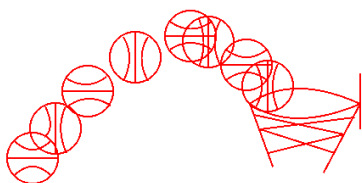


1 初识 Canvas 画布

1.1 引例描述



薛总工在小新完成的前面几个项目的基础上,要求小新学习用 Canvas 绘制体育动画的功能,一方面体现热爱运动,热爱生命的寓意,另一方面希望小新能够通过交互式的方法进行终端应用开发并掌握 HTML5 的画布功能。当然该项目的原理还可以满足用户不同的需要,提升网站档次,黏住用户带来流量。每次登录后有一个与众不同的动画为你而来,用户难道不想常回去看看吗。

1.2 任务陈述

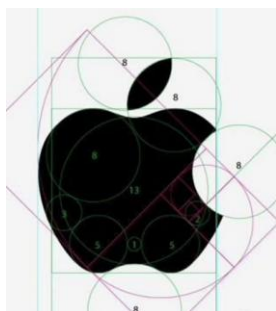
表 3-1 学习任务单

一、达成目标
1. 掌握选择器的用法、理解盒子模型完成示例页面；
2. 能够根据学到的 CSS 知识重构个人主页；
二、学习任务
1.掌握 CSS 样式表结构和引用方式；
2.对 HTML 布局、框架标签进行操作，可以在 Chrome 浏览器中查看实现效果；
3.以小组的方式对完成的页面进行讨论，记录未能成功修改的需求与教师讨论；
4.结合自身兴趣，制订针对 HTML5 技术的学习计划；
三、参考资源
1. 在智慧职教平台 http://www.icve.com.cn/portal/ 注册并学习课程；
2. 搜索和学习 API,可参考 DevDocs API Documentation http://devdocs.io/css/ ；
三、困惑与建议
将遇到的难点和不清晰的地方通过截屏等方式记录，准备与教师面对面讨论问题。

本单元将通过学习 HTML5 的最新特性 Canvas（画布）实现在网页上绘制一个投篮效果。任务实施的步骤如下：

- 第一步：HTML 基本结构文档的编写；
- 第二步：使用 JavaScript 编写绘制动画帧的方法；
- 第三步：使用 setInterval 方法周期性地执行绘制帧的方法。

1.3 知识准备



在 HTML5 中什么是 Canvas ? Canvas 译为画布或画板,其功能和现实生活中的画布或画板功能一样,不过它是用在网页中进行绘画还需要学习一些它的操作方法 (API) 以及概念知识。

Canvas 在 HTML4 以前并不存在,在 HTML5 中新增加的成员的其中之一就包含了 Canvas,但这并不代表它是一个非常新颖的技术。Canvas 最先在苹果公司(Apple)的 Mac OS X Dashboard 上被引入,然后被应用于 Safari 浏览器。随后在 Delphi、VB、VC 和 C++ Builder 等众多桌面应用程序开发工具中,也相继引入了 Canvas 画布技术的应用。

1.3.1 知识点 1 Canvas 绘图情景



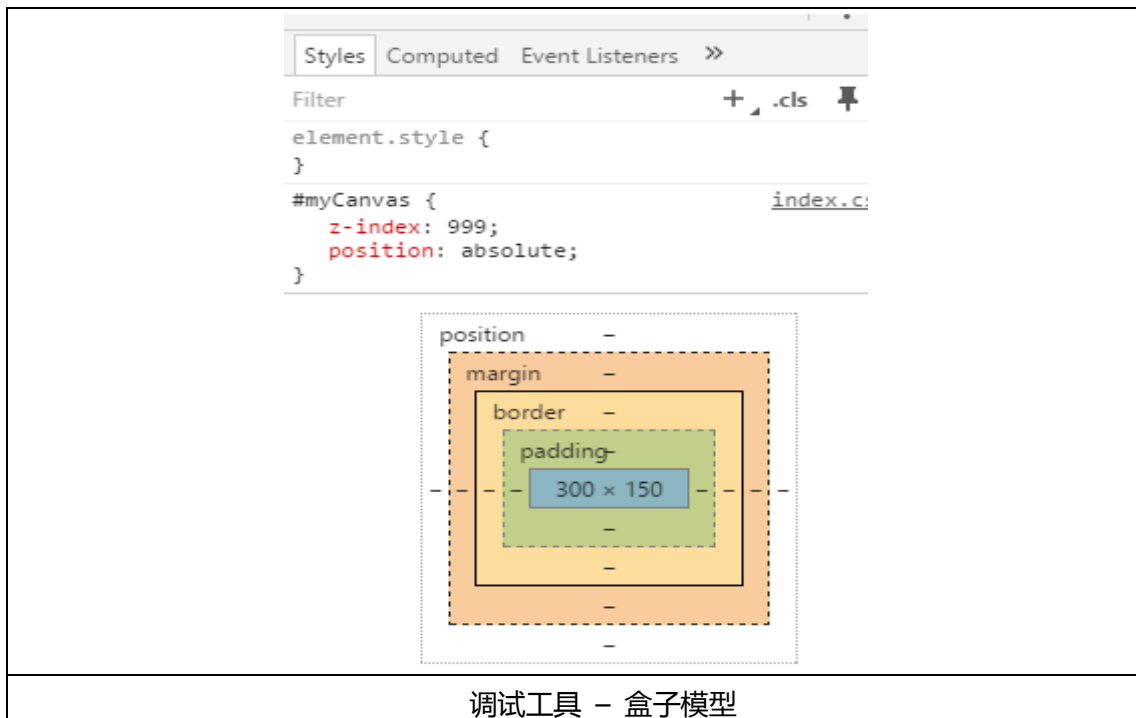
想熟练使用 Canvas 完成各种绘画操作不仅要具备基本 HTML 和 JavaScript 知识还要掌握它们的操作方法。但最重要的一点是你需要一个可以支持 Canvas(HTML5)的浏览器才可以,比如 IE10、Chrome V28、Opera15 等。HTML5 为我们提供了 Canvas 标签,可以直接作为工具使用,还可以通过 JavaScript 在过程中动态创建。使用 Canvas 绘制图形的步骤如下:

1. 创建 HTML 页面,设置画布标签;
2. 编写 js,获取画布 DOM 对象;
3. 通过 Canvas 标签的 DOM 对象获取上下文;
4. 设置绘制线样式、颜色;
5. 绘制图形,或者填充图形。

要使用 Canvas 来绘制图形必须在页面中添加 Canvas 的标签。以标签的方式进行使用较为简单,在 HTML 文件中定义一个 Canvas 标签的代码如下:

