HTML5 Canvas 教程

原作者：Eric Rowell

原文地址：http://www.html5canvastutorials.com/

翻译：ysm @ http://iysm.net

译者注：本文是来自于网络上的一个教程，英文原版在 http://www.html5canvastutorials.com/。原文比较简单明了，适合初学入门，跟着上面的代码很快就能学会 HTML5 的 Canvas 基本的绘图功能。本文就是根据此文翻译而来，只不过原文中有些地方个人感觉介绍得过于简单，因此结合其他相关材料，根据自己的理解，对某些章节的内容略作修改。如有什么不够准确的地方，请不吝赐教。谢谢。

Part 1: HTML5 Canvas 教程基础篇

欢迎来到基础篇部分。在此，我们将主要看一下 HTML5 Canvas 的基本绘图功能，包括画直线、画路径、图形绘制、渐变、 模式、图像和文本。

学前准备：

在开始基础教程之前，您首先需要准备一个不太旧的 web 浏览器，比如 Google Chrome ， Firefox， Safari， Opera， 或者 IE9 这些都可以。然后您需要对 Javascript 有一定的熟悉，并且还得有一个文本编辑器，比如 notepad。

1.1 HTML5 Canvas

1.1.1 HTML5 Canvas 的元素

在 HTML5 页面里， canvas 就是像 ， <a>， 或 <table>之类的一种标签，所不同的是， canvas 需要用 Javascript 来渲染。要使用 canvas，我们就需要在 HTML5 文件的适当位置添加 canvas 标签，然后创建一个 Javascript 初始化函数，使这个函数在页面加载的时候就执行，同时在

函数里用调用 HTML5 Canvas API 在 canvas 上画图就可以了。

比如我们像下面这样添加一个 id 为 myCanvas 的 canvas 标签：

```
<body>
  <canvas id="myCanvas"></canvas>
</body>
```

然后添加初始化 Javascript 函数：

```
<!DOCTYPE HTML>
<html>
  <head>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

    // 在此添加绘图代码
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

HTML5 Canvas 有关元素的说明

上述代码将在本文后续中作为 HTML5 Canvas 程序的基本模板。就像其他 HTML 标签一样，我们也可以用 canvas 标签的 height 和 width 属性为 canvas 指定其在页面上高度和宽度。在初始化 Javascript 函数中，我们可以用 canvas 标签的 id 获得 canvas 的 DOM 对象，并用 getContext() 方法获得这个 canvas 的 “2d”上下文对象，其后的绘图操作都将以此上下文对象为基础。

1.2 直线

1.2.1 画直线

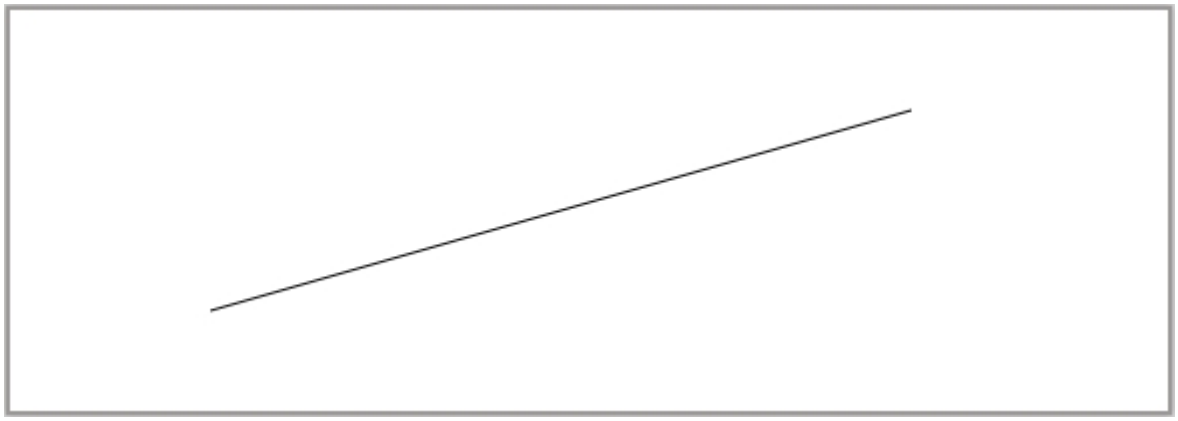
画直线的功能可以用 beginPath()，moveTo()，lineTo() 和 stroke() 几个方法的组合来

实现。

如：

```
<script>
  context.beginPath();
  context.moveTo(x,y);
  context.lineTo(x,y);
  context.stroke();
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(100, 150);
        context.lineTo(450, 50);
        context.stroke();
```

```
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

关于画直线的有关说明

方法 `beginPath()` 定义了一个新的路径绘制动作的开始。

方法 `moveTo()` 为指定点创建了一个新的子路径，这个点就变成了新的上下文点。我们可以把 `moveTo()` 方法看成用来定位我们的绘图鼠标用的。

方法 `lineTo()` 以上下文点为起点，到方法参数中指定的点之间画一条直线。

方法 `stroke()` 为所画的线赋予颜色，并使其可见。如果没有特别的指定颜色的话，则默认使用黑色画直线。

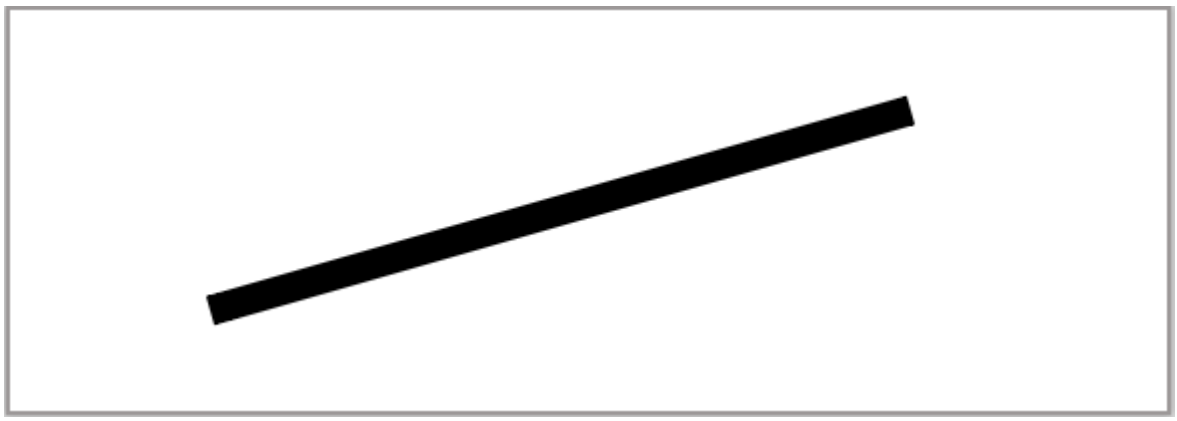
1.2.2 直线的宽度

直线的宽度用 `lineWidth` 属性设定。

如：

```
<script>
  context.lineWidth = 5;
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(100, 150);
        context.lineTo(450, 50);
        context.lineWidth = 15;
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

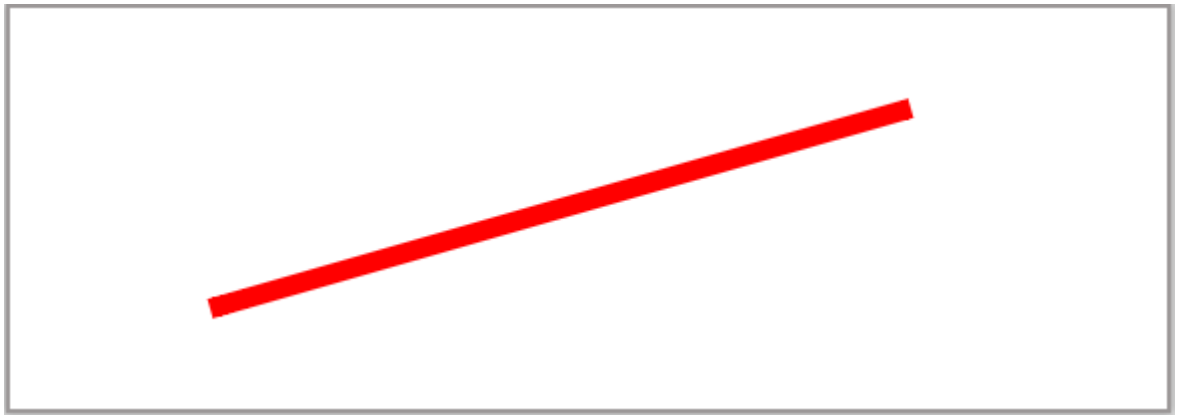
1.2.3 直线颜色

直线的颜色用 `strokeStyle` 属性设定。

如：

```
<script>
  context.strokeStyle = 'blue';
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(100, 150);
        context.lineTo(450, 50);
        context.lineWidth = 10;

        // 设置线的颜色
        context.strokeStyle = "#ff0000";
        context.stroke();
      };
```

