**Canvas应用**

大家好，今天我来简单讲述一下HTML5中的canvas标签，canvas是个很神奇的标签，它给我们提供一张画布，你可以在上面画任何你想到的东西，有人说手绘能做到的事canvas也能做到，这句话可能有夸大，但也能说明canvas功能强大。

<canvas>是为了客户端矢量图形而设计的，浏览器自己没有行为，但却把一个绘图 API 展现给客户端 JavaScript 以使脚本能够把想绘制的东西都绘制到一块画布上。

<canvas>标记由 Apple 在 Safari 1.3 Web 浏览器中引入, 对 HTML 的这一根本扩展的原因在于，HTML 在 Safari 中的绘图能力也为 Mac OS X 桌面的 Dashboard 组件所使用，并且 Apple 希望有一种方式在 Dashboard 中支持脚本化的图形。Firefox 1.5 和 Opera 9 都跟随了 Safari 的引领。这两个浏览器都支持 <canvas> 标记。我们甚至可以在 IE 中使用 <canvas> 标记，并在 IE 的 VML 支持的基础上用开源的 JavaScript 代码（由 Google 发起）来构建兼容性的画布。<canvas> 的标准化的努力由一个 Web 浏览器厂商的非正式协会在推进，现在已经是HTML5中的一名正式成员！

<canvas>的默认大小为300像素×150像素，它只是生成一个容器，就像一个画布本身是没有绘画能力的，需要使用画笔才行，在<canvas>上，这个画笔就是它自身带的getContext()方法，该方法可以传递一个参数，2d或者3d,今天我所讲述的内容都是基于2d的基础。

需要注意的是，canvas本身是一个HTML元素，所以它是可以使用CSS进行宽高的设置的，但是在绘制时画布会以默认大小为基础伸缩来适应css设置的宽高，如果CSS的尺寸与初始画布的比例不一致，它会出现扭曲。所以一般情况下，我们都会使用canvas的width以及height属性来设置宽高。

有了画布跟画笔，我们还需要颜料才能够真正的开始绘制，在开始绘制之前，我们简单了解一下canvas的坐标系，手绘的时候我们都是随意下笔，但是在canvas中，每一步都要考虑到它是从哪个坐标出来，通过怎么样的方式到达另一个坐标才能绘制出我们想要的图形，在这里显示鼠标对应在canvas上的坐标点，可以看到canvas横坐标与纵坐标的0值都是在左上角，往下为X轴正值，往右为Y轴正值。再来看看canvas常用的API。

常规操作：

* getContext('2d') 获取2d对象
* beginPath() 起始（重置）当前路径
* moveTo( x, y ) 将笔触移动到指定的坐标(x,y)
* fill() 填充当前绘图（路径）
* stroke() 绘制已定义的路径

Canvas给我们提供常规图形的绘制方法：

* 矩形：rect( x, y, width, height )
* 填充矩形：fillRect( x, y, width, height )
* 无填充矩形：strokeRect( x, y, width, height )

可以看到右边是canvas使用以上API画出来的图形，看似第一个跟第三个一摸一样，它们有什么区别呢？rect()方法会根据你最后使用的是fill()方法还是stroke()方法，渲染成不同的样子。lineTo方法只能使用stroke()绘制，因为它是一个路径，不存在填充的情况。所以使用fill()方法时，lineTo绘制的方法是看不到的。

* 绘制直线：lineTo( x, y )
* 二次贝塞尔曲线：quadraticCurveTo()
* 创建三次贝塞尔曲线：bezierCurveTo()

一次贝塞尔曲线，其实就是直线了，所以我这里把它归到了贝塞尔曲线里面，贝塞尔曲线类似photoShop中的钢笔。确定两个点，然后以弧线连过去，不同的是，PS中通过角度来调整圆弧的大小，canvas中则通过坐标点的距离，坐标点离得越远，圆弧就越小，一次塞尔曲线是连接两个点，那么二次就是连接三个点，以此类推，那么问题来了，为什么二次贝塞尔曲线只有两个参数呢？这是因为参数中不包括起始点，默认会从之前绘制结束的点开始，如果要调整起始点，我们可以用moveTo方法，这个方法会把画笔不留痕迹的移到你指定的位置。