<code><canvas id="myCanvas"> </canvas></code>
画布标签

既然已经在 HTML 中创建了 Canvas 内容标签,可以在浏览器中看到吗?虽然上图代码没有给定宽度和高度属性,但实际上 Canvas 在页面中是**可见的**,因为它是默认宽度和高度分别为 300px 和 150px,背景颜色**透明的**,那又有些同学可能会说为什么不是白色?第一个原因当浏览器没有任何样式时,浏览器页面为默认背景颜色-白色,那为了证明它是透明的,可以把<body>标签中的背景颜色改为黑色,可以看到 Canvas 的颜色也跟着变为了黑色。第二个原因可以设置一个有颜色的标签,让该标签和<canvas>标签重叠,这时可以看到 canvas 标签并没有挡住背景颜色的标签。通过这两个判定可认为 canvas 标签没有背景颜色的,是透明的。


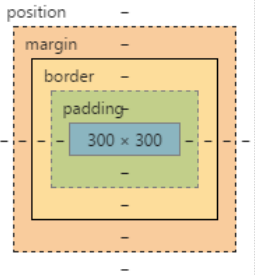


调试工具 - 盒子模型

既然有默认的宽度和高度,又如何去改变 Canvas 的默认大小呢?运用我们前面学习的知识点来解决这个问题,比如会通过 document 的选择器、外部 CSS 文件样式设置样式属性等方法。但是这样的结果是正确的吗?先试试看。

```
... 略  
<script>  
document.getElementById('myCanvas').style.cssText =  
"width:300px; height: 300px";  
var canvas_1 = document.getElementById("myCanvas");  
var ctx = canvas_1.getContext("2d");  
ctx.beginPath();  
ctx.moveTo(0,0);  
ctx.lineTo(100,0);  
ctx.lineTo(0, 200);  
ctx.closePath();  
ctx.fillStyle = "green";  
ctx.fill();  
</script>  
... 略
```


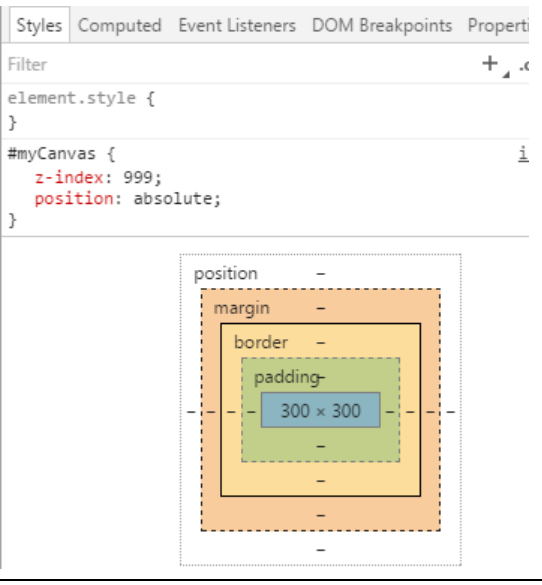
上图代码中 canvas 相关的绘制方法,这些方法在后面会讲解到,先看下效果如何。

	<pre>element.style { width: 300px; height: 300px; } #myCanvas { z-index: 999; position: absolute; }</pre> 
调试工具 - 参数信息查看	

从效果图来看，前文代码确实改变了 canvas 的宽度和高度，这点可以从调试工具中的盒子模型看出，但从效果图来看起来奇怪，变成了一个奇怪的形状，像是图片被强行拉申过的形状。试下另外一种修改方法，它是官方文档中的使用方法，看下效果如何，代码如下：

<pre><canvas id="myCanvas" width="300" height="300"></canvas> <script> //去掉 document 选择器操作方法 //原有 canvas 操作方法不变 </script></pre>

重新刷新浏览器后，canvas 的效果就变成了以下情况。

	
调试工具 - 参数信息查看	

从效果图上看，确实是一个倒直角三角形，那这种修改方法和上一种修改方法有什么不一样？后这一种方法是正确的修改方式，它修改的是 Canvas 标签中的属性，而前一种方法修改的是 Canvas 标签 CSS 样式的属性，在本方法中可以看到它是通过 width=300 和

height=300 的方式修改，也就是说 canvas 标签和其它标签**最大的不同是**，它带有 width 和 height 属性，比如大部分标签都带有 alt、title、onclick 属性。

前文说的通过外部 CSS 文件样式改变是否可以呢？它的效果和 dom 选择器修改后的效果一致。

Canvas 上下文 (Context) :

回到上文中已经设置好了 canvas 标签的宽度和高度，但存在一个问题，通过 dom 选择器获得的对象是一个元素节点对象并不是一个具体的绘制对象，canvas 元素自身属性与常规的 HTML 元素没有多大区别，我们应该如何找到 canvas 元素的绘制对象呢？

当前 canvas 的绘图操作方法都定义在一个 CanvasRenderingContext2D 对象上（称为上下文对象），该对象通过 getContext()方法获得，以下是代码示例：

```
var canvas = document.getElementById("canvas");  
var ctx = canvas.getContext("2d");  
console.log(ctx);
```

通常把这个绘制对象称为 canvas 的上下文 (Context)，它包含了所有绘图操作所需的属性设置和操作方法，把它理解为一个绘制环境对象也可以很好的描述它的作用。

刷新浏览器，如果可以在调试工具中看到输出对象，则表示已经对象已经获取到。但是请注意 getContext()中的参数，“2d”而不是“2D”。未来会有 supportsContext()等方法，让我们可以使用 3d 的画布。

得到了具体的环境，也通过调试工具发现该环境是可以使用的，就可以使用该情境（Context）来绘制图形。Context 属于 canvas 节点对象，情境中包含了许多操作方法。

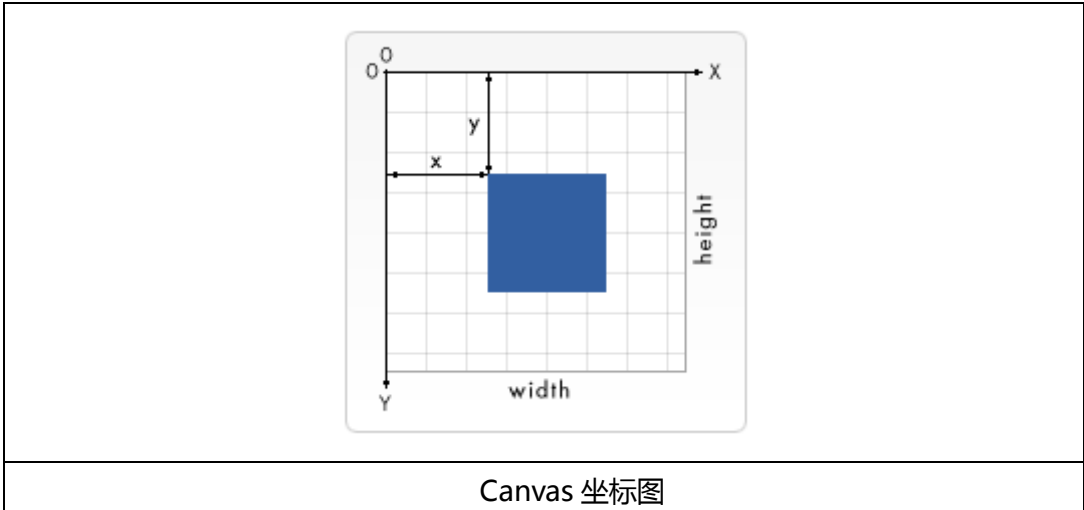
1.3.2 知识点 2 Context 操作方法



Canvas 中的 Context 操作方法，从易到难可分为以下几种类型：绘制图形，比如矩形、三角形、多边形、曲线等；渲染图像，缩放图片、渲染图片、裁剪图片等；应用颜色，样式，阴影和变形等等。