```
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.2.4 直线端点样式

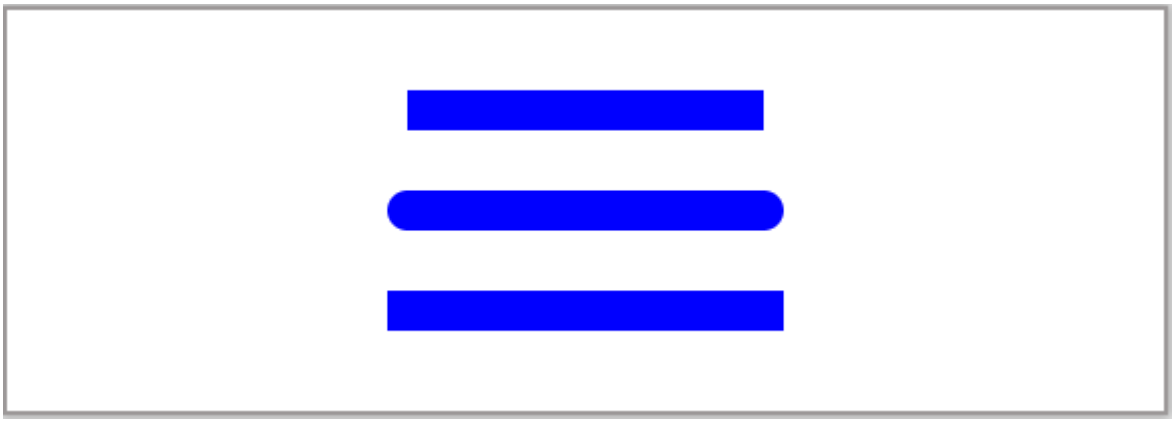
HTML5 canvas 支持 3 种直线的端点样式，包括： butt， round， 和 square。 设定端点样式是用 lineCap 属性设定。缺省情况下，将使用 butt 样式。

如：

```
<script>
  context.lineCap = 'round';
</script>
```

注意：在使用 round 或 square 样式的时候，直线的长度会增加，具体增加的长度等于线的宽度。比如，一条长 200px，宽 10px 的直线，如果使用 round 或 square 样式，由于每个端点增加了 5px，则直线的实际长度会达到 210px。

效果图，分别为 Butt， Round 和 Square



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
```

```
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 最上面的线是butt样式
        context.beginPath();
        context.moveTo(200, canvas.height / 2 - 50);
        context.lineTo(canvas.width - 200, canvas.height / 2 - 50);
        context.lineWidth = 20;
        context.strokeStyle = "#0000ff";
        context.lineCap = "butt";
        context.stroke();

        // 中间的线是round样式
        context.beginPath();
        context.moveTo(200, canvas.height / 2);
        context.lineTo(canvas.width - 200, canvas.height / 2);
        context.lineWidth = 20;
        context.strokeStyle = "#0000ff";
        context.lineCap = "round";
        context.stroke();

        // 最下面的是square样式
        context.beginPath();
        context.moveTo(200, canvas.height / 2 + 50);
        context.lineTo(canvas.width - 200, canvas.height / 2 + 50);
        context.lineWidth = 20;
        context.strokeStyle = "#0000ff";
        context.lineCap = "square";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.3 弧线

1.3.1 画弧线

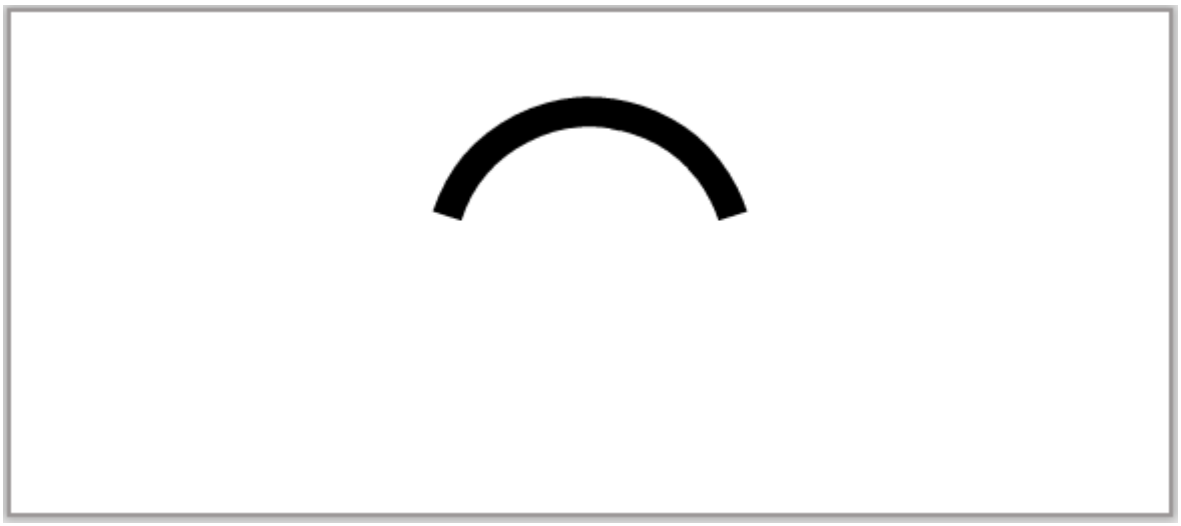
画弧线的方法是 `arc()` 。每条弧线都需要由中心点、半径、起始角度（弧度`n*Math.PI`）、结束角度（弧度`m*Math.PI`）和绘图方向（顺时针 `false` 还是逆时针 `true` ）这几个参数来确定。

如：

```
<script>
  context.arc(x, y, radius, startAngle, endAngle, antiClockwise);
</script>
```

另外，我们还可以用`arcTo()`方法来画弧线，用来在路径中绘制圆角，具体内容详见`1.4.3`。

效果图

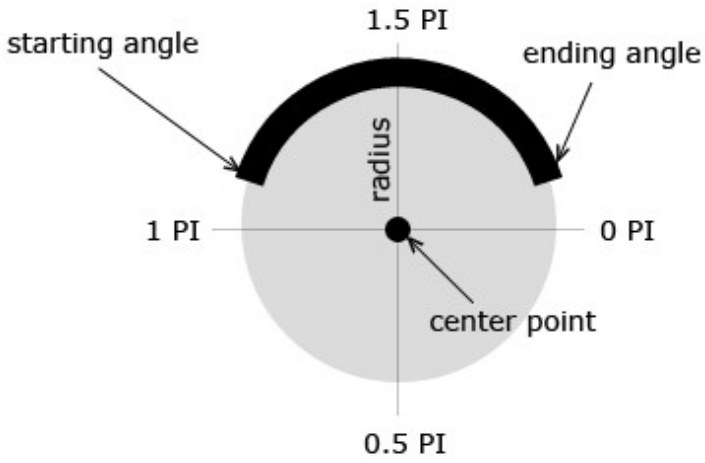


代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
```

```
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = canvas.width / 2;
        var y = canvas.height / 2;
        var radius = 75;
        var startAngle = 1.1 * Math.PI;
        var endAngle = 1.9 * Math.PI;
        var counterClockwise = false;

        context.beginPath();
        context.arc(x, y, radius, startAngle, endAngle,
counterClockwise);
        context.lineWidth = 15;
        // 设置线的颜色
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="250"></canvas>
  </body>
</html>
```

弧线的有关解释



由上图可见，一条弧线无非就是一个以中心点为圆心，指定半径的圆的一部分。这个圆的一部分就是圆周上，从 `startAngle` 到 `endAngle` 的部分，其中 `startAngle` 和 `endAngle` 都用弧度来表示。而最后一个方向参数则是指的从 `startAngle` 到 `endAngle` 两点之间画线的方向是否为逆时针，默认情况下这个参数是 `false`，因此也就是顺时针画线。

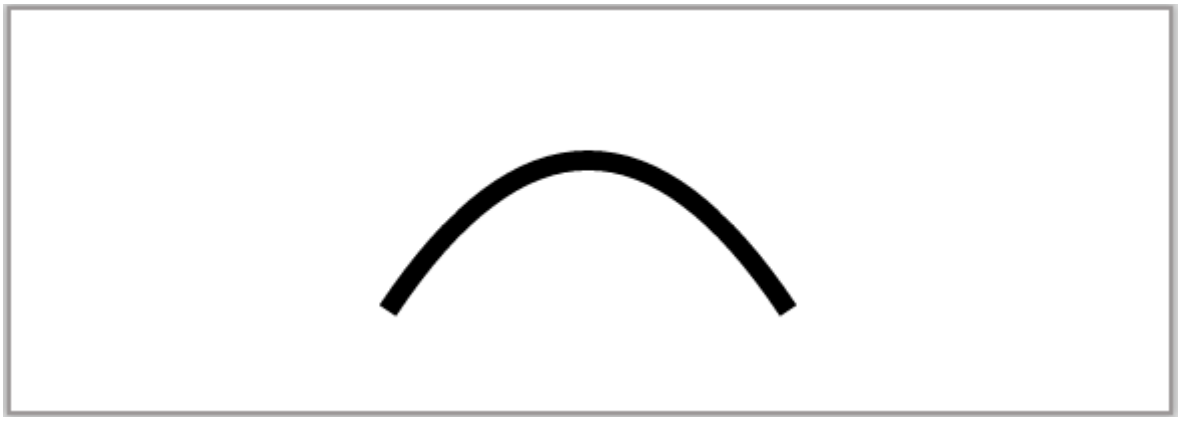
1.3.2 二次曲线

二次曲线使用 `quadraticCurveTo()` 方法来绘制。每条二次曲线要由上下文点、一个控制点和一

个终止点来定义。

