

# 数据可视化 Canvas

刘军 liujun

# 目录

## content



**1 邂逅 Canvas**

**2 Canvas绘制图形**

**3 Canvas样式和颜色**

**4 Canvas状态和形变**

**5 Canvas动画和案例**






## ■ 什么是Canvas

- Canvas 最初由Apple于2004 年引入，用于Mac OS X WebKit组件，为仪表板小部件和Safari浏览器等应用程序提供支持。后来，它被Gecko内核的浏览器（尤其是Mozilla Firefox），Opera和Chrome实现，并被网页超文本应用技术工作小组提议为下一代的网络技术的**标准元素（HTML5新增元素）**。
- Canvas提供了非常多的JavaScript绘图API（比如：绘制路径、矩形、圆、文本和图像等方法），与<canvas>元素**可以绘制各种2D图形**。
- Canvas API 主要聚焦于 2D 图形。当然也可以使用<canvas>元素对象的 WebGL API 来绘制 2D 和 3D 图形。

## ■ Canvas的应用场景

- 可以用于**动画、游戏画面、数据可视化、图片编辑以及实时视频处理**等方面。

## ■ Canvas 浏览器兼容性

Element					
<canvas>	4.0	9.0	2.0	3.1	9.0

## ■ Canvas 优点:

- Canvas提供的功能更原始，适合像素处理，动态渲染和数据量大的绘制，如：图片编辑、热力图、炫光尾迹特效等。
- Canvas非常适合图像密集型的游戏开发，适合频繁重绘许多的对象。
- Canvas能够以 .png 或 .jpg 格式保存结果图像，适合对图片进行像素级的处理。

## ■ Canvas 缺点:

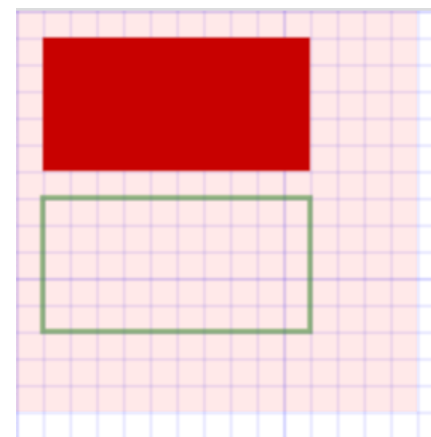
- 在移动端可能会因为Canvas数量多，而导致内存占用超出了手机的承受能力，导致浏览器崩溃。
- Canvas 绘图只能通过JavaScript脚本操作 (all in js) 。
- Canvas 是由一个个像素点构成的图形，放大会使图形变得颗粒状和像素化，导致模糊。

## ■ 使用Canvas的注意事项:

- `<canvas>` 和 `<img>` 元素很相像, 唯一的不同就是它并没有 `src` 和 `alt` 属性。
- `<canvas>` 标签只有两个属性——`width`和`height`( 单位默认为`px` )。当没有设置宽度和高度时, `canvas` 会初始化宽为 `300px` 和高为 `150px`。
- 与 `<img>` 元素不同, `<canvas>` 元素**必须需要结束标签** (`</canvas>`)。如结束标签不存在, 则文档其余部分会被认为是**替代内容**, 将不会显示出来。
- 测试 `canvas.getContext()` 方法的存在, 可以检查浏览器是否支持Canvas。

## ■ 初体验Canvas

- 1.Canvas通用模板
- 2.Canvas绘制正方形



# Canvas Grid 和 坐标空间

■ 在开始画图之前，我们需要了解一下Canvas网格（canvas grid）和 坐标系。

■ Canvas Grid 或 坐标空间

□ 假如，HTML 模板中有个宽 150px, 高 150px 的 `<canvas>` 元素。`<canvas>`元素默认被网格所覆盖。

□ 通常来说网格中的一个单元相当于 canvas 元素中的一像素。

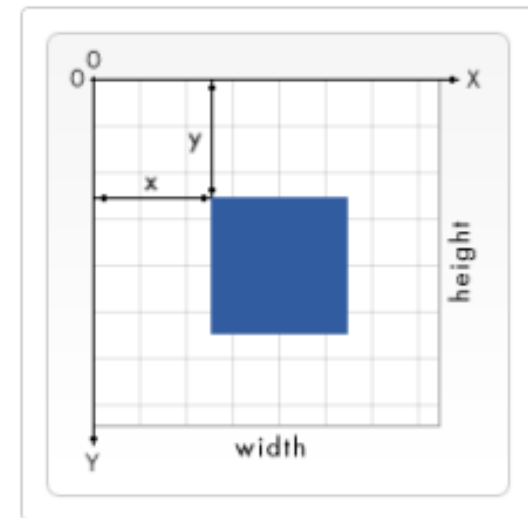
□ 该网格的原点位于坐标 (0,0) 的左上角。所有元素都相对于该原点放置。

□ 网格也可以理解为坐标空间（坐标系），坐标原点位于canvas元素的左上角，被称为初始坐标系

✓ 如右图中蓝色正方形，左上角的坐标为距离左边 x 像素，距离上边 y 像素，坐标为 (x, y)