绘制图形：

在进行绘制图形之前，先要理解 Canvas 中的坐标系，在前文的代码中 Canvas 标签的宽度和高度为 300，现在在画布上叠加一个网格，进行说明。



通常网格的 1 个单元对应 canvas 上的 1 个像素。网格的原点是定位在左上角（坐标 (0,0) ）。画面里的所有物体的位置都是相对这个原点。这样，左上角的蓝色方块的位置就是距左边 x 像素和距上边 Y 像素（坐标(x, y) ）。后面的教程中将会介绍如何把移动原点，旋转以及缩放网格。我们首先学习如何绘制直线。

绘制线条：

既然要画一条线，有几个必要条件呢？让线条可见需要：线的起点、线的终点、线的颜色，线的粗细（宽度）。如果确定了以上信息，应该如何去绘制呢？绘制线条的具体方法，如下表所示：

方法名字	说明
beginPath()	开始定义路径
closePath()	关闭定义的路径，连接起始点和结束点，通常以上 2 个方法成对出现
moveTo(x, y)	把 canvas 中当前路径的结束点移动到 x , y 点
lineTo(x, y)	把 canvas 中当前路径的结束点连接到 x , y 点

需要注意的是上表中的 moveTo 和.lineTo 方法，moveTo 方法可以理解为把当前在 canvas 中的坐标点移动到指定的坐标点中，比如以下代码：

```
... 略
ctx.beginPath();
ctx.moveTo(50, 50);
ctx. closePath();
... 略
```

前文说过 canvas 默认的坐标是从左上角（0，0）开始，如果没有调用任何方法，上图代码的意思是把 canvas 的坐标（0，0）移动到坐标（50，50）。如果没有其它代码，当前

环境下的 canvas 坐标为 (50 , 50)。

而 lineTo 方法则是将当前的 canvas 坐标和指定的坐标点连接起来 ,还是继续使用上图中的代码做说明 :

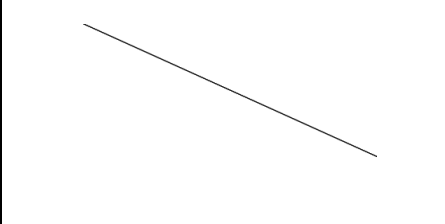
```
... 略
ctx.beginPath();
ctx.moveTo(50, 50);
ctx.lineTo(100, 100);
ctx.closePath();
... 略
```

上图中的加粗代码是新增加的 , canvas 坐标点从 (0 , 0) 移动到 (50 , 50) 接下来执行新增加的代码 lineTo()方法后 , 将坐标 (50 , 50) 点连接到坐标 (100 , 100) 点上。这样确定两点坐标后 , 就可以绘制一条直线了。

现在刷新浏览器 , 看下效果如何 , 为什么图形没有出来 ? 还记得前文中提到的绘制一条直线需要的几个必要条件吗 ? 现在已经有的条件有线的起点和终点 , 缺少颜色、宽度、是否提交绘画操作。所以还需要借助其它方法配合 canvas 完成绘画的请求操作。

方法名字	说明
fill();	填充当前 canvas 闭合区域路径
stroke();	填充当前 canvas 路径 (类似 PhotoShop 的 描边)

在代码末尾添加 strokeStyle()方法之后 , 刷新浏览器后就可以看到线条了。

ctx.beginPath(); ctx.moveTo(0,0); ctx.lineTo(700,400); ctx.closePath(); ctx.strokeStyle();	

上图效果中的线条好丑怎么办 ? 想要画出个性化的线条应该如何做 ? Context 提供了相关的方法 , 如下表所示 :

方法名字	说明
fillStyle()	设置 填充 canvas 路径的属性值
strokeStyle()	设置 绘制 canvas 路径的属性值

方法分别都支持以下三个属性值 :

符合颜色格式的字符串值 , 表示使用纯色填充 , 比如"red";。

CanvasGradient , 表示使用渐变填充。

CanvasPattern , 表示使用位图填充。

除此之外 , 还可以设置线的粗细 (宽度) , 方法如下 :

方法名字	说明
lineWidth()	设置线的 宽度

通过以上几个方法，我们可以绘制出各种各样的线条了。

绘制三角形：

在前文中已经学会了如何创建一条直线，但是在 Canvas 开发中不仅仅只是绘制一条直线这么简单，比如如何绘制一个三角形呢？三角形要有三个点和三条线，需要利用前文所学的 canvas 操作方法来**移动坐标点**和**连接两个坐标点之间的线**。


```
ctx.beginPath();  
//移动坐标点  
ctx.moveTo(0, 0);  
//设置绘制下一个点  
ctx.lineTo(100, 0);  
//设置绘制下一个点  
ctx.lineTo(0, 200);  
ctx.closePath();  
ctx.fill();
```

刷新浏览器，可看到倒直角三角形已经被绘制出来了。上图代码中可以看到三个点 0,0、100,0、0, 200。两条线（两次 lineTo 方法），这跟前文所说的三个点和三条线有所区别，但为什么它还是可以被绘制出来？

还记得前文中对 **closePath()** 的描述吗？**关闭定义路径，并连接起始点和结束点**。回头看下上图中的代码，起始坐标点为 0,0，canvas 最后一次连接点为 0,200。调用 closePath 使这两点连接起来了（效果相当于 lineTo()），并且结束路径绘制，等于完成了第三边的操作。这时候上图代码完成了三个点移动和三条边的连接工作。


通过以上学习，应该有跃跃欲试的想法吧。利用这些特性可以完成正方形，矩形，多边形，不规则形状等操作。

使用点和线画一个正方形，正方形具有四个点和四条边，假设四个点坐标分别为 (0,0)，(100,0)，(100,100)，(0,100)。接着把这四个点分别连接起来，连接起来后会形成一个路径闭合的区域，接着填充颜色即可。

<pre>ctx.beginPath(); ctx.moveTo(0,0); //坐标点 1 ctx.lineTo(100, 0); //坐标点 2 ctx.lineTo(100, 100); //坐标点 3 ctx.lineTo(0, 100); //坐标点 4 ctx.closePath(); ctx.stroke();</pre>	