```
<script>
  context.quadraticCurveTo(controlX, controlY, endX, endY);
</script>
```

效果图

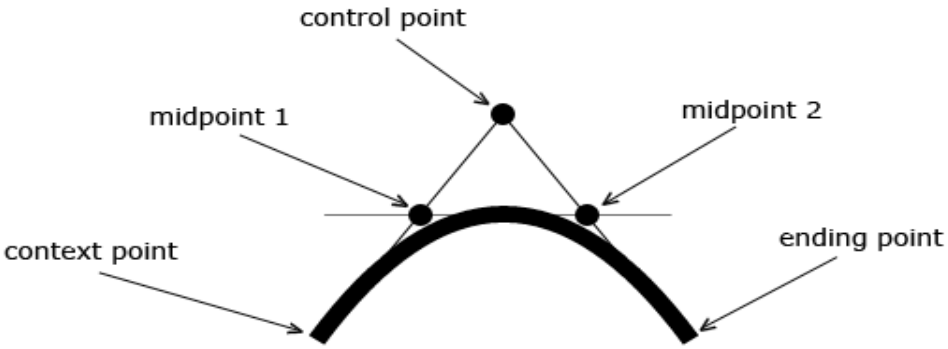


代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(188, 150);
        context.quadraticCurveTo(288, 0, 388, 150);
        context.lineWidth = 10;
        // line color
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
```

```
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

关于二次曲线的说明



二次曲线就是如上图所示的样式，控制点通过分别与上下文点和终止点之间的虚拟切线控制曲线的形状。其中上下文点由 `moveTo()` 方法在调用 `quadraticCurveTo()` 方法前指定。控制点离上下文点和终止点越远，曲线就越尖锐，反之，曲线则越平缓。

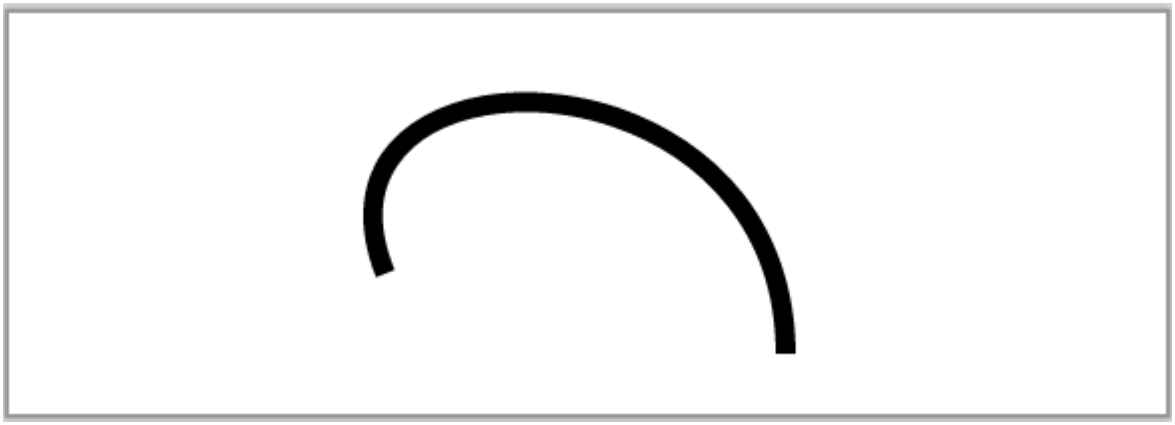
1.3.3 贝塞尔曲线

绘制贝塞尔曲线使用方法 `bezierCurveTo()` 。每条贝塞尔曲线需要由上下文点、两个控制点和一个终止点来确定。由于贝塞尔曲线有两个控制点，因此贝塞尔曲线可以比二次曲线更加的复杂多变。

如：

```
<script>
  context.bezierCurveTo(controlX1, controlY1, controlX2, controlY2, endX,
endY);
</script>
```

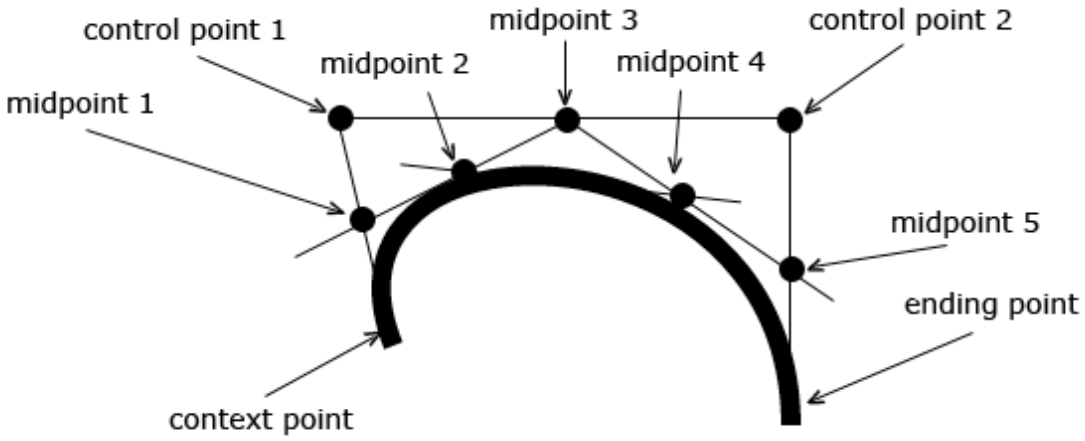
效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(188, 130);
        context.bezierCurveTo(140, 10, 388, 10, 388, 170);
        context.lineWidth = 10;
        // 设置线的颜色
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

关于贝塞尔曲线的说明



由上图可见，贝塞尔曲线由当前上下文点、两个控制点和终止点确定。贝塞尔曲线的第一部分是由上下文点和第一个控制点确定的虚拟线的切线，第二部分则是由第二个控制点和终止点确定的虚拟线的切线。

1.4 路径

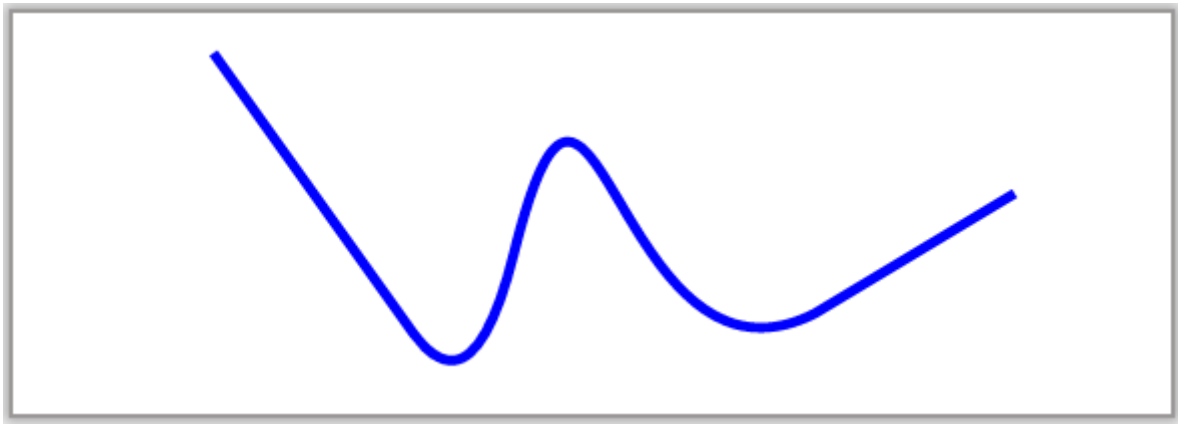
1.4.1 画路径

路径是由多条子路径连接构成的。每条子路径的终止点就将作为新的上下文点。我们可以使用 `lineTo()`，`arcTo()`，`quadraticCurveTo()` 和 `bezierCurveTo()` 创建新的子路径。每次要开始画一条路径的时候就要使用 `beginPath()` 方法。

如：

```
<script>
  context.beginPath();
</script>
```

效果图

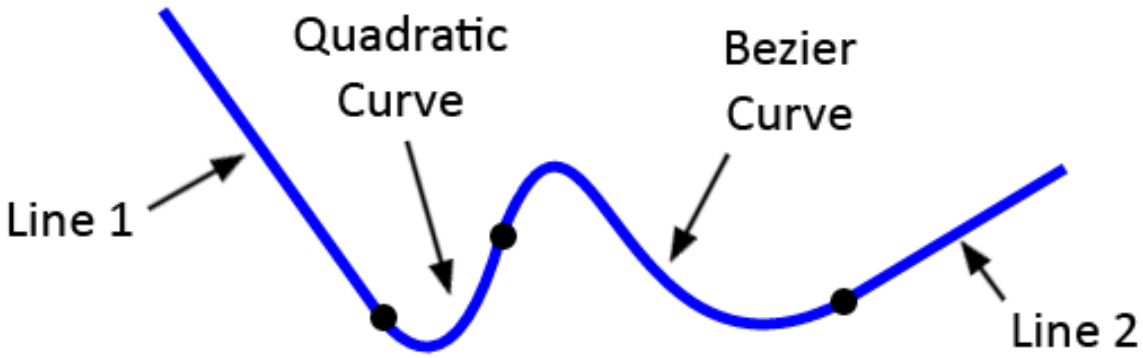


代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(100, 20);

        // 第一条直线
        context.lineTo(200, 160);

        // 二次曲线
        context.quadraticCurveTo(230, 200, 250, 120);

        // 贝塞尔曲线
        context.bezierCurveTo(290, -40, 300, 200, 400, 150);

        // 第二条直线
        context.lineTo(500, 90);

        context.lineWidth = 5;
        context.strokeStyle = "blue";
        context.stroke();
      };

    </script>
  </head>
```

```
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

关于路径的说明



1.4.2 线条的连接样式

HTML5 canvas 支持 3 种线条的连接样式，包括： miter， round， 和 bevel。 设定连接样式是用 lineJoin 属性设定。缺省情况下，将使用 miter 样式。

如：

