\*4];

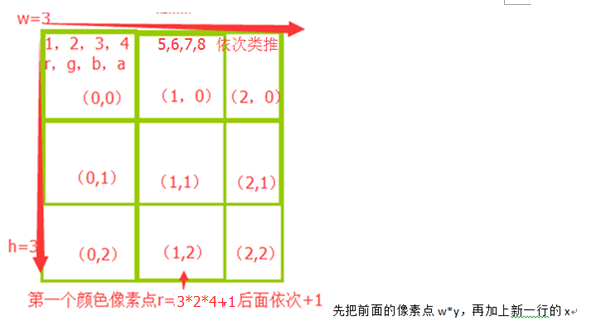
color[1]= data[(y\*w+x)\*4+1];

color[2]= data[(y\*w+x)\*4+2];

color[3]= data[(y\*w+x)\*4+3];

return color;

}



width:图片宽度，单位是像素

height:图片高度，单位是像素

data:Uint8ClampedArray类型的一维数组，

包含着RGBA格式的整型数据，范围在0至255之间（包括255）

R:0 --> 255(黑色到白色)

G:0 --> 255(黑色到白色)

B:0 --> 255(黑色到白色)

A:0 --> 255(透明到不透明)

**在场景中写入像素数据**

putImageData()方法去对场景进行像素数据的写入。

putImageData(准备插入的对象, dx, dy)

dx和dy参数表示你希望在场景内左上角绘制的像素数据所得到的设备坐标

###创建一个ImageData对象

ctx.createImageData(width, height);

width : ImageData 新对象的宽度。

height: ImageData 新对象的高度。

**默认创建出来的是透明的**

function setPxInfo(imgData,x,y,color){

//(x,y)之前有多少个像素点: y\*imgData.width + x

var data = imgData.data;

var w = imgData.width;

var h = imgData.height;

data[(y\*w+x)\*4] = color[0];

data[(y\*w+x)\*4+1] = color[1];

data[(y\*w+x)\*4+2] = color[2];

data[(y\*w+x)\*4+3] = color[3];

}

**与视频相结合：**

**body里：**

<video src="video/test.mp4" autoplay width="0" height="0"></video>

**js里面放：**

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

setInterval(function(){

//大概放在这个位置，加个定时器来播放

ctx.drawImage(video,0,0,canvas.width,canvas.height)

（获得video的dom标签，位置起始x，y，在画布的大小w，h）

})

画布的事件

1. 点击画布就可以用的事件

直接给画布添加事件，都是点击画布就可以触发

1. 就是画出来的东西添加事件**,作用于最后画的东西**

canvas.onmousedown = function(ev){

ev = ev||event;

var x = ev.clientX - this.offsetLeft;

var y = ev.clientY - this.offsetTop;

if(ctx.isPointInPath(x,y)){

alert(123);

}

}

3.如果想一个画的东西给一个事件，就构造函数出来了之后，new一个新