□ 网格或坐标空间是可以变换的，后面会讲如何将原点转换到不同的位置，旋转网格甚至缩放它。

✓ 注意：移动了原点后，默认所有后续变换都将基于新坐标系的变换。



# 绘制矩形( Rectangle )

## ■ Canvas支持两种方式来绘制矩形：矩形方法 和 路径方法。

- 路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合。
- 除了矩形，其他的图形都是通过一条或者多条路径组合而成的。
- 通常我们会通过众多的路径来绘制复杂的图形。

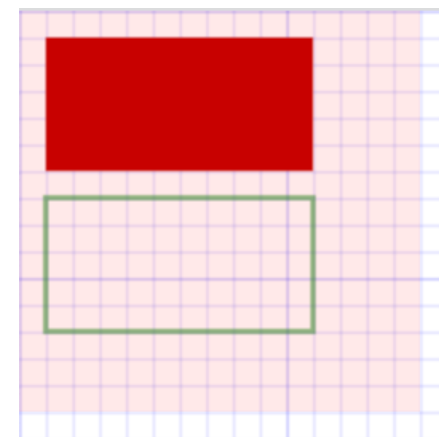


## ■ Canvas 绘图的矩形方法：

- fillRect(x, y, width, height): 绘制一个填充的矩形
- strokeRect(x, y, width, height): 绘制一个矩形的边框
- clearRect(x, y, width, height): 清除指定矩形区域，让清除部分完全透明。

## ■ 方法参数：

- 上面的方法都包含了相同的参数。
- x 与 y 指定了在canvas画布上所绘制矩形的左上角（相对于原点）的坐标（不支持 undefined）。
- width 和 height 设置矩形的尺寸。



## ■ 什么是路径？

- 图形的基本元素是路径。路径是通过不同颜色和宽度的**线段或曲线相连形成的不同形状的点的集合**。
- **路径**是可由很多**子路径**构成，这些子路径都是在一个**列表**中，列表中所有**子路径（线、弧形等）**将构成图形。
- 一个**路径**，甚至一个**子路径**，通常都是闭合的。



## ■ 使用路径绘制图形的步骤：

- 1.首先需要创建路径起始点 (beginPath) 。
- 2.然后使用画图命令去画出路径( arc 、 lineTo )。
- 3.之后把路径闭合( closePath , 不是必须)。
- 4.一旦路径生成，就能通过**描边(stroke)**或**填充路径区域(fill)**来渲染图形。

## ■ 以下是绘制路径时，所要用到的函数

- **beginPath()**：新建一条路径，生成之后，图形绘制命令被指向到新的路径上绘图，**不会关联到旧的路径**。
- **closePath()**：闭合路径之后图形绘制命令又重新指向到 beginPath之前的上下文中。
- **stroke()**：通过线条来绘制图形轮廓/描边（**针对当前路径图形**）。
- **fill()**：通过填充路径的内容区域生成实心的图形（**针对当前路径图形**）。

```
ctx.beginPath();
ctx.arc(75, 75, 50, 0, Math.PI * 2, true); // 绘制
ctx.moveTo(110, 75);
ctx.arc(75, 75, 35, 0, Math.PI, false); // 口 (顺时针)
ctx.moveTo(65, 65);
ctx.arc(60, 65, 5, 0, Math.PI * 2, true); // 左眼
ctx.moveTo(95, 65);
ctx.arc(90, 65, 5, 0, Math.PI * 2, true); // 右眼
ctx.stroke();
```



# 路径-绘制直线

## ■ 移动画笔 (moveTo) 方法

- moveTo 方法是不能画出任何东西，但是它也是路径列表的一部分
- moveTo 可以想象为在纸上作业，一支钢笔或者铅笔的笔尖从一个点到另一个点的移动过程。
- moveTo(x, y): 将笔移动到指定的坐标 x、y 上。
- 当 canvas 初始化或者beginPath()调用后，我们通常会使用moveTo(x, y)函数设置起点。
- 使用moveTo函数能够绘制一些不连续的路径。

## ■ 绘制直线 (lineTo) 方法

- lineTo(x, y): 绘制一条从当前位置到指定 (x, y)位置的直线。
  - ✓ 该方法有两个参数(x, y)代表坐标系中直线结束的点。
  - ✓ 开始点和之前的绘制路径有关，之前路径的结束点就是接下来的开始点。
  - ✓ 当然开始点也可以通过moveTo(x, y)函数改变。

## ■ 绘制一条直线

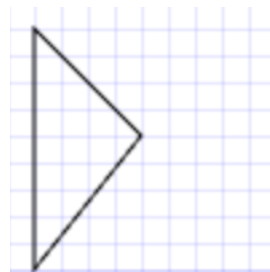
- 第一步：调用 beginPath() 来生成路径。本质上，路径是由很多子路径（线、弧形、等）构成。
- 第二步：调用moveTo、lineTo函数来绘制路径（路径可以是连续也可以不连续）。
- 第三步：闭合路径 closePath()，虽然不是必需的，但是通常都是要闭合路径。
- 第四步：调用stroke()函数来给直线描边。