```
<script>
  context.lineJoin = 'round';
</script>
```

注意：如果线条比较细，而且之间的连接并不形成很尖锐的角度，那么不同的连接样式之间可能会比较难以区分。

效果图，分别为 Miter， Round 和 Bevel

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 设置线的宽度
        context.lineWidth = 25;

        // 最左边的是 miter 样式
        context.beginPath();
        context.moveTo(99, 150);
        context.lineTo(149, 50);
        context.lineTo(199, 150);
        context.lineJoin = "miter";
        context.stroke();

        // 中间的是 round 样式
        context.beginPath();
        context.moveTo(239, 150);
        context.lineTo(289, 50);
        context.lineTo(339, 150);
        context.lineJoin = "round";
        context.stroke();
        // 最右边的是 bevel 样式
        context.beginPath();
        context.moveTo(379, 150);
        context.lineTo(429, 50);
```

```
        context.lineTo(479, 150);
        context.lineJoin = "bevel";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

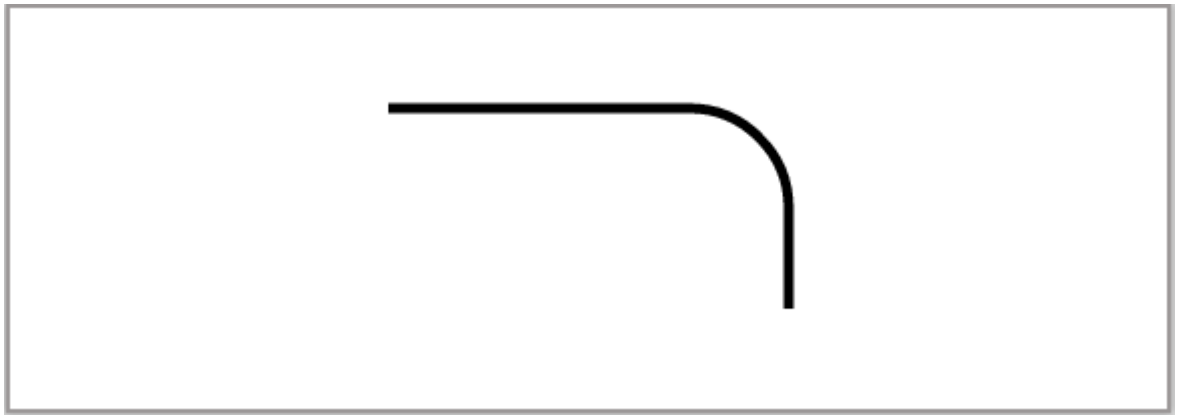
1.4.3 圆角

画圆角使用方法 `arcTo()` 。此方法需要一个控制点、一个终止点和半径作为必要的参数。

如：

```
<script>
  context.arcTo(controlX,controlY,endX,endY,radius);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
```

```
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 200;
        var rectHeight = 100;
        var rectX = 189;
        var rectY = 50;
        var cornerRadius = 50;

        context.beginPath();
        context.moveTo(rectX, rectY);
        context.lineTo(rectX + rectWidth - cornerRadius, rectY);
        context.arcTo(rectX + rectWidth, rectY, rectX + rectWidth, rectY
+ cornerRadius, cornerRadius);
        context.lineTo(rectX + rectWidth, rectY + rectHeight);
        context.lineWidth = 5;
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.5 图形

1.5.1 画用户自定义图形

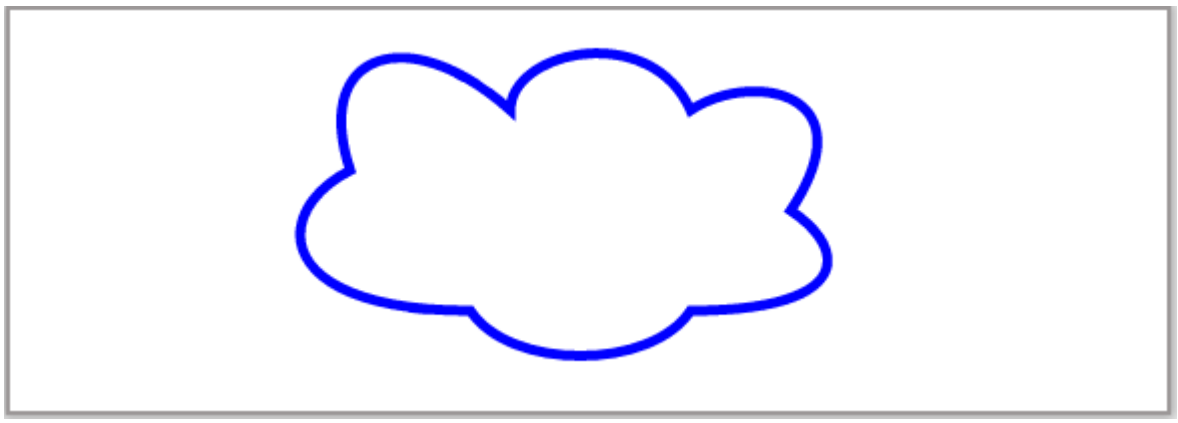
要画一个用户自定义的图形，需要创建一个路径，然后用 `closePath()` 方法关闭此路径即可。

我们可以使用 `lineTo()`，`arcTo()`，`quadraticCurveTo()` 或 `bezierCurveTo()` 来创建每个子路径，然后用这些子路径组成想要的图形。

如：

```
<script>
  context.closePath();
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 开始用户图形的绘制
        context.beginPath();
        context.moveTo(170, 80);
        context.bezierCurveTo(130, 100, 130, 150, 230, 150);
        context.bezierCurveTo(250, 180, 320, 180, 340, 150);
        context.bezierCurveTo(420, 150, 420, 120, 390, 100);
        context.bezierCurveTo(430, 40, 370, 30, 340, 50);
        context.bezierCurveTo(320, 5, 250, 20, 250, 50);
        context.bezierCurveTo(200, 5, 150, 20, 170, 80);

        // 完成用户图形的绘制
        context.closePath();
        context.lineWidth = 5;
        context.strokeStyle = "blue";
        context.stroke();
      };

    </script>
```

```
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.5.2 图形的颜色填充

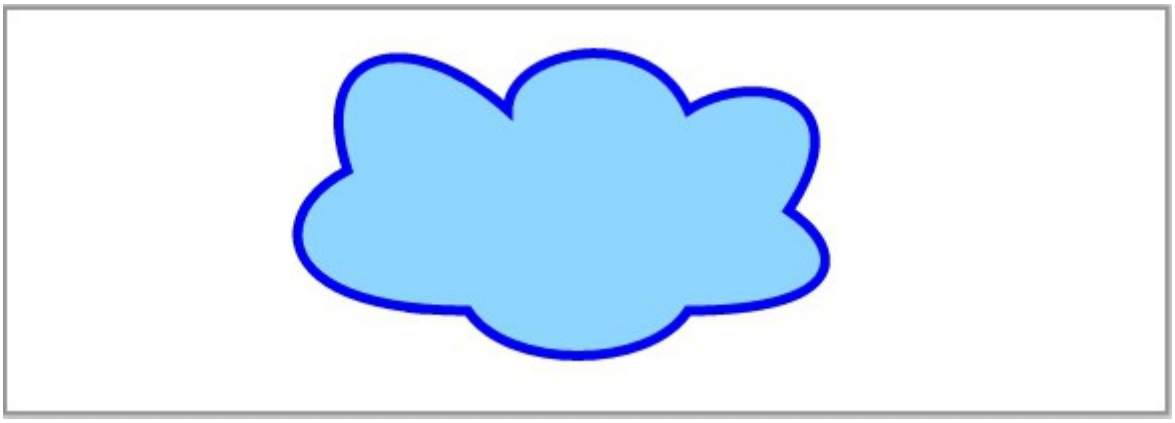
要填充图形，需要用 `fillStyle` 属性设置填充图形用的颜色，然后使用 `fill()` 方法完成对图形的填充。默认情况下， `fillStyle` 的颜色是黑色。

如：

```
<script>
  context.fillStyle = "blue";
  context.fill();
</script>
```

注意： `fill` 方法要在 `strok` 方法之前执行，否则 `fill` 会覆盖掉 `stroke` 的一半区域。

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
```

```
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 开始用户图形的绘制
        context.beginPath();
        context.moveTo(170, 80);
        context.bezierCurveTo(130, 100, 130, 150, 230, 150);
        context.bezierCurveTo(250, 180, 320, 180, 340, 150);
        context.bezierCurveTo(420, 150, 420, 120, 390, 100);
        context.bezierCurveTo(430, 40, 370, 30, 340, 50);
        context.bezierCurveTo(320, 5, 250, 20, 250, 50);
        context.bezierCurveTo(200, 5, 150, 20, 170, 80);

        // 完成用户图形的绘制
        context.closePath();
        context.lineWidth = 5;
        context.fillStyle = "#8ED6FF";
        context.fill();
        context.strokeStyle = "blue";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

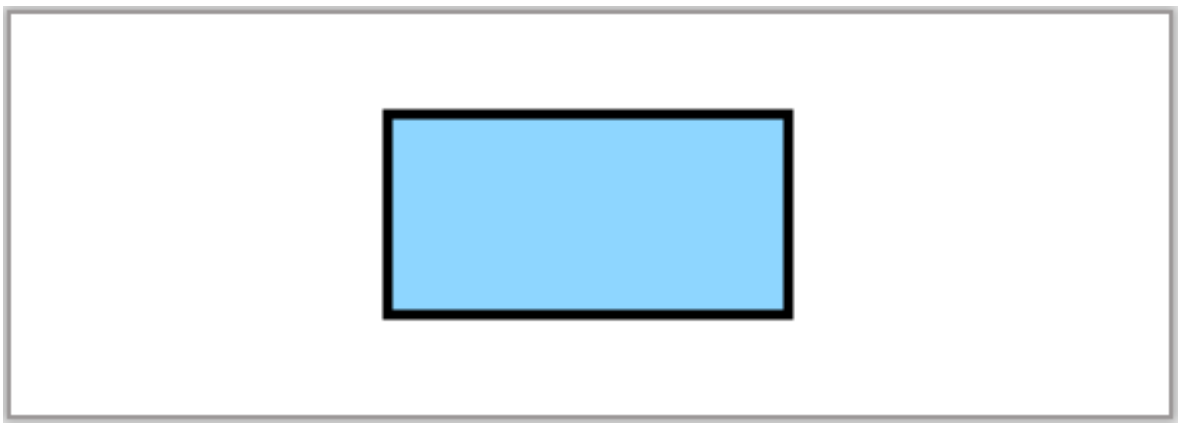
1.5.3 矩形

绘制矩形使用 `rect()` 方法。每个矩形需要由左上角坐标 `(x, y)` 和矩形的宽与高 `(width, height)`来确定。

如：