以上图代码所示比绘制三角形多出了一点坐标点，从三角形变为了正方形。代码中使用了 **stroke()** 方法填充坐标点连接的路径（默认情况下坐标点连接路径是无法看到的），默认颜色为黑色，当使用 **fill()** 方法之后，它会填充上图中黑色路径包围的区域，这个区域称为

闭合区域，填充完的颜色也是默认颜色。这个就是 fill 和 stroke 的区别，可能会有人同时调用这两个方法，看起来好像也没有什么冲突，当前文提到的 fillStyle 和 strokeStyle 的属性不一致时它们的区别可以看出来。

<pre>... 略 ctx.fillStyle = "green"; ctx.strokeStyle = "red"; ctx.stroke(); ctx.fill();</pre>	

绘制一个正方形需要 7-9 行代码，假设某任务中需要大量绘制不同大小、不同颜色的正方形，有没有减少工作量的方法呢？我们可以运用前面单元（第五单元知识点 2）中所学的封装思想来封装各种使用频率较高的方法。封装好的方法胜在使用方便、简单、不需要重复写大量相同的代码，比如 beginPath 和 closePath，只需要创建对象，调用对应方法即可，也不必关心具体参数和方法之间的关系。这种设计方法，称为装饰者模式，在不必改变原类文件和使用继承的情况下，动态地扩展一个对象的功能是经典设计模式之一。在原有对象的基础上包装一层新的对象，这层新的对象是对原有对象中的属性和方法的增强实现。



```
function CanvasWrap(context){
  this.ctx = context;
}

//填充方形函数
CanvasWrap.prototype.fillSquare = function(x, y, width, height){
  this.ctx.beginPath();
  this.ctx.moveTo(x, y);
  this.ctx.lineTo(width, y);
  this.ctx.lineTo(width, height);
  this.ctx.lineTo(x, height);
  this.ctx.closePath();
};


var wrap = new CanvasWrap(ctx);
wrap.fillSquare(0,0,100,100);
```

比如上图中的代码，Canvas 中的 Context 对象提供简单的点、线、绘制路径的相关方法，但它没有一个具体形状的方法，刚好实验得出通过几个不同的代码操作可以绘画一个图形，于是把这些代码包装为一个方法，更进一步把它设计为一个装饰对象，只需要创建一个对象，调用一个方法，canvas 标签就绘制了一个图形，这是封装的魅力。

比如在后期版本更新时，fillSquare 函数需要添加一个新的功能，只需找到某个对象中的某个函数修改代码即可，所有应用该函数的地方都会做出改变：

<pre>//填充方形函数 CanvasWrap.prototype.fillSquare = function(x, y, width, height, fill){ this.ctx.beginPath(); this.ctx.moveTo(x, y); this.ctx.lineTo(width, y); this.ctx.lineTo(width, height); this.ctx.lineTo(x, height); this.ctx.closePath(); fill ? this.ctx.fill() : this.ctx.stroke(); }; ctx.fillStyle = "#ff6666" var wrap = new CanvasWrap(ctx); wrap.fillSquare(0,0,100,100, true);</pre>	
	
wrap.fillSquare(0,0,100,100, true)	wrap.fillSquare(0,0,100,100, false)

Canvas 也提供和上图中类似的方法来绘制一个矩形，使用方法如下：

<pre>...略 ctx.fillRect(10, 10, 100, 100);</pre>


canvas 提供的方法就更简单了，也不需要封装，不需要知道坐标点，直接通过 Context 就可以使用了。但它的功能性没有前文中封装代码灵活，封装方法可以根据需求随时改变绘制路径。

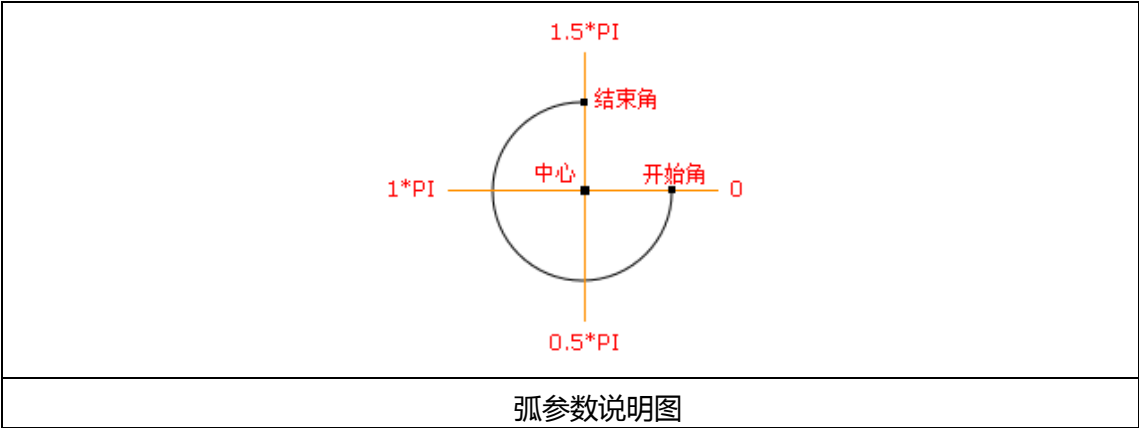
通过以上介绍，大家可能已经绘制出来各种形状了，但还有曲线形状无法绘制，前面介绍的方法只能直来直去。

绘制圆形/弧线：

Canvas 提供的 Context 中有相关绘制弧线的操作方法，它所需要的必要参数和前文介绍画点、线不同，这里有个概念需要说明，Context 方法中提供的 **arc()** 方法只可以绘制弧线，不能绘制圆形，但本节标题提到的绘制圆形又是怎么回事呢？当绘制弧线的**结束点数值大于开始点数值并且开始点和结束点重叠时**，它就形成一个闭合路径，这一个闭合路径的区域填充颜色后就是一个圆形。所以关于 arc() 方法的参数说明将带入圆的概念：

参数名字（顺序说明）	参数说明
x	圆点坐标 X
y	圆点坐标 Y
radius	圆半径
startAngle	圆弧点起始弧度（注意是弧度，不是角度）
endAngle	圆弧点结束弧度（注意是弧度，不是角度）
anticlockwise	顺时针（false）或逆时针（true）

使用下图来进行说明表中的 startAngle 和 endAngle。



前文表中的 startAngle 是上图中的开始角，endAngle 是上图中的结束角，x 和 y 是上图中的中心点坐标，以中心点到弧线的距离为半径对应的是 radius。不管是顺时针还是逆时针，开始角（startAngle）都是以圆形的三点钟位置开始。

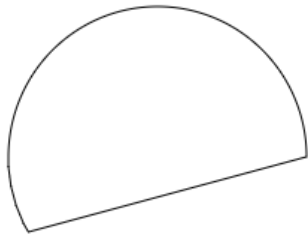
在上图中和上表中有两个参数的单位需要注意，startAngle 和 endAngle 它们是以弧度为单位而不是度为单位。度和弧度这两个单位值可以使用公式进行转换，公式代码：