# 路径-绘制三角形( Triangle )

## ■ 绘制一个三角形步骤

- 第一步：调用 `beginPath()` 来生成路径。
- 第二步：调用 `moveTo()`、`lineTo()` 函数来绘制路径。
- 第三步：闭合路径 `closePath()`，不是必需的。
  - ✓ `closePath()` 方法会通过绘制一条从当前点到开始点的直线来闭合图形。
  - ✓ 如果图形是已经闭合了的，即当前点为开始点，该函数什么也不做。
- 第四步：调用 `stroke()` 函数来给线描边，或者调用 `fill()` 函数来填充（使用填充 `fill` 时，路径会自动闭合，而 `stroke` 不会）。



# 路径-绘制圆弧 (Arc)、圆 (Circle)

## ■ 绘制圆弧或者圆，使用arc()方法。

□ `arc(x, y, radius, startAngle, endAngle, anticlockwise)`，该方法有六个参数：

- ✓ `x`、`y`：为绘制圆弧所在圆上的圆心坐标。
- ✓ `radius`：为圆弧半径。
- ✓ `startAngle`、`endAngle`：该参数用弧度定义了开始以及结束的弧度。这些都是以 `x` 轴为基准。
- ✓ `anticlockwise`：为一个布尔值。为 `true`，是逆时针方向，为 `false`，是顺时针方向，默认为 `false`。

## ■ 计算弧度

□ `arc()` 函数中表示角的单位是弧度，不是角度。

□ 角度与弧度的 JS 表达式：弧度 =  $(\text{Math.PI} / 180) * \text{角度}$ ，即  $1\text{角度} = \text{Math.PI} / 180$  个弧度

- ✓ 比如：旋转  $90^\circ$ ： $\text{Math.PI} / 2$ ；旋转  $180^\circ$ ： $\text{Math.PI}$ ；旋转  $360^\circ$ ： $\text{Math.PI} * 2$ ；旋转  $-90^\circ$ ： $-\text{Math.PI} / 2$ ；

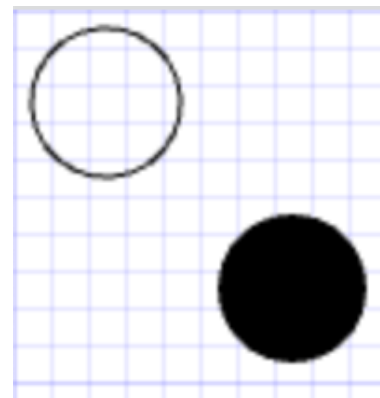
## ■ 绘制一个圆弧的步骤

□ 第一步：调用 `beginPath()` 来生成路径。

□ 第二步：调用 `arc()` 函数来绘制圆弧。

□ 第三步：闭合路径 `closePath()`，不是必需的。

□ 第四步：调用 `stroke()` 函数来描边，或者调用 `fill()` 函数来填充（使用填充 `fill` 时，路径会自动闭合）。



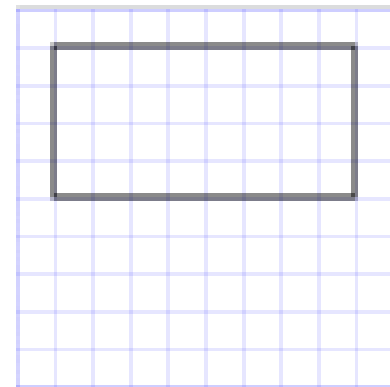
# 路径-矩形 (Rectangle)

## ■ 绘制矩形的另一个方法:

- 调用rect() 函数绘制, 即将一个矩形路径增加到当前路径上
- rect(x, y, width, height)
  - ✓ 绘制一个左上角坐标为 (x,y) , 宽高为 width 以及 height 的矩形。

## ■ 注意:

- 当该方法执行的时候, moveTo(x, y) 方法自动设置坐标参数 (0,0) 。也就是说, 当前笔触自动重置回默认坐标。



■ 前面已经学过了很多绘制图形的方法。如果我们**想要给图形上色**，有两个重要的属性可以做到：

□ **fillStyle = color**：设置图形的填充颜色，需在 **fill()** 函数前调用。

□ **strokeStyle = color**：设置图形轮廓的颜色，需在 **stroke()** 函数前调用。

```
// 这些 fillStyle 的值均为 '橙色'  
ctx.fillStyle = "orange";  
ctx.fillStyle = "#FFA500";  
ctx.fillStyle = "rgb(255,165,0)";  
ctx.fillStyle = "rgba(255,165,0,1)";
```

## ■ color颜色

□ color 可以是表示 CSS 颜色值的字符串，支持：关键字、十六进制、rgb、rgba格式。

□ 默认情况下，线条和填充颜色都是黑色（CSS 颜色值 #000000）。

## ■ 注意

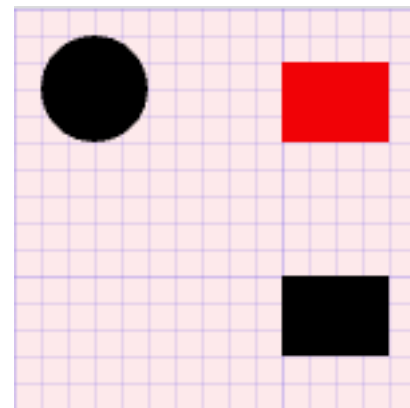
□ 一旦设置了 **strokeStyle** 或者 **fillStyle** 的值，那么这个新值就会成为新绘制的图形的默认值。