```
<script>
  context.rect(x, y, width, height);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      #myCanvas {
        border: 1px solid #9C9898;
      }
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById('myCanvas');
        var context = canvas.getContext('2d');

        context.beginPath();
        context.rect(188, 50, 200, 100);
        context.fillStyle = '#8ED6FF';
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = 'black';
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

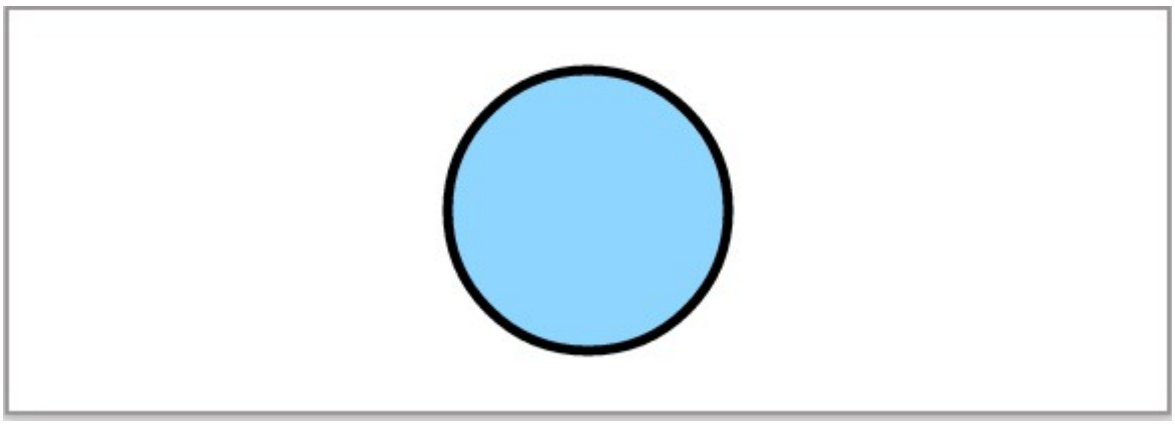
1.5.4 圆

画圆只需要在调用 `arc()` 方法时，将起始角度设为 `0`，终止角度设为 `2 * PI` 即可。

如：

```
<script>
  context.arc(x, y, radius, 0 , 2 * Math.PI, false);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var centerX = canvas.width / 2;
        var centerY = canvas.height / 2;
        var radius = 70;

        context.beginPath();
```

```
        context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
        context.fillStyle = "#8ED6FF";
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

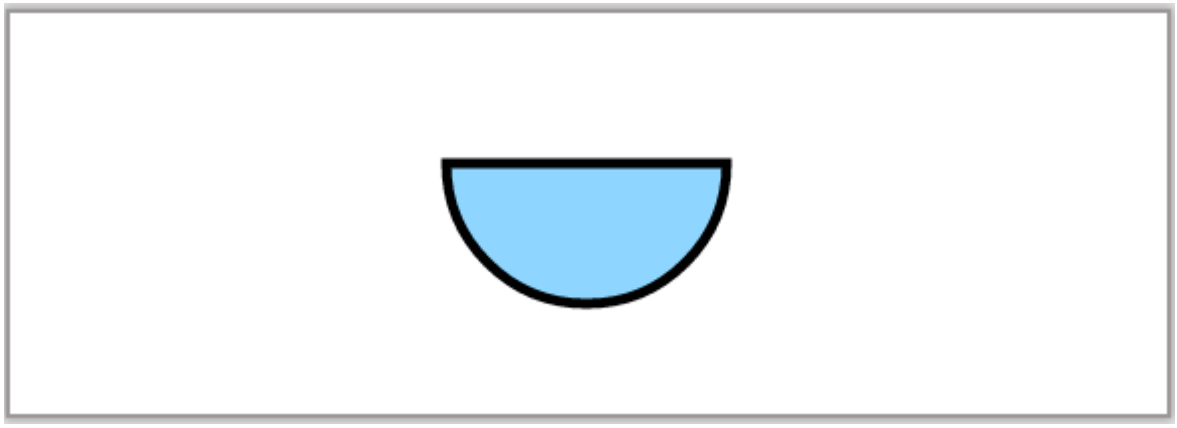
1.5.5 半圆

画半圆也是用 `arc()` 方法，只需要将终止角度设为起始角度 `+ PI` 即可。

如：

```
<script>
  context.arc(x, y, radius, startAngle, startAngle + Math.PI,
antiClockwise);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
```

```
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.arc(288, 75, 70, 0, Math.PI, false);
        context.closePath();
        context.lineWidth = 5;
        context.fillStyle = "#8ED6FF";
        context.fill();
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.6 填充类型

1.6.1 线性渐变

要使用线性渐变效果填充图形，首先要使用 `createLinearGradient()` 方法从上下文对象中创建线性渐变对象。 `createLinearGradient()` 方法的四个参数确定一条虚拟线段，渐变就沿着此条线段的方向。

然后用 `addColorStop` 方法为线性渐变对象设置渐变线上的关键点的颜色， `offset` 表示关键点是在渐变虚拟线段的什么位置， `offset` 的取值范围是 `0` 到 `1`，其中 `0` 表示是起始点，`1` 表示是终止点，`0` 到 `1` 之间的值表示是此虚拟线段中间的某一位置。

再将此线性渐变对象作为填充类型赋值给上下文对象的 `fillStyle` 属性。`canvas` 将根据渐变虚拟线段上关键点的颜色变化填充图形。

如：

```
<script>
  var grd = context.createLinearGradient(startX, startY, endX, endY);
  grd.addColorStop(offset, color);
  ......
  context.fillStyle = grd;
  context.fill();
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(170, 80);
        context.bezierCurveTo(130, 100, 130, 150, 230, 150);
        context.bezierCurveTo(250, 180, 320, 180, 340, 150);
        context.bezierCurveTo(420, 150, 420, 120, 390, 100);
        context.bezierCurveTo(430, 40, 370, 30, 340, 50);
```

```
        context.bezierCurveTo(320, 5, 250, 20, 250, 50);
        context.bezierCurveTo(200, 5, 150, 20, 170, 80);
        context.closePath();

        // 创建线性渐变对象
        var grd = context.createLinearGradient(230, 0, 370, 200);
        // 添加渐变的起点处颜色
        grd.addColorStop(0, "#8ED6FF");
        // 添加渐变终点处的颜色
        grd.addColorStop(1, "#004CB3");
        context.fillStyle = grd;
        context.fill();

        // 画边缘线
        context.lineWidth = 5;
        context.strokeStyle = "blue";
        context.stroke();
      };
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

关于线性渐变的说明

线性渐变的方向是从 `createLinearGradient()` 方法的 `(startX, startY)` 点到 `(endX, endY)` 点的一条虚拟线段。我在这里用了两个 `color stop` ，渐变的起始点是淡蓝色，渐变的终止点是深蓝色。`Color stop` 是指的在虚拟线段上的位置点，范围是 `0` 到 `1`，其中 `0` 表示是起始点，`1` 表示是终止点。

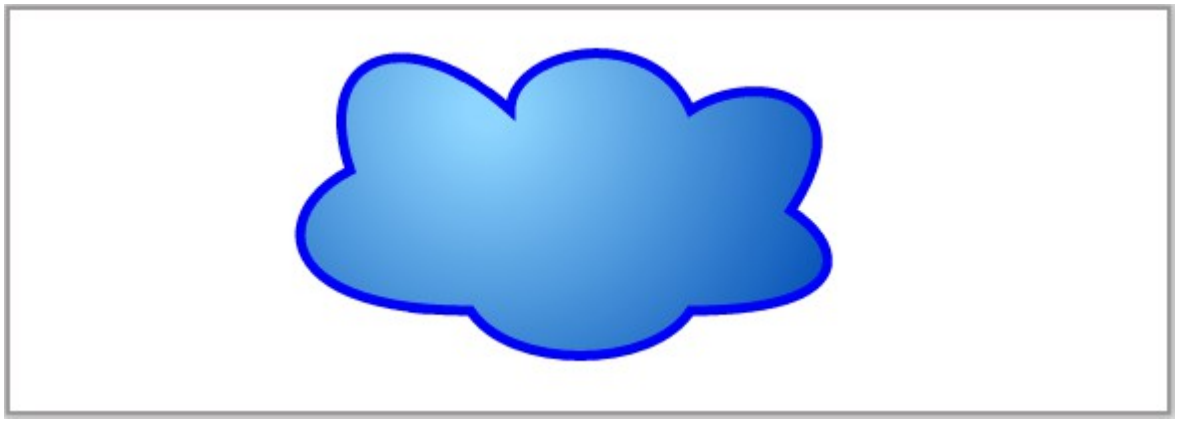
1.6.2 径向渐变

径向渐变与线性渐变类似，只不过渐变方向不是线段，而是两个圆之间。使用 `createRadialGradient()` 方法创建渐变对象，参数是渐变起始圆和终止圆。

如：

```
<script>
  var grd=context.createRadialGradient(startX, startY, startRadius, endX, endY, endRadius);
  grd.addColorStop(offset, color);
  ......
  context.fillStyle = grd;
  context.fill();
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
        context.moveTo(170, 80);
        context.bezierCurveTo(130, 100, 130, 150, 230, 150);
        context.bezierCurveTo(250, 180, 320, 180, 340, 150);
        context.bezierCurveTo(420, 150, 420, 120, 390, 100);
        context.bezierCurveTo(430, 40, 370, 30, 340, 50);
        context.bezierCurveTo(320, 5, 250, 20, 250, 50);
        context.bezierCurveTo(200, 5, 150, 20, 170, 80);
        context.closePath();