<code>var radians = (Math.PI / 180) * degree;</code>
弧度和度转换公式

再次运用前面单元的所学封装方法，把该转换公式封装为一个工具方法，该工具对象可命名为 Utils，函数名为 computeRadians，代码实现如下：

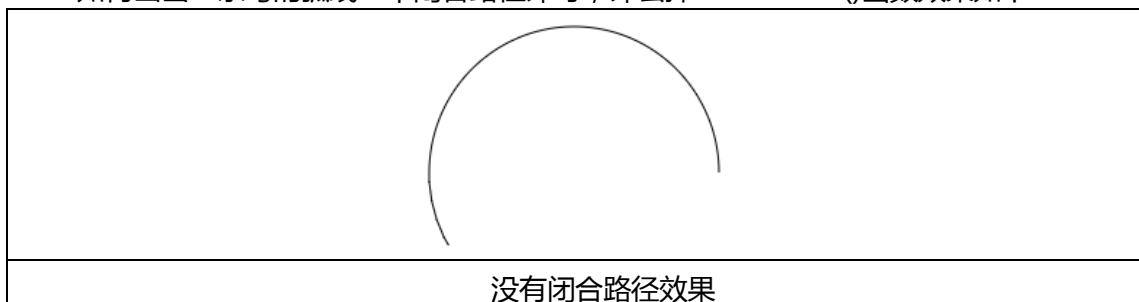
<pre>//新建一个 JavaScript 文件，命名为 utils.js var Utils = { /**弧度转换度*/ computeRadians: function (degree) { return (Math.PI / 180) * degree; } };</pre>
弧度转换函数

继续使用 arc()方法绘制圆和弧线，首先是弧线，以下代码会运用上图中工具对照中的方法。

<pre> ... 略 ctx.beginPath(); //arc() 参数说明 //200 =x 坐标 , 300 = y 坐标 , 100 = radius // Utils.computeRadians(0) = startAngle // Utils.computeRadians(150)=endAngle //true = anticlockwise ctx.arc(200, 300, 100, Utils.computeRadians(0), Utils.computeRadians(150), true); ctx.closePath(); ctx.stroke(); </pre>	
--	--

刷新浏览器，canvas 绘制出来结果却是一个奇怪的图形，它并不是一个弧线，不符合预想的效果，那为什么会出现这种奇怪的图形？如果对前文中介绍的方法理解透了就会明白，那本文在进行一次说明。还记得前文中对 **closePath()** 的描述吗？**关闭定义路径，并连接起始点和结束点。**

如何画出一条弯的弧线？不闭合路径即可，即去掉 closePath() 函数效果如下：

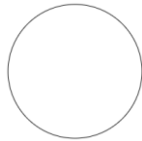


前文说过 beginPath 和 closePath 一般成对出现，上图为了可以绘制出弧线去掉了 closePath 函数，需要注意的是当前绘制路径还未结束（还没有调用 closePath），它当前坐标点是弧线的结束角。

现在已经知道如何绘制一条弧线，还记得前文中介绍的条件如何绘制一个圆吗？当绘制弧线的**结束点数值大于开始点数值并且开始点和结束点重叠**时，它就形成一个闭合路径。这里说的条件可分为两点：

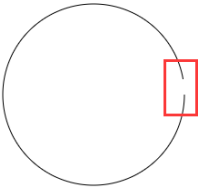
- 1、其中一个数值一定要大于另外一个。
- 2、满足第一个条件后，开始点和结束点重叠。

其中上文中的条件 2，有点难以理解就以代码的方式来说明：

<pre> ctx.beginPath(); ctx.arc(200, 300, 100, Utils.computeRadians(0), Utils.computeRadians(360), false); ctx.closePath(); ctx.stroke(); </pre>	
---	---

--	--

上图代码中参数满足条件 1，开始角大于结束角，同时也满足条件 2,开始角和结束角重叠在一起了。再对上图中代码的参数进行改变，可以看到开始角和结束角。

<pre> ctx.beginPath(); ctx.arc(200, 300, 100, Utils.computeRadians(0), Utils.computeRadians(350), false); // ctx.closePath(); ctx.stroke(); </pre>	

上图中的开始角和结束角差一段距离就可以完全的重叠在一起了，如果这时直接调用 cloasePath 函数，它会自动将开始角和结束角重叠在一起一个圆形的闭合路径。

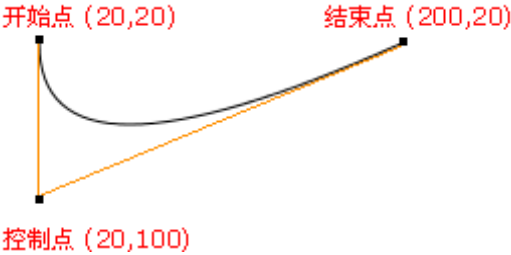
本节中讲解了如何绘制弧线路径 ,通过绘制弧线路径造成一个闭合区域来间接的完成绘制圆的需求。在后面小节中将讲解如何绘制曲线。

绘制贝塞尔曲线：

同样是曲线，那本文中的曲线和前文中的曲线（弧线）有哪些不一样？不同点有以下几个部分：

- 1、本节中的曲线被称为贝塞尔曲线又称为贝兹曲线或贝济埃曲线，是应用于二维图形应用程序的数学曲线。
- 2、贝塞尔曲线由**线段和节点**组成，线段又包含开始点和结束点。节点是一个可以被拖动的节点，线段像可伸缩的橡皮筋，它根据节点情况来改变伸缩大小。

上文所述贝塞尔曲线有一个重要特点，它的线段是根据节点（本文中把它称为控制点）进行改变的，它的曲线弧度可以实时变化，不断重复组合，可以绘制出复杂的图形。





贝塞尔曲线参数说明

上图中的开始点和结束点决定了线段的长度，控制点决定了线段的弯曲大小，接下来介绍它的使用方法。在 HTML5 的 Canvas 中提供了绘制**二次贝塞尔曲线**和绘制**三次贝塞尔曲线**的函数，本节中先讲解如何创建二次贝塞尔曲线。在 JavaScript 它的语法如下：

ctx.quadraticCurveTo(cpx,cpy,x,y)	
二次贝塞尔曲线语法	
函数中包含四个参数，这四个参数的用途使用下表作为描述：	
参数名字	说明
cx	二次贝塞尔控制点的 x 坐标

cy	二次贝塞尔控制点的 y 坐标
x	结束点的 x 坐标
y	结束点的 y 坐标


似乎上表中的参数少了一些和前文中图例所描述的不太一样，缺少**开始点**，但贝塞尔的开始点不是由该方法指定的，而是由 moveTo()方法指定，如果没有指定坐标点，请注意该贝塞尔曲线函数将失去作用。

<pre> ctx.beginPath(); ctx.moveTo(20, 30); ctx.quadraticCurveTo(20,130 , 100, 130); //ctx.quadraticCurveTo(100,130 , 100, 130); ctx.stroke(); </pre>		
		