□ 如果你要给图形上不同的颜色，你需要重新设置 **fillStyle** 或 **strokeStyle** 的值。

## ■ 额外补充

□ **fill()** 函数是图形填充，**fillStyle**属性是设置填充色

□ **stroke()** 函数是图形描边，**strokeStyle**属性是设置描边色



# 透明度 Transparent

- 除了可以绘制实色图形，我们还可以用 canvas 来绘制半透明的图形。
- 方式一：strokeStyle 和 fillStyle 属性结合 RGBA：
- 方式二：globalAlpha 属性
  - globalAlpha = 0 ~ 1
    - ✓ 这个属性影响到 canvas 里所有图形的透明度
    - ✓ 有效的值范围是 0.0（完全透明）到 1.0（完全不透明），默认是 1.0。

```
// 设置透明度值
ctx.globalAlpha = 0.2;

// 画半透明圆
for (var i=0;i<7;i++){
  ctx.beginPath();
  ctx.arc(75,75,10+10*i,0,Math.PI*2,true);
  ctx.fill();
}
```

```
// 指定透明颜色，用于描边和填充样式
ctx.strokeStyle = "rgba(255,0,0,0.5)";
ctx.fillStyle = "rgba(255,0,0,0.5)";
```

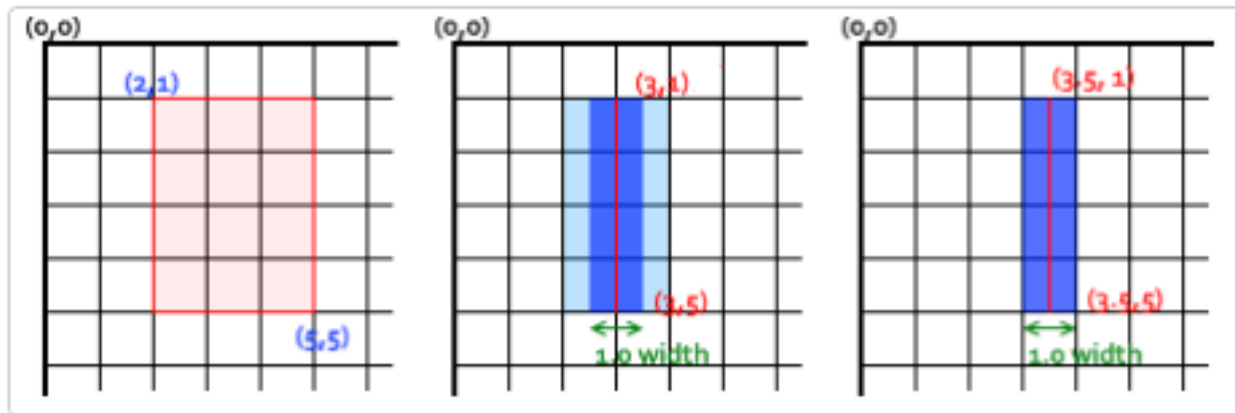
# 线型 Line styles

## ■ 调用lineTo()函数绘制的线条，是可以通过一系列属性来设置线的样式。

- lineWidth = value: 设置线条宽度。
- lineCap = type: 设置线条末端样式。
- lineJoin = type: 设定线条与线条间接合处的样式。
- .....

## ■ lineWidth

- 设置线条宽度的属性值必须为正数。默认值是 1.0px，不需单位。（零、负数、Infinity和NaN值将被忽略）
- 线宽是指给定路径的中心到两边的粗细。换句话说就是在路径的两边各绘制线宽的一半。
- 如果你想要绘制一条从 (3,1) 到 (3,5)，宽度是 1.0 的线条，你会得到像第二幅图一样的结果。
  - ✓ 路径的两边个各延伸半个像素填充并渲染出1像素的线条（深蓝色部分）
  - ✓ 两边剩下的半个像素又会以实际画笔颜色一半色调来填充（浅蓝部分）
  - ✓ 实际画出线条的区域为（浅蓝和深蓝的部分），填充色大于1像素了，这就是为何宽度为 1.0 的线经常并不准确的原因。
- 要解决这个问题，必须对路径精确的控制。如，1px的线条会在路径两边各延伸半像素，那么像第三幅图那样绘制从 (3.5, 1) 到 (3.5, 5) 的线条，其边缘正好落在像素边界，填充出来就是准确的宽为 1.0 的线条。



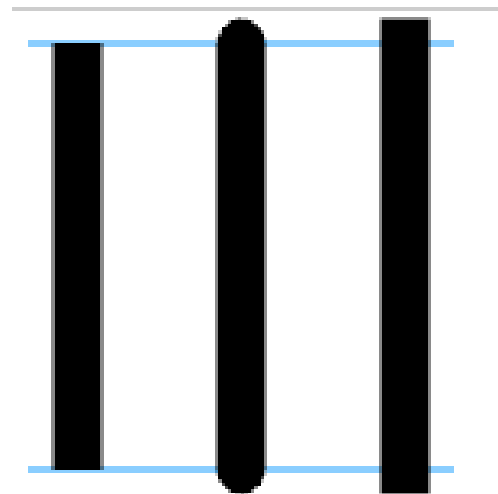
# 线型 Line styles

■ **lineCap**: 属性的值决定了线段**端点显示的样子**。它可以为下面的三种的其中之一:

- butt 截断, 默认是 butt。
- round 圆形
- square 正方形

■ **lineJoin**: 属性的值决定了图形中线段**连接处所显示的样子**。它可以是这三种之一:

- round 圆形
- bevel 斜角
- miter 斜槽规, 默认是 miter。





# 绘制文本

## ■ canvas 提供了两种方法来渲染文本：

### □ fillText(text, x, y [, maxWidth])

- ✓ 在 (x,y) 位置，填充指定的文本
- ✓ 绘制的最大宽度（可选）。

### □ strokeText(text, x, y [, maxWidth])

- ✓ 在 (x,y) 位置，绘制文本边框
- ✓ 绘制的最大宽度（可选）。

## ■ 文本的样式（需在绘制文本前调用）

□ font = value：当前绘制文本的样式。这个字符串使用 and CSS font 属性相同的语法。默认的字体是：10px sans-serif。

□ textAlign = value：文本对齐选项。可选的值包括：start, end, left, right or center. 默认值是 start

□ textBaseline = value：基线对齐选项。可选的值包括：top, hanging, middle, alphabetic, ideographic, bottom。

- ✓ 默认值是 alphabetic。



■ 绘制图片，可以使用 `drawImage` 方法将它渲染到 `canvas` 里。`drawImage` 方法有三种形态：

□ `drawImage(image, x, y)`

✓ 其中 `image` 是 `image` 或者 `canvas` 对象，`x` 和 `y` 是其在目标 `canvas` 里的起始坐标。

□ `drawImage(image, x, y, width, height)`

✓ 这个方法多了 2 个参数：`width` 和 `height`，这两个参数用来控制 当向 `canvas` 画入时应该缩放的大小

□ `drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`

✓ 第一个参数和其它的是相同的，都是一个图像或者另一个 `canvas` 的引用。其它 8 个参数最好是参照右边的图解，前 4 个是定义图像源的切片位置和大小，后 4 个则是定义切片的目标显示位置和大小。

■ 图片的来源，`canvas` 的 API 可以使用下面这些类型中的一种作为图片的源：

□ `HTMLImageElement`：这些图片是由 `Image()` 函数构造出来的，或者任何的 `<img>` 元素。

□ `HTMLVideoElement`：用一个 HTML 的 `<video>` 元素作为你的图片源，可以从视频中抓取当前帧作为一个图像。

□ `HTMLCanvasElement`：可以使用另一个 `<canvas>` 元素作为你的图片源。

□ ....

# Canvas绘画状态-保存和恢复

## ■ Canvas绘画状态

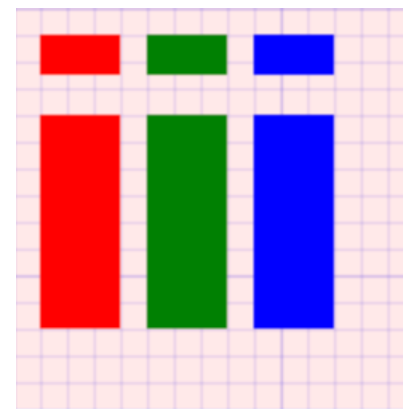
- 是当前绘画时所产生的**样式和变形**的一个快照。
- Canvas在绘画时，会产生相应的**绘画状态**，其实我们是可以将某些绘画的状态存储在栈中来为以后复用。
- Canvas 绘画状态的可以调用 **save 和 restore 方法**是用来**保存和恢复**，这两个方法都没有参数，并且**它们是成对存在的**。

## ■ 保存和恢复（Canvas）绘画状态

- save(): 保存画布 (canvas) 的所有绘画状态
- restore(): 恢复画布 (canvas) 的所有绘画状态

## ■ Canvas绘画状态包括:

- 当前应用的**变形**（即**移动，旋转和缩放**）
- 以及这些**属性**：**strokeStyle, fillStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, font, textAlign, textBaseline .....**
- 当前的**裁切路径** (clipping path)



# 变形 Transform

■ Canvas和CSS3一样也是支持变形，形变是一种更强大的方法，可以将坐标原点移动到另一点、形变可以对网格进行旋转和缩放。

■ Canvas的形变有4种方法实现：

□ `translate(x, y)`：用来移动 canvas 和它的原点到一个不同的位置。

✓ `x` 是左右偏移量，`y` 是上下偏移量（无需要单位），如右图所示。

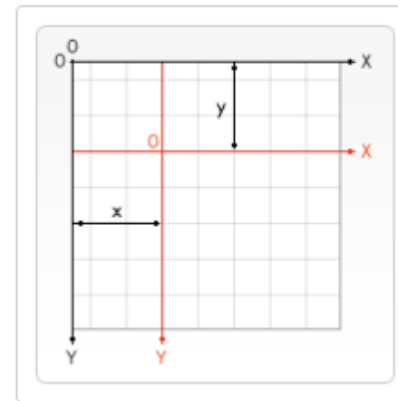
□ `rotate(angle)`：用于以原点为中心旋转 canvas，即沿着z轴旋转。