        // 创建径向渐变对象
        var grd = context.createRadialGradient(238, 50, 10, 238, 50,
200);
        // 设置渐变起始圆处的颜色
        grd.addColorStop(0, "#8ED6FF");
        // 设置渐变终止圆处的颜色
        grd.addColorStop(1, "#004CB3");
        context.fillStyle = grd;
```

```
        context.fill();

        // 画边缘线
        context.lineWidth = 5;
        context.strokeStyle = "blue";
        context.stroke();
      };
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.6.3 图案填充

要实现图案填充功能，首先要用上下文对象的 `createPattern()` 方法创建一个图案填充对象，将这个对象赋值给上下文对象的 `fillStyle` 属性，然后使用 `fill()` 方法填充图形。

其中 `createPattern()` 方法需要两个参数，第一个参数是一个图像对象，第二个参数是重复模式，可选的模式是 `repeat`， `repeat-x`， `repeat-y`， 以及 `no-repeat`，默认情况是 `repeat` 。

如：

```
<script>
  var pattern = context.createPattern(imageObj, repeatOption);
  context.fillStyle = pattern;
  context.fill();
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        var imageObj = new Image();
        imageObj.onload = function() {
          var pattern = context.createPattern(imageObj, "repeat");

          context.rect(10, 10, canvas.width - 20, canvas.height - 20);
          context.fillStyle = pattern;
          context.fill();
        };
        imageObj.src =
"http://www.html5canvastutorials.com/demos/assets/wood-pattern.png";
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.7 图像

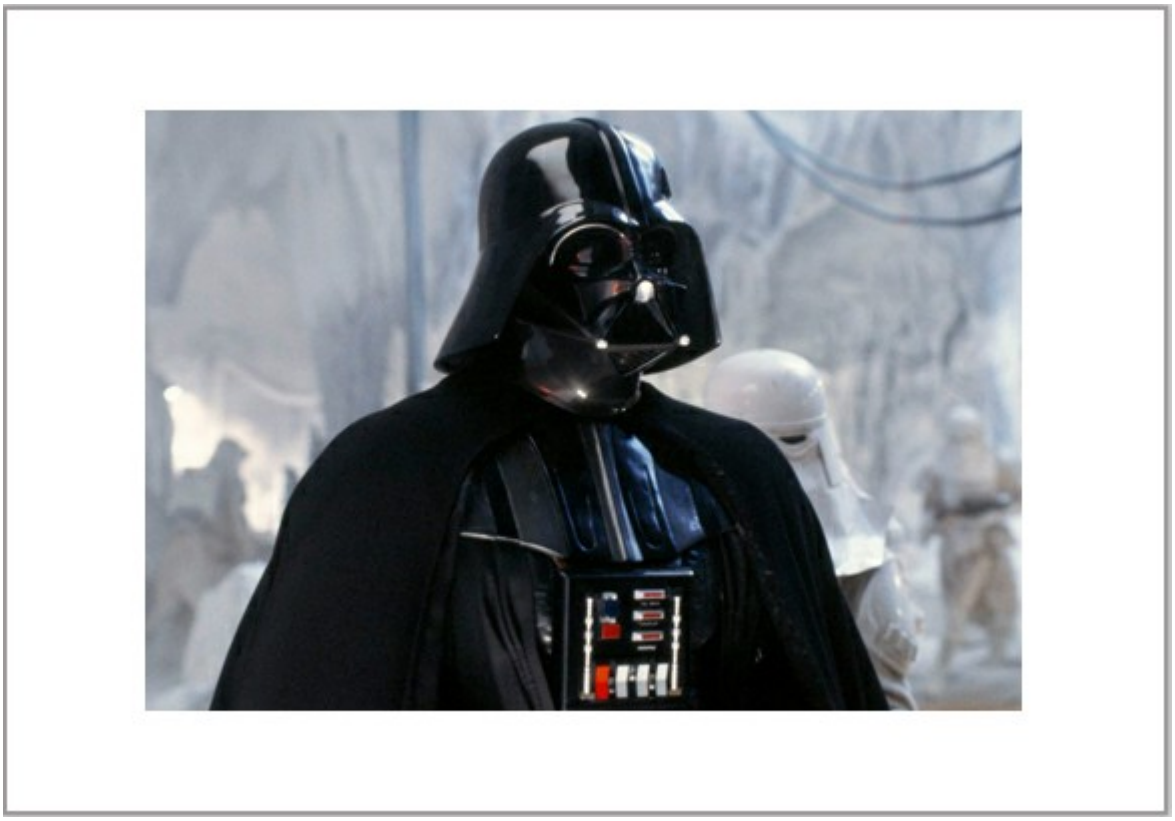
1.7.1 绘制图像

绘制图像需要使用 `drawImage()` 方法。此方法需要一个图像对象和一个起始点坐标作为参数，其中起始点坐标是相对于 `canvas` 的左上角的位置。

如：

```
<script>
  context.drawImage(imageObj, x, y);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var imageObj = new Image();

        imageObj.onload = function() {
          context.drawImage(imageObj, 69, 50);
        };
        imageObj.src =
"http://www.html5canvastutorials.com/demos/assets/darth-vader.jpg";
```

```
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="400"></canvas>
  </body>
</html>
```

关于图像的说明

由于 drawImage() 方法需要一个图像对象作为参数，所以我们需要在实际调用 drawImage() 之前就加载图像。这一要求可以通过图像对象的 onload 事件来实现。不过要注意的是， onload 应该在用图像对象的 src 属性指定图像来源之前就进行赋值。

1.7.2 图像尺寸

方法 drawImage() 还可以接受 destWidth 和 destHeight 两个参数用来以任意指定的尺寸显示图像。

如：

```
<script>
  context.drawImage(imageObj, x, y, width, height);
</script>
```

效果图

代码

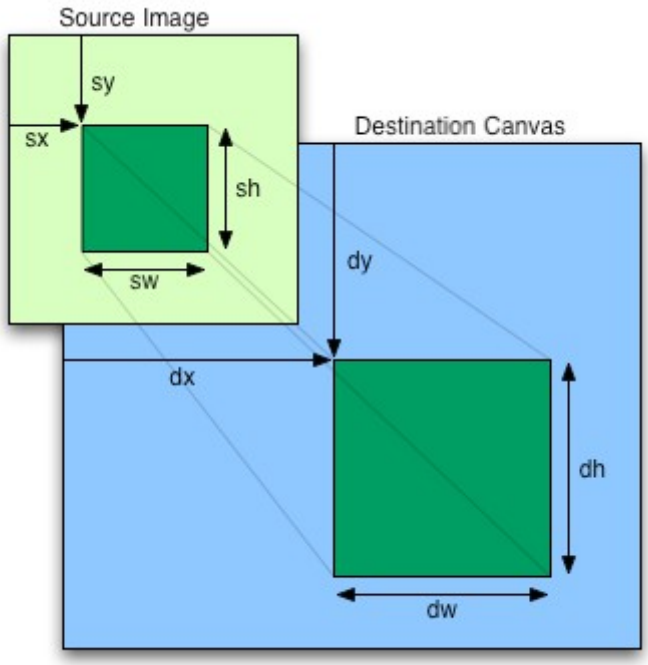
```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = 188;
        var y = 130;
        var width = 200;
        var height = 137;
        var imageObj = new Image();

        imageObj.onload = function() {
          context.drawImage(imageObj, x, y, width, height);
        };
        imageObj.src =
"http://www.html5canvastutorials.com/demos/assets/darth-vader.jpg";
```

```
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="400"></canvas>
  </body>
</html>
```

1.7.3 图像裁剪

方法 drawImage() 还可以增加另六个参数来实现对图像的裁剪。这六个参数是，sourceX，sourceY，sourceWidth，sourceHeight，destWidth 和 destHeight。这六个参数对应的含义可以参阅后面的示意图。

```
<script>
  context.drawImage(imageObj, sx, sy, sw, sh, dx, dy, dw, dh);
</script>
```



Reference: www.whatwg.org

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var imageObj = new Image();

        imageObj.onload = function() {
          // 绘制剪裁的图像
          var sourceX = 150;
          var sourceY = 0;
          var sourceWidth = 150;
          var sourceHeight = 150;
          var destWidth = sourceWidth;
          var destHeight = sourceHeight;
          var destX = canvas.width / 2 - destWidth / 2;
```

```
            var destY = canvas.height / 2 - destHeight / 2;

            context.drawImage(imageObj, sourceX, sourceY, sourceWidth,
sourceHeight, destX, destY, destWidth, destHeight);
          };
          imageObj.src =
"http://www.html5canvastutorials.com/demos/assets/darth-vader.jpg";
        };

      </script>
    </head>
    <body>
      <canvas id="myCanvas" width="578" height="400"></canvas>
    </body>
</html>
```

1.7.4 图像加载器

当在 canvas 中要使用多幅图像的时候，最好是在绘制图像之前就把图像全部加载。一个方便的办法是用一个图像加载函数一下在把图像全加载进图像对象中来，然后再调用一个用户定义的函数。

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
```

```
    <script>
      function loadImages(sources, callback) {
        var images = {};
        var loadedImages = 0;
        var numImages = 0;
        // 获取图像源的数量
        for(var src in sources) {
          numImages++;
        }
        for(var src in sources) {
          images[src] = new Image();
          images[src].onload = function() {
            if(++loadedImages >= numImages) {
              callback(images);
            }
          };
          images[src].src = sources[src];
        }
      }

      window.onload = function(images) {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        var sources = {
          darthVader:
"http://www.html5canvastutorials.com/demos/assets/darth-vader.jpg",
          yoda:
"http://www.html5canvastutorials.com/demos/assets/yoda.jpg"
        };

        loadImages(sources, function(images) {
          context.drawImage(images.darthVader, 100, 30, 200, 137);
          context.drawImage(images.yoda, 350, 55, 93, 104);
        });
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8 文本