* 绘制圆或圆弧：arc( x, y, radius, startAngle, endAngle, anticlockwise)
* 绘制扇形：arcTo( x1, y1, x2, y2, radius)

虽然圆弧看起来有两个函数可以使用，但是在MDN的介绍上我们可以看到，arcTo函数不太靠谱，连MDN都不是很想介绍它，当然根本原因是因为arcTo可以实现的东西，arc也能实现，而且操作起来更精准。

Canvas同样可以设置字体颜色大小，还能够自由的设置画笔的大小粗细，除了单色，也可以自由的使用渐变色与阴影。

* 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x用于笔触的颜色、渐变或模式
* globalAlpha 设置透明度

globalAlpha 这个属性相对于rgba来说不太灵活，只能单一得设置透明度，并且具有继承性，而rgba可以同时改变颜色以及透明度。

* 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 设置或返回阴影与形状的水平距离
* shadowOffsetY 设置或返回阴影与形状的垂直距离
* lineCap 设置或返回线条的结束点样式（butt、round、square）

butt和square都是方形，butt是对齐坐标点，square则在两端加上了二分之一线宽的高度；

* lineJoin 设置或返回当两条线交汇时，边角的类型（bevel、round、miter）

bevel 和 miter 是可以根据，miterLimit互相转换的，倘若设置为miter时，两条线之间的角度过小，内角至外角的长度超过了miterLimi设置的数值，则会显示为bevel

* miterLimit 设置或返回最大斜接长度（斜接长度指的是在两条线交汇处内角和外角之间的距离。）
* lineWidth 设置或返回当前的线条宽度
* createLinearGradient( x0, y0, x1, y1 ) 创建线性渐变
* createRadialGradient( x0, y0, r0, x1, y1, r1 ) 创建径向渐变
* addColorStop( stop, color ) 规定渐变对象中的颜色和停止位置

设置字体并绘制

* font 设置或返回文本内容的当前字体属性（和css的font一样）
* textAlign 设置或返回文本内容的当前对齐方式
* textBaseline 设置或返回在绘制文本时使用的当前文本基线
* fillText( text, x, y ) 在画布上绘制“被填充”的文本
* strokeText( text, x, y ) 在画布上绘制文本（无填充）
* measureText( text ) 返回包含指定文本宽度的对象（属性width获取宽度）

面向图片的操作：

* drawImage( image/canvas, x, y )、drawImage( image/canvas, x, y, width, height )、drawImage( image/canvas, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight ) 在画布上绘制图像、画布或视频
* toDataURL() 将canvas转换成图片，返回地址
* toBlob() 将canvas转换成bolb类型

以上是我们处理图片常用的API，将图片绘制到画布上，根据需求，可以使用toDataURL()将画布转换成base64格式或使用toBolob转换成bolb类型，也就是file类型。

画布变形操作：

* save() 保存当前环境的状态
* restore() 恢复之前保存过的路径状态和属性
* scale( x, y ) 缩放当前绘图
* translate( x, y ) 重新设置画布上的(0,0)位置
* rotate( angle ) 选择当前绘图，单位为“弧度”，角度转弧度公式(degrees\*Math.PI/180)
* transform( m11, m12, m21, m22, dx, dy ) 替换绘图的当前转换矩阵(m11:水平缩放绘图,m12:水平倾斜绘图,m21:垂直倾斜绘图,m22:垂直缩放绘图,dx:水平移动绘图,dy:垂直移动绘图);
* setTransform() 将当前转换重置为单元矩阵，然后运行transform()