✓ `angle`是旋转的弧度，是顺时针方向，以弧度为单位。

□ `scale(x, y)`：用来增减图形在 canvas 中像素数目，对图形进行缩小或放大。

✓ `x` 为水平缩放因子，`y` 为垂直缩放因子。如果比 1 小，会缩小图形，如果比 1 大会放大图形。默认值为 1，也支持负数。

□ `transform(a, b, c, d, e, f)`：允许对变形矩阵直接修改。这个方法是将当前的变形矩阵乘上一个基于自身参数的矩阵。



■ 注意事项：

□ 在做变形之前先调用 `save` 方法保存状态是一个良好的习惯。

□ 大多数情况下，调用 `restore` 方法比手动恢复原先的状态要简单得多。

□ 如果在一个循环中做位移但没有保存和恢复canvas状态，很可能到最后会发现有些东西不见了，因为它很可能已超出canvas画布以外了。

□ 形变需要在绘制图形前调用。

# 移动-translate

## ■ translate方法，它用来移动 canvas 和它的原点到不同的位置。

- `translate(x, y)`

- ✓ `x` 是左右偏移量，`y` 是上下偏移量（无需单位）。

## ■ 移动 canvas 原点的好处

- 如不使用 `translate` 方法，那么所有矩形默认都将被绘制在相同的  $(0,0)$  坐标原点。

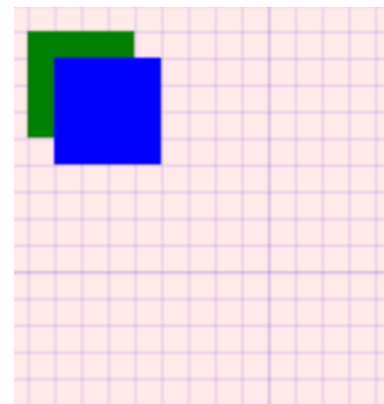
- `translate` 方法可让我们任意放置图形，而**不需要手工一个个调整坐标值**。

## ■ 移动矩形案例

- 第一步：先保存一下 canvas 当前的状态

- 第二步：在绘制图形前 `translate` 移动画布

- 第三步：开始绘制图形，并填充颜色

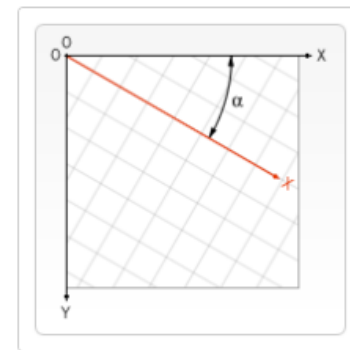


# 旋转-rotate

■ rotate方法，它用于以原点为中心旋转 canvas，即沿着 z轴 旋转。

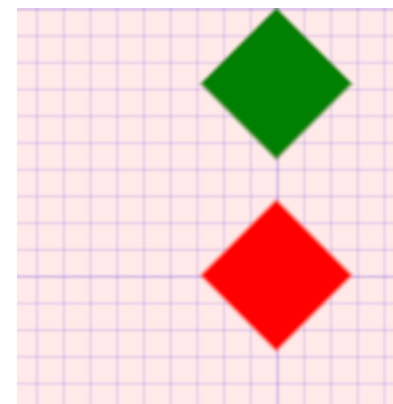
□ rotate(angle)

- ✓ 只接受一个参数：旋转的角度 (angle)，它是顺时针方向，以弧度为单位的值。
- ✓ 角度与弧度的 JS 表达式：弧度 =  $(\text{Math.PI} / 180) * \text{角度}$ ，即 1角度 =  $\text{Math.PI}/180$  个弧度。
- ✓ 比如：旋转90°： $\text{Math.PI} / 2$ ；旋转180°： $\text{Math.PI}$ ；旋转360°： $\text{Math.PI} * 2$ ；旋转-90°： $-\text{Math.PI} / 2$ ；
- ✓ 旋转的中心点始终是 canvas 的原坐标点，如果要改变它，我们需要用到 translate方法。



## ■ 旋转案例

- 第一步：先保存一下Canvas当前的状态，并确定旋转原点
- 第二步：在绘制图形前旋转画布（坐标系会跟着旋转了）
- 第三步：开始绘制图形，并填充颜色



# 缩放-scale

■ **scale方法可以缩放画布。**可用它来增减图形在 canvas 中的像素数目，**对图形进行缩小或者放大。**

□ scale(x, y)

- ✓ x 为水平缩放因子，y 为垂直缩放因子，也支持负数。
- ✓ 如果比 1 小，会缩小图形，如果比 1 大会放大图形。默认值为 1。

## ■ 注意事项

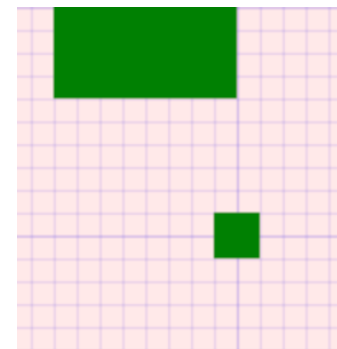
□ **画布初始情况下，是以左上角坐标为原点。**如果参数为负实数，相当于以 x 或 y 轴作为对称轴镜像反转。