1.8.1 文本的字体、大小和样式

要设置字体、大小和样式，需要用到上下文对象的 `font` 属性。样式可以是 `normal`, `italic` 或 `bold` 。默认情况是 `normal` 。

```
<script>
  context.font = 'italic 40px Calibri';
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.font = "italic 40pt Calibri";
        context.fillText("Hello World!", 150, 100);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8.2 文本颜色

文本的颜色用 `fillStyle` 属性设置。

```
<script>
  context.fillStyle = 'blue';
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.font = "40pt Calibri";
        context.fillStyle = "blue";
        context.fillText("Hello World!", 150, 100);
      };

    </script>
  </head>
```

```
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8.3 描绘文本边缘

要描绘字体边缘的效果，要使用 `strokeText()` 方法替代 `fillText()`，同时要用 `strokeStyle` 属性替代 `fillStyle` 属性。

如：

```
<script>
  context.strokeStyle = 'blue';
  context.lineWidth = 3;
  context.strokeText('Hello World!', x, y);
</script>
```

注意：如果要同时填充字体和描绘字体边缘，那么就必须同时使用 `fillText()` 和 `strokeText()` 方法。而且记得要先执行 `fillText()` ，然后再执行 `strokeText()` 。

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
```

```
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = 80;
        var y = 110;

        context.font = "60pt Calibri";
        context.lineWidth = 3;
        // 轮廓线的颜色
        context.strokeStyle = "blue";
        context.strokeText("Hello World!", x, y);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8.4 文本对齐

文本的对齐功能设定使用 `textAlign` 属性。其可用的选项包括 `start`, `end`, `left`, `center` 和 `right` 。对齐的位置是相对于一条虚拟的垂直线，这条线是由 `fillText()` 或 `strokeText()` 定义的文本 `x` 位置。

文本被左对齐的情况包括：

1. `textAlign` 属性被设为 `left` 时；
2. `textAlign` 属性被设为 `start` ，且文档方向是 `ltr (left to right)` 时；
3. `textAlign` 属性被设为 `end` ，且文档方向是 `rtl (right to left)` 时。

文本被右对齐的情况包括：

1. `textAlign` 属性被设为 `right` 时；
2. `textAlign` 属性被设为 `start` ，且文档方向是 `rtl (right to left)` 时；
3. `textAlign` 属性被设为 `end` ，且文档方向是 `ltr (left to right)` 时。

默认情况下 `textAlign` 属性被设为 `start` 。

如：

```
<script>
  context.textAlign = 'center';
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = canvas.width / 2;
        var y = canvas.height / 2;

        context.font = "30pt Calibri";
        context.textAlign = "center";
        context.fillStyle = "blue";
        context.fillText("Hello World!", x, y);
      };
```

```
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8.5 文本基线

垂直对齐文本需要用到 `textBaseline` 属性。此属性可用的选项包括： `top`, `hanging`, `middle`, `alphabetic`, `ideographic` 和 `bottom` 。默认情况下是 `alphabetic` 。

如：

```
<script>
  context.textBaseline = 'middle';
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
```

```
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = canvas.width / 2;
        var y = canvas.height / 2;

        context.font = "30pt Calibri";
        // textAlign 根据文本的放置进行水平的对齐
        context.textAlign = "center";
        // textBaseline 根据字体样式垂直对齐字体
        context.textBaseline = "middle";
        context.fillStyle = "blue";
        context.fillText("Hello World!", x, y);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

1.8.6 文本度量

要获取有关文本的尺度信息，可以使用 `measureText()` 方法。此方法需要一个文本字符串组为参数，并返回一个尺度对象。尺度对象中的数据是基于所提供的字符串参数和当前的文本字体信息而来的。

如：

```
<script>
  var metrics = context.measureText('measure me!');
  var width = metrics.width;
</script>
```

注意：由于文本的像素高度等于字体尺寸的磅值，所以，从 `measureText()` 返回的尺度对象并不包含高度数据。

效果图

代码

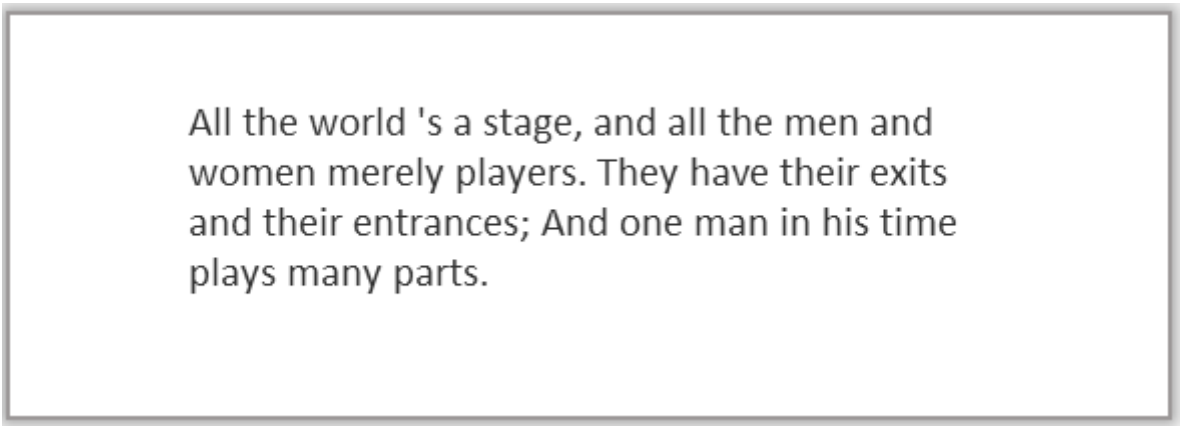
```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = canvas.width / 2;
        var y = canvas.height / 2 - 10;
        var text = "Hello World!";

        context.font = "30pt Calibri";
        context.textAlign = "center";
        context.fillStyle = "blue";
        context.fillText(text, x, y);

        // 获得文本的尺寸
        var metrics = context.measureText(text);
        var width = metrics.width;
        context.font = "20pt Calibri";
        context.textAlign = "center";
        context.fillStyle = "#555";
        context.fillText("(" + width + "px wide)", x, y + 40);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
```

```
</html>
```

1.8.7 文本换行

要实现文本换行功能，我们需要创建一个用户自定义函数，此函数需要 `canvas` 上下文对象、一个文本字符串、一个位置、一个最大宽度和行高度信息。函数将使用 `measureText()` 计算何时换行。

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      function wrapText(context, text, x, y, maxWidth, lineHeight) {
        var words = text.split(" ");
        var line = "";

        for(var n = 0; n < words.length; n++) {
          var testLine = line + words[n] + " ";
          var metrics = context.measureText(testLine);
          var testWidth = metrics.width;
          if(testWidth > maxWidth) {
            context.fillText(line, x, y);
```

```
            line = words[n] + " ";
            y += lineHeight;
          }
          else {
            line = testLine;
          }
        }
        context.fillText(line, x, y);
      }

      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var maxWidth = 400;
        var lineHeight = 25;
        var x = (canvas.width - maxWidth) / 2;
        var y = 60;
        var text = "All the world 's a stage, and all the men and women
merely players. They have their exits and their entrances; And one man in
his time plays many parts.";

        context.font = "16pt Calibri";
        context.fillStyle = "#333";

        wrapText(context, text, x, y, maxWidth, lineHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

Part 2: HTML5 Canvas 教程进阶篇

欢迎来到进阶篇，在此我们将学习有关复合、转换、图像数据与 URLs、动画和鼠标检测方面的内容。

学前准备：

在开始基础教程之前，您首先需要准备一个不太旧的 web 浏览器，比如 Google Chrome ， Firefox， Safari， Opera， 或者 IE9 这些都可以。然后您需要对 Javascript 有一定的熟悉，并且还得有一个文本编辑器，比如 notepad。如果您刚刚开始接触 HTML5 Canvas，那么您最好还是先看一下本教程的基础篇部分，然后再回到本篇。