加入控制点的贝塞尔	没有控制点的贝塞尔	对话框效果图

从上图效果可以看出由贝塞尔函数画出来的曲线弧度是由控制点控制的，控制点的坐标值决定着该线段的弯曲大小和方向，比如上图代码控制点参数和结束点参数一致，它的效果是一条直线；当控制点 x 坐标大于结束点 x 坐标，它向上弯曲，反之向下，上图效果中控制点小于结束点 x 坐标，所以向下。

前文中说过它可以绘制比较复杂的图形，可以完成其它方法完成不了的绘制工作，比如对话框图形。上图中的对话框图形算是一个比较复杂的图形了，有圆角、不规则图形，但是它蕴含着一些规律，比如图中有四个圆角，两对圆角弧度一致。

接下来运用贝塞尔曲线函数来绘制上图中的对话框，代码如图所示：

<pre> ctx.beginPath(); ctx.moveTo(75, 25); ctx.quadraticCurveTo(25,25,25,62.5); ctx.quadraticCurveTo(25,25,25,62.5); ctx.quadraticCurveTo(25,100,50,100); ctx.quadraticCurveTo(50,120,30,125); ctx.quadraticCurveTo(60,120,65,100); ctx.quadraticCurveTo(125,100,125,62.5); ctx.quadraticCurveTo(125,25,75,25); ctx.stroke(); </pre>	

该效果和前文中提高的效果有略微的差别，想要和前文中的效果图一致，需要慢慢的调教每个贝塞尔函数的参数，还可以绘制更多类型的图形。

1.3.3 知识点 3 绘制简单动画

1、基本动画（Canvas 实现动画原理、动画帧、setTimeout()）

如何通过 Canvas 实现一个动画效果？通过前面单元可知实现一个动画效果的基本流程,但如果想通过学习原理实现一个动画该如何做呢？它是 setInterval() ,属于 HTML DOM Window 对象中的方法，具体用途如下：

setInterval 方法可以按照指定的周期（单位时间为毫秒）来调用函数，它会一直不停的调用直到程序满足一定条件结束或者被 clearInterval 方法结束。

参数	描述
code	必须，需要调用的函数
millisec	必须，执行周期，以毫秒为单位

返回值：一个 code 值，它可以用于取消 setInterval() 标记。

clearInterval 方法可以取消由 setInterval 设置的函数,必须为 clearInterval 传递一个取消目标的值，一般是 setInterval 方法返回的 code（ID）值。

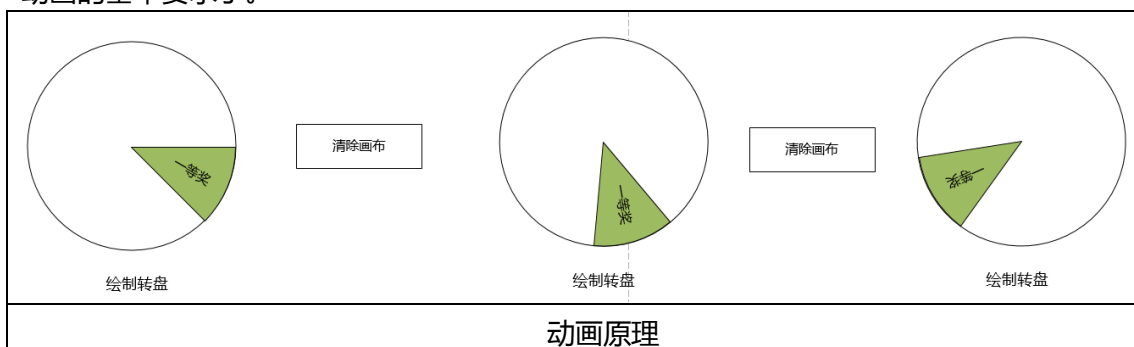
参数	描述
id	setInterval 返回的 code 值



以上 2 个方法又和动画有什么关系呢？它们既不是 Canvas 的方法,似乎和动画也无联系。首先了解关于动画的基础知识,什么是动画？顾名思义是会动的图像，在英文中有诸多描述，如 aimation、cartoon、aimation cartoon、cameracature。其中较正式的"Animation" 一词源自于拉丁文字根 anima，意为“灵魂” 动词 animate 是“赋予生命”的意思，引申为使某物活起来的意思。所以动画可以定义为使用绘画的手法，创造生命运动的艺术。

动画技术较为规范的定义是采用逐帧拍摄对象并连续播放而形成运动的影像技术。无论拍摄对象是什么，拍摄方式采用**逐帧方式**，观看时**连续播放形成了活动影像**，它就是动画。

逐帧方式和**连续播放**使它们被观看时形成一组连续帧组合，可以通过 JavaScript 去控制 Canvas 绘制每帧的操作，从而形成多帧序列，多个帧的播放可以通过 JavaScript 的变量记录，通过逻辑表达式去计算这些变量，使它们形成一个循环，为了使这个循环一直持续下去，借助 setInterval 方法，不断执行 Canvas 的绘制帧操作，这样便可以达到单一方向动画的基本要求了。

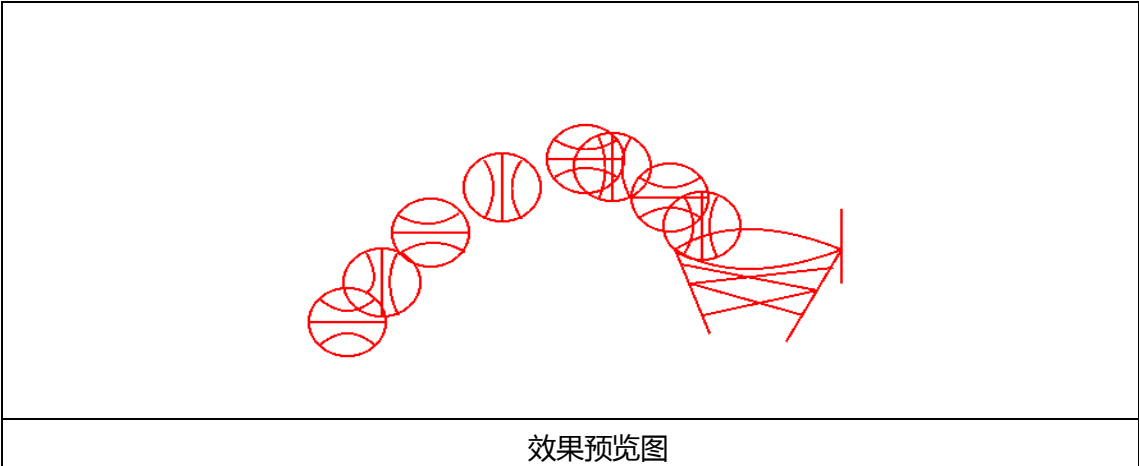


上图是 Canvas 绘制一帧的基本方法，通过绘制一帧->清除->绘制一帧->清除不断循

环，可以看到上图中的圆形角度处于变化中，这个变化由 JavaScript 中逻辑表达式控制，中间的过程系统方法控制，前文提到过它的方法执行周期是毫秒级的，人眼无法观察出来中间的变化，但是我们能看见圆的角度不断变化，形成一个动画效果。