基本动画：

requestAnimationFrame最大的优势是由系统来决定回调函数的执行时机。具体一点讲，如果屏幕刷新率是60Hz,那么回调函数就每16.7ms被执行一次，如果刷新率是75Hz，那么这个时间间隔就变成了1000/75=13.3ms，换句话说就是，requestAnimationFrame的步伐跟着系统的刷新步伐走。它能保证回调函数在屏幕每一次的刷新间隔中只被执行一次，这样就不会引起丢帧现象，也不会导致动画出现卡顿的问题。

其他：

* clip() 从原始画布剪切任意形状和尺寸的区域
* isPointInPath( x, y ) 检测指定的点是否在当前路径中，在则返回true。
* createPattern( image/canvas/video, repeat ) 在指定的方向内重复绘制指定的元素
* getImageData() 获取canvas的像素数据
* globalCompositeOperation 设置或返回新图像如何绘制到已有的图像上。

叠合操作模式。画布中**先后绘制**的内容会根据不同的模式来呈现效果。  
默认为后绘制的覆盖先绘制的，也可以设置为先绘制的覆盖后绘制的，甚至可以让两次绘制重叠部分镂空或者提亮颜色。

CanvasAPI如此多，覆盖范围也十分的广泛，不仅可以绘制常规图形图片，还能绘制视频及动画，那么具体的它能做些什么呢？这里准备了三个小案例。

第一个是手写签名的简易版，鼠标按下可以开始绘制，可以撤销恢复清除以及导出。

签名的实现原理：

1. 鼠标按下时开启记录点，
2. 鼠标移动时计算出相对canvas的坐标点，并将起绘制出来。
3. 鼠标弹起时关闭记录点。

这样就可以记录鼠标在按下时移动的轨迹。

撤销与恢复的实现原理：

1. 使用toDataURL将当前的canvas对象转换成base64生成快照，并使用数组存储，
2. 用户点击撤销或者点击恢复时，使用drawImage()方法将数组中对应索引的快照在canvas上进行重绘。
3. 执行新的绘制操作时，删除当前位置之后的数组记录，然后添加新的快照。

清除与导出的实现原理：

1. 清除时使用clearRect()方法，将整个画布大小的矩形范围做清除。
2. 导出则使用了toBlob()方法，将canvas转换成blob类型，然后使用window.URL.createObjectURL()方法生成一个下载链接，使用a标签的href属性点击下载。

第二个是个趣味刮刮卡，主要思路：

1. 随机选择数组中的数据，在canvas上绘制出来然后生成图片，放置到canvas的背景图片上。
2. 清空画布后绘制刮刮卡的图层
3. 捕捉鼠标操作，鼠标按下时，设置canvas的globalCompositeOperation属性为destination-out
4. 在鼠标经过的地方绘制圆形或方形，因为已经设置了globalCompositeOperation，所以新绘制出来的地方会变成透明的，就营造了一种刮刮卡的效果。
5. 为了用户体验，在大部分蒙版被刮掉之后，使蒙版层消失。这里可以使用getImageData()方法获取canvas上的数据点，这些数据点以4个为一组，分别代表了像素点的rgba的值，计算出有多少像素点为透明或半透明状态，达到一定比例触发clearRect()方法清除蒙版层。

第三个是图片的裁剪与合成，主要思路

1. 将选择的图片放置到画布上
2. 捕捉鼠标滚动事件，使用canvas的scale属性对画布进行缩放
3. 用户选择边框后保存边框图片。
4. 用户点击裁剪时，计算画布的缩放值以及到父盒子的距离，得到数据后使用drawImage方法绘制到另一个canvas中
5. 如果有边框的图片数据，再将边框绘制上去。
6. 用户点击导出后，同样的利用toBlob方法与a标签将图片下载下来。

Canvas的应用领域十分广阔，以上仅仅是我自己最近做组件时的整理，它的功能当然不止这些，canvas 的 context 支持自定义，可以自己添加一些原创的方法和规则进去；canvas 甚至能进行一些图像处理，例如打马赛克，添加滤镜等等，在此就暂不深入介绍了。有兴趣的同事可以去MDN中查看canvasApi的介绍以及教程。今天我的分享就到这，谢谢大家