- ✓ 例如，使用translate(0, canvas.height); scale(1,-1); 以 y 轴作为对称轴镜像反转。

□ 默认情况下，canvas 的 1 个单位为 1 个像素。如果我们设置缩放因子是 0.5，1 个单位就变成对应 0.5 个像素，这样绘制出来的形状就会是原先的一半。同理，设置为 2.0 时，1 个单位就对应变成了 2 像素，绘制的结果就是图形放大了 2 倍。

## ■ 缩放案例实战

- 第一步：先保存一下Canvas当前的状态，并确定缩放原点
- 第二步：在绘制图形前缩放画布
- 第三步：开始绘制图形，并填充颜色



## ■ Canvas绘图都是通过JavaScript 去操控的，如要实现一些交互性动画是相当容易的。那Canvas是如何做一些基本动画的？

- canvas可能最大的限制就是图像一旦绘制出来，它就是一直保持那样了。
- 如需要执行动画，**不得不对画布上所有图形进行一帧一帧的重绘**（比如在1秒绘60帧就可绘出流畅的动画了）。
- 为了实现动画，我们需要一些可以定时执行重绘的方法。然而在Canvas中有三种方法可以实现：
  - ✓ 分别为 **setInterval**、**setTimeout** 和 **requestAnimationFrame** 三种方法来定期执行指定函数进行重绘。

## ■ Canvas 画出一帧动画的基本步骤（如要画出流畅动画，1s 需绘60帧）：

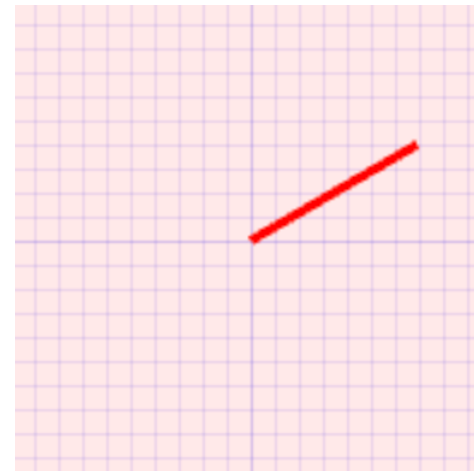
- 第一步：**用 clearRect 方法清空 canvas**，除非接下来要画的内容会完全充满 canvas（例如背景图），否则你需要清空所有。
- 第二步：**保存 canvas 状态**，如果加了 canvas 状态的设置（样式，变形之类的），又想在每画一帧之时都是原始状态的话，你需要先保存一下，后面再恢复原始状态。
- 第三步：**绘制动画图形**（animated shapes），即绘制动画中的一帧。
- 第四步：**恢复 canvas 状态**，如果已经保存了 canvas 的状态，可以先恢复它，然后重绘下一帧。



# 绘制秒针- setInterval

## ■ 绘制秒针动画，绘制一帧的步骤：

- 第一步：用 `clearRect(x,y, w,h)`方法，清空 canvas 。
- 第二步：保存 canvas 状态 。
- 第三步：修改 canvas 状态 （样式、移动坐标、旋转等） 。
- 第四步：绘制秒针图形（即绘制动画中的一帧） 。
- 第五步：恢复 canvas 状态 ，准备重绘下一帧。



## ■ setTimeout定时器的缺陷

- setTimeout定时器不是非常精准的，因为setTimeout的回调函数是放到了宏任务中等待执行。
- 如果微任务中一直有未处理完成的任务，那么setTimeout的回调函数就有可能不会在指定时间内触发回调。
- 如果想要更加平稳和更加精准的定时执行某个任务的话，可以使用requestAnimationFrame函数。

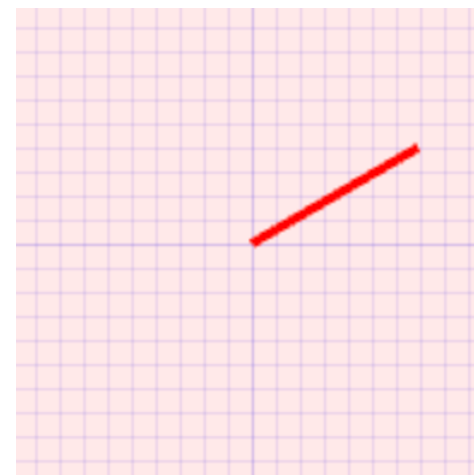
# 绘制秒针-requestAnimationFrame

## ■ requestAnimationFrame函数

- ❑ 告诉浏览器——你希望执行一个动画，并且要求浏览器在下次重绘之前调用该函数的回调函数来更新动画。
- ❑ 该方法需要传入一个回调函数作为参数，该回调函数会在浏览器下一次重绘之前执行
- ❑ 若想在浏览器下次重绘之前继续更新下一帧动画，那么在回调函数自身内必须再次调用 requestAnimationFrame()
- ❑ 通常每秒钟回调函数执行 60 次左右，也有可能被降低。