1.4 任务实施

根据"任务陈诉"所要求，本单元将使用 JavaScript 的知识和前文知识点中介绍的 Canvas 操作方法在画布上绘制一组动画，为了弄懂动画原理，该动画效果每 500 毫秒执行一次，目的是放慢执行周期时间，理解每帧的是如何绘制的，它的实际效果如下图所示：



效果预览图

以上案例可以分为以下几个步骤分段完成：

- 1、通过 JavaScript 完成绘制每一帧的代码。
- 2、使用 setInterval 把第一步完成的步骤连接起来。
- 3、用开发者选项测试动画效果。

1.4.1 任务 1 绘制篮筐和篮球

通过前文的知识点介绍，已经知道如何创建和使用 Canvas，并需要注意的细节点，在空项目文件中创建一个 index.html 文件，因为效果不需要任何静态资源，所以项目中只存在一个 html 文件。

创建 Canvas 代码如下：

```
<canvas id="myCanvas" width="500" height="500" style="border:1px solid#d3d3d3;"></canvas>
```

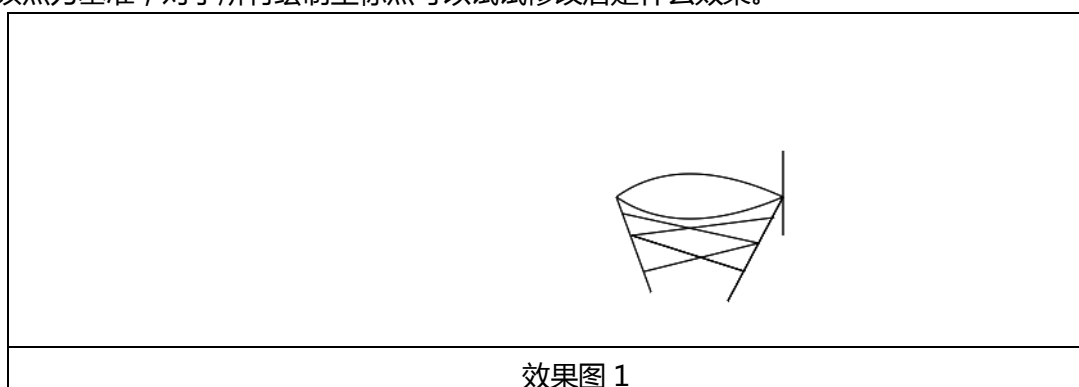
上图中为 Canvas 添加 border 属性，为测试 Canvas 是否创建成功，在前文的知识点中提高它的背景色为透明，无法看到效果，还需使用 JavaScript 代码测试，较为麻烦。

作为项目基础的底层创建后，便可以开始通过 JavaScript 编写逻辑调用 CanvasAPI 来绘制一个帧动画了，从右边的篮筐开始绘制吧，篮筐的图形来看是多个不规则的弧线和部分直线构成，运用前文知识点内容，可以使用贝塞尔曲线相关点解决该问题，不管是直线还是

曲线统统都可以使用贝塞尔完成。以下是绘制篮筐的动画帧代码：


<pre>//绘制篮筐 cxt.beginPath(); cxt.moveTo(306, 336); cxt.quadraticCurveTo(356, 300, 436, 336); cxt.stroke(); cxt.beginPath(); cxt.moveTo(306, 336); cxt.quadraticCurveTo(356, 370, 436, 336); cxt.stroke(); cxt.beginPath(); cxt.moveTo(306, 336); cxt.quadraticCurveTo(333, 410, 333, 410); cxt.stroke(); cxt.beginPath(); cxt.moveTo(436, 336); cxt.quadraticCurveTo(393, 417, 393, 417); cxt.stroke(); cxt.beginPath(); cxt.moveTo(436, 336); cxt.quadraticCurveTo(393, 417, 393, 417); cxt.stroke(); cxt.beginPath(); cxt.moveTo(311, 349); cxt.quadraticCurveTo(417, 372, 417, 372); cxt.stroke(); cxt.beginPath(); cxt.moveTo(318, 366); cxt.quadraticCurveTo(429, 352, 429, 352); cxt.stroke(); cxt.beginPath(); cxt.moveTo(328, 394); cxt.quadraticCurveTo(417, 372, 417, 372); cxt.stroke(); cxt.beginPath(); cxt.moveTo(406, 394); cxt.quadraticCurveTo(318, 366, 318, 366); cxt.stroke(); cxt.beginPath(); cxt.moveTo(406, 394); cxt.quadraticCurveTo(318, 366, 318, 366); cxt.stroke(); cxt.beginPath(); cxt.moveTo(436, 366); cxt.quadraticCurveTo(436, 300, 436, 300); cxt.stroke();</pre>
绘制篮筐

由于该效果是基于 Canvas 的整体宽高（500 * 500）来做，所以以上坐标点也都是根据该点为基准，对于所有绘制坐标点可以试试修改后是什么效果。



效果图 1

Canvas 中的篮筐效果，篮筐有了，接下来必须有一个篮球，提到篮球，可以想到它是一个圆形，通过圆形又可以联想到前文学到的知识点绘制一个圆，篮球表面有几条弯曲的弧线，又可以再次运用贝塞尔完成，以下是绘制篮球的代码：

<pre> cxt.beginPath(); cxt.arc(50, 400, 30, 0, Math.PI * 2, true); cxt.closePath(); cxt.stroke(); cxt.beginPath(); cxt.moveTo(20, 400); cxt.quadraticCurveTo(80, 400, 80, 400); cxt.stroke(); cxt.beginPath(); cxt.moveTo(29, 380); cxt.quadraticCurveTo(50, 400, 71, 380); cxt.stroke(); cxt.beginPath(); cxt.moveTo(29, 420); cxt.quadraticCurveTo(50, 400, 71, 420); cxt.stroke(); </pre>	
	
效果图 2	

有了篮球运动中 2 个基本的属性后，便可以让 Canvas 把篮球投入篮筐中，投进篮筐的球从起始点到终点是一个弧线，这又可以运用贝塞尔曲线完成运动路径了。

1.4.2 任务 2 实现连续动作

在图像渲染领域，存在“实时模式” (immediate mode)和“保留模式” (retained mode)的区别。HTML5 Canvas 属于实时模式。当图像发生改变时，Canvas 中所有的元素都需要重画。这种模式有一定的好处；比如使用全局属性可以让滤镜效果的实现变得非常容易。一旦掌握了实时模式的运用，在 Canvas 上绘图就是一件简单的事情。每当图像有任何更新的时候，直接重画整个 Canvas 即可。剩余的任务就是绘制篮球从起点到终点的过程了，为了使球看上去有更多的变化，绘制了多个不同角度的篮球，这部分代码比较庞大，为了不占用过多篇幅，以下贴出一小部分代码：

<pre> cxt.beginPath(); cxt.arc(77, 365, 30, 0, Math.PI * 2, true); cxt.closePath(); cxt.stroke(); cxt.beginPath(); cxt.moveTo(77, 335); cxt.quadraticCurveTo(77, 395, 77, 395); cxt.stroke(); cxt.beginPath(); cxt.moveTo(65, 339); cxt.quadraticCurveTo(77, 365, 64, 375); cxt.stroke(); cxt.beginPath(); cxt.moveTo(89, 339); cxt.quadraticCurveTo(77, 365, 90, 394); cxt.stroke(); </pre>
--

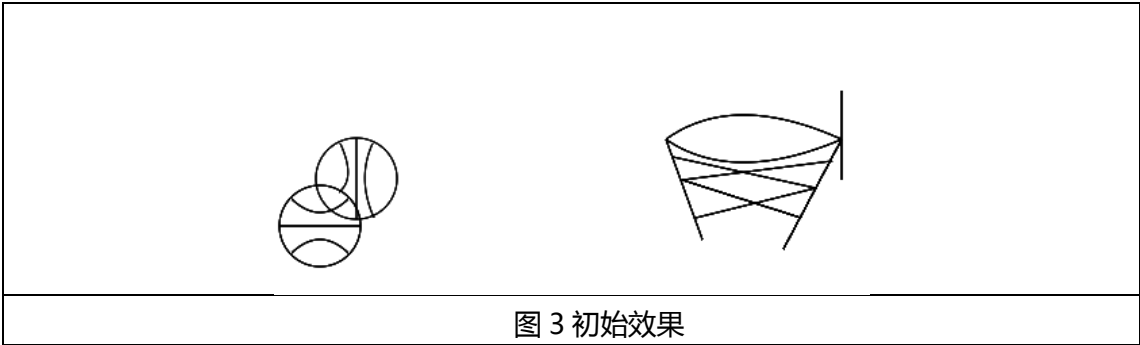


图 3 初始效果

通过 JavaScript 完成了动画中每一帧的操作过程，现在运行后可以看到以下效果：

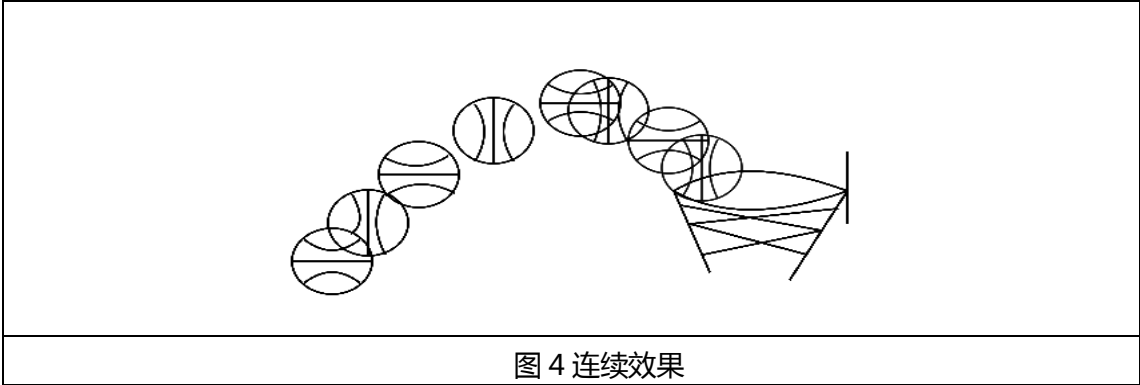


图 4 连续效果

刷新浏览器立刻就可以看到了，但这个并不是一个动画，它只是一个由 Canvas 绘制的图形，怎么样才可以让它动起来呢？

前文中总共通过 JavaScript 绘制了 10 个动画帧的操作，如果一个一个顺序执行就接近成为一个连贯的动画效果。运用 setInterval 方法（周期性执行某个函数），并把这 10 个动画帧代码拆分为 10 个步骤，使用逻辑表达式去辨别它们，代码如下：

```
function basketball(step){  
    if(step > 1){  
        //步骤一  
    }  
    If(step > 2){  
        //步骤二  
    }  
    If(step > 3){  
        //步骤三  
    }  
    ... 省略  
}
```

以上方法通过外部传入的 step 参数来辨别当前函数执行到哪个步骤，并且 step 参数是递增的，直到不满足判断为止。

把前文写的 10 个绘制代码分别放进以上代码中，在创建一个函数用于传入的一个不断递增的 step 参数，代码如下：

```
function draw() {  
    var step = 0;  
    basketball (i);  
    if (step < 11) step ++;  
}
```

draw 函数本身不具备执行多次 basketball()的能力，除非用户调用多次，但是调用此处用户无法控制，可以借助 setInterval 方法，周期时间内执行函数，由于被执行函数内有逻辑判断，所以不必担心它会造成一些逻辑上的错误。

```
setInterval(draw, 500);
```

最终它会一步一步执行绘制操作，当 step 大于预定值，结束流程。该效果只有 10 帧，看上去不是非常流畅。可以缩短方法执行周期使它流畅的执行，但因为它只有 10 帧看到的效果就是一闪而过。

1.5 单元小结

HTML5 的 canvas 元素使用 JavaScript 在网页（画布）上绘制各种图形，包括路径、矩形、圆形、字符以及添加图像等操作。画布是定义的一个矩形区域，读者可以在其中控制其每一像素。Canvas 在 HTML5 有着重要意义，可以说是 HTML5 强大功能的体现。它改写了网页上不能直接交互绘图的历史。

到目前为止，我们系统地学习了 Canvas 画布相关的操作，包括 Canvas 基本用法和样式设计，绘制图形，调用、渲染、裁剪和缩放图像；以及学习了多种图片显示效果，图片的变形、组合方法和基本动画制作方法。本章的重点仍然是基本的静态画布制作。但一旦读者懂得灵活应用 JavaScript 结合目前的 canvas 知识，读者就可以制作出漂亮的动画或游戏。

1.6 单元练习

【练习目的】

1. 能用 Canvas 绘制图形；
2. 用 Canvas 完成各种图像处理；
3. 尝试自己完成制作一个场景动画。

【练习内容】

生成运动中的正弦波曲线

当我们将一个石子儿投入平静的湖面时，会激起一圈一圈的波纹。波纹中有波峰和波谷，如果我们将某个截面上的波纹放大看，会发现其形状与图 9-50 所示的形状非常相似。其实

这是一种正弦波是，在 JavaScript 中可以用函数 `sin()` 来模拟。本题的要求就是制作一个自左向右行进的正弦波，截图下所示。