## ■ 绘制秒针动画，绘制一帧的步骤：

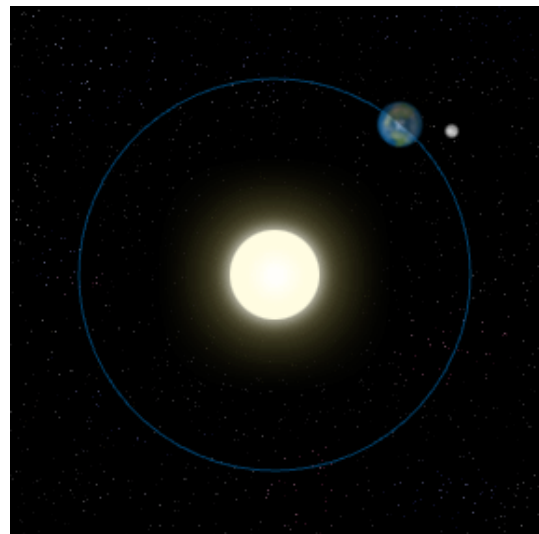
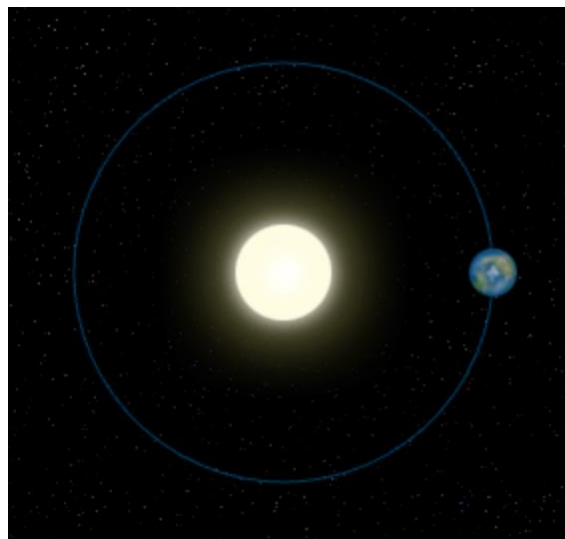
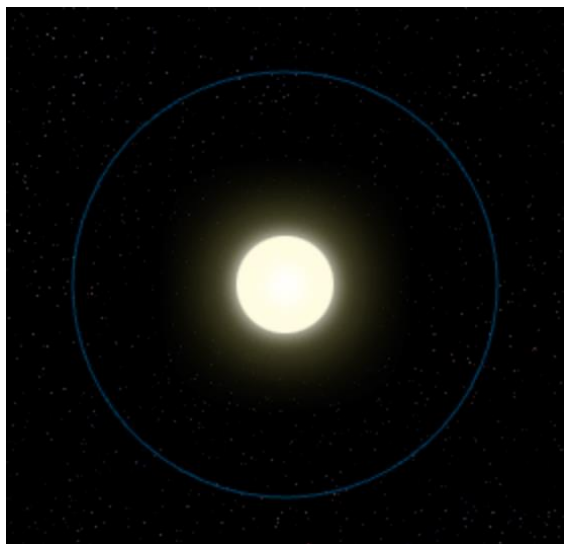
- ❑ 第一步：用 `clearRect(x,y, w,h)`方法，清空 canvas 。
- ❑ 第二步：保存 canvas 状态 。
- ❑ 第三步：修改 canvas 状态（样式、移动坐标、旋转等）。
- ❑ 第四步：绘制秒针图形（即绘制动画中的一帧）。
- ❑ 第五步：恢复 canvas 状态，准备重绘下一帧。



# 太阳系旋转

## ■ 太阳系旋转动画，绘制一帧的步骤：

- 第一步：用 `clearRect(x,y, w,h)`方法，清空 canvas ，并初始化全局样式。
- 第二步：保存 canvas 状态。
- 第三步：绘制背景、绘制地球（绘制月球）、绘制阴影效果。
- 第五步：恢复 canvas 状态，准备重绘下一帧。



## ■ 求圆上x, y的坐标:

□ 圆上x, y轴坐标实际上就是右图的 ( AB, BC ), AC为时钟半径

□  $x = AB = \cos a * AC \Rightarrow x = \text{Math.cos}(\text{弧度}) * R$

□  $y = BC = \sin a * AC \Rightarrow y = \text{Math.sin}(\text{弧度}) * R$

□ 角度与弧度的 JS 表达式:  $\text{弧度} = (\text{Math.PI} / 180) * \text{角度} \Rightarrow \text{弧度} = 1\text{角度对应的弧度} * \text{角度}$ 。

□ 比如: 旋转90°: 弧度为  $\text{Math.PI} / 2$ ; 旋转180°: 为  $\text{Math.PI}$ ; 旋转360°: 为  $\text{Math.PI} * 2$ ; 旋转-90°: 为  $-\text{Math.PI} / 2$ ;

□ 第 i 小时的坐标:

✓  $x = \text{Math.cos}(\text{Math.PI} * 2 / 12 * i) * R$

✓  $y = \text{Math.sin}(\text{Math.PI} * 2 / 12 * i) * R$

## ■ 绘制时钟, 绘制一帧的步骤:

□ 第一步: 用 `clearRect(x,y, w,h)`方法, 清空 canvas。

□ 第二步: 保存 canvas 状态。

□ 第三步: 绘制白背景、绘制数字、绘制时/分/秒针、绘制圆、绘制时分刻度。

□ 第四步: 恢复 canvas 状态, 准备重绘下一帧。