2.1 组合

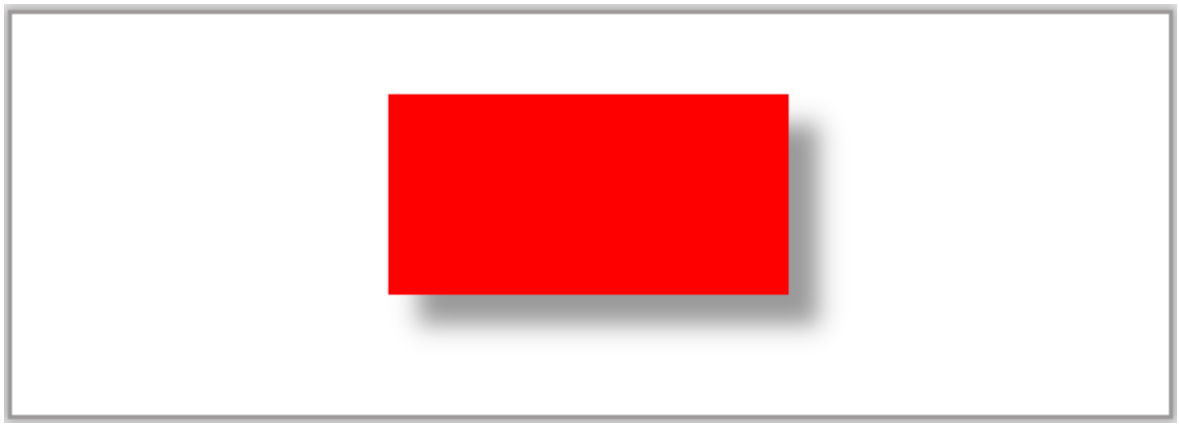
2.1.1 阴影

要为图形添加阴影需要用到 `shadowColor`，`shadowBlur`，`shadowOffsetX` 和 `shadowOffsetY` 属性。

如：

```
<script>
  context.shadowColor = "black";
  context.shadowBlur = 20;
  context.shadowOffsetX = 10;
  context.shadowOffsetY = 10;
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
```

```
        context.rect(188, 40, 200, 100);
        context.fillStyle = "red";
        context.shadowColor = "#999";
        context.shadowBlur = 20;
        context.shadowOffsetX = 15;
        context.shadowOffsetY = 15;
        context.fill();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

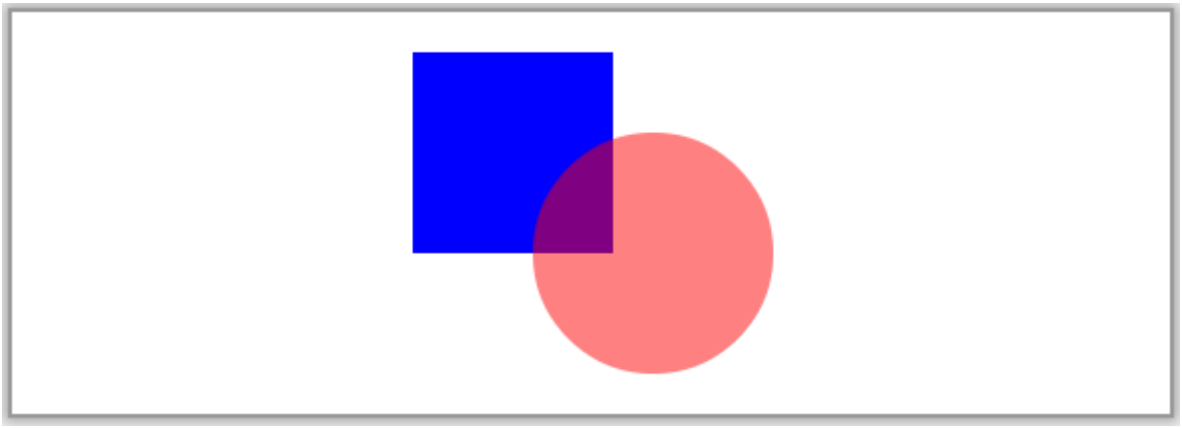
2.1.2 透明

设置图形的透明度要用到 `globalAlpha` 属性。 `globalAlpha` 属性的值是一个介于 `0` 到 `1` 之间的浮点数。 `0` 表示完全透明，而 `1` 表示完全不透明。

如：

```
<script>
  context.globalAlpha = 0.5;
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // draw blue rectangle
        context.beginPath();
        context.rect(200, 20, 100, 100);
        context.fillStyle = "blue";
        context.fill();

        // draw transparent red circle
        context.globalAlpha = 0.5;
        context.beginPath();
        context.arc(320, 120, 60, 0, 2 * Math.PI, false);
        context.fillStyle = "red";
        context.fill();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

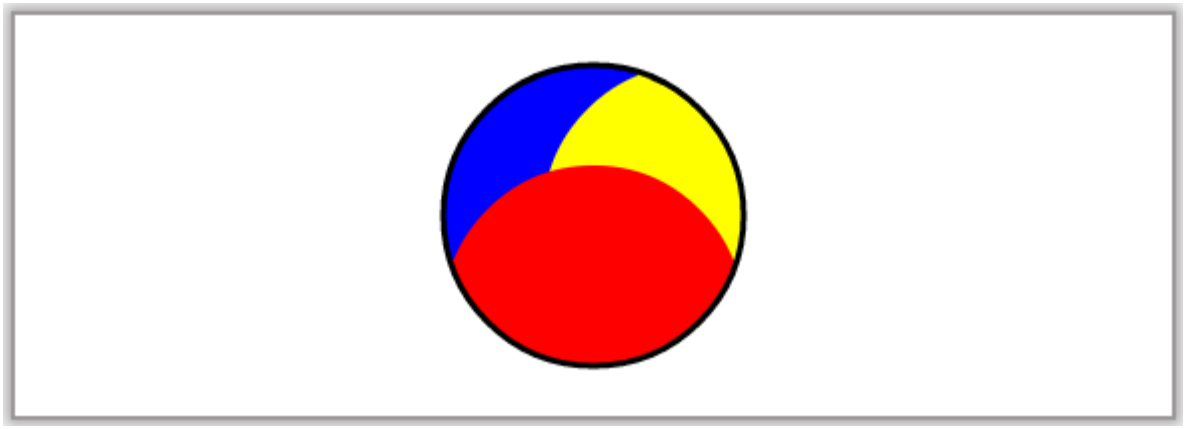
2.1.3 裁剪区

创建裁剪区的方法是先画一个路径，然后用 `clip()` 方法。

如：

```
<script>
  context.clip();
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var x = canvas.width / 2;
        var y = canvas.height / 2;
        var radius = 75;
        var offset = 50;

        /*
         * save() 的作用是在创建剪裁区之前将 canvas 的当前状态保存起来，
         * 这样在后面就可以恢复上下文对象的原始状态了
         */
        context.save();
        context.beginPath();
        context.arc(x, y, radius, 0, 2 * Math.PI, false);
        context.clip();

        // 在剪裁区内部画一个蓝色的源圆圈
        context.beginPath();
        context.arc(x - offset, y - offset, radius, 0, 2 * Math.PI,
false);
        context.fillStyle = "blue";
        context.fill();

        // 在剪裁区内部画一个黄色的源圆圈
        context.beginPath();
```

```
        context.arc(x + offset, y, radius, 0, 2 * Math.PI, false);
        context.fillStyle = "yellow";
        context.fill();

        // 在剪裁区内部画一个红色的源圆圈
        context.beginPath();
        context.arc(x, y + offset, radius, 0, 2 * Math.PI, false);
        context.fillStyle = "red";
        context.fill();

        /*
         * restore() 方法使 canvas 上下文对象恢复到创建剪裁区之前的状态
         */
        context.restore();
        context.beginPath();
        context.arc(x, y, radius, 0, 2 * Math.PI, false);
        context.lineWidth = 3;
        context.strokeStyle= "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.1.4 全局组合操作

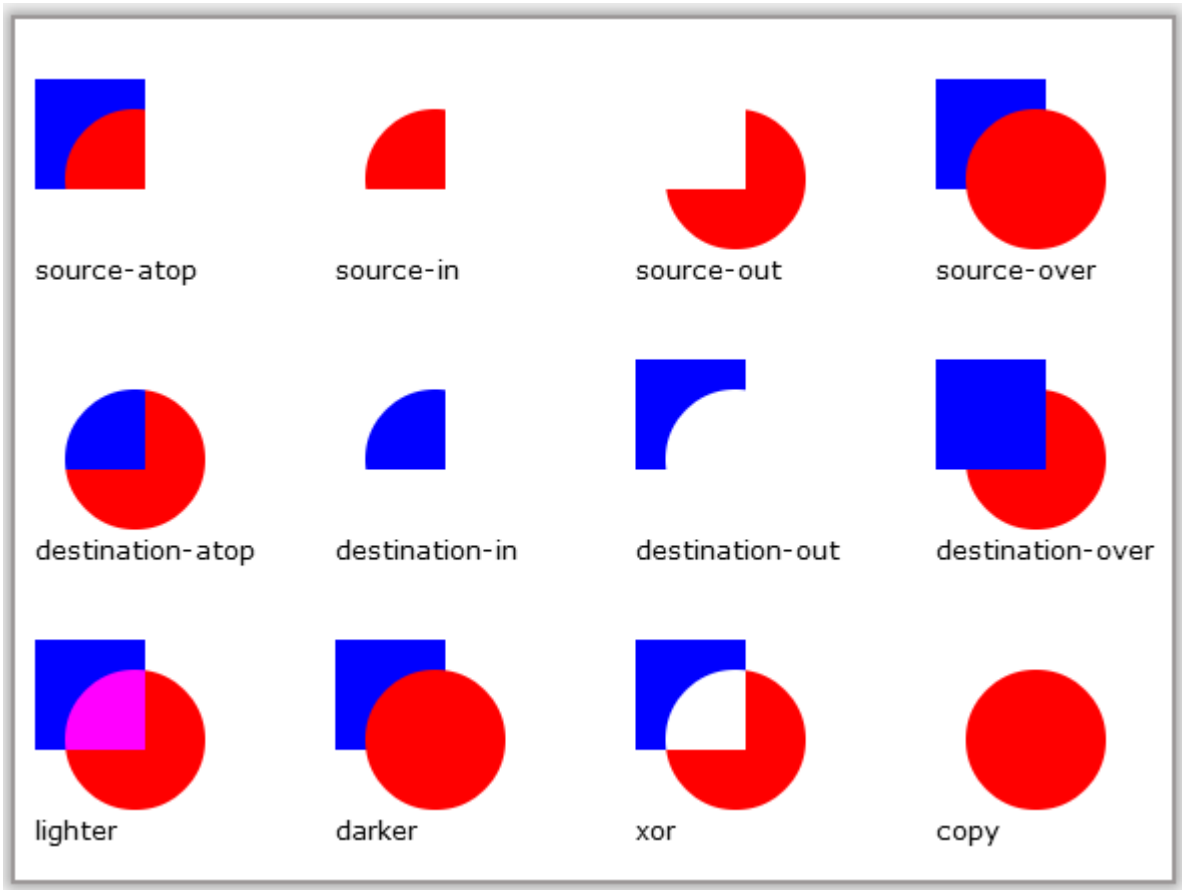
上下文对象的 globalCompositeOperation 属性定义了组合操作的方式，也就是 canvas 的源与目的的状态。目的是组合操作之前的状态，而源指的是组合操作后面的状态。（***译者注：****理解起来比较别扭是吧，看后面的效果示意图就比较好懂了*）

共有 12 种组合操作可供我们选择使用，其中包括： source-atop , source-in , source-out , source-over , destination-atop , destination-in , destination-out , destination-over , lighter , xor , 和 copy 。默认状态下是 source-over 。

如：

```
<script>
  context.globalCompositeOperation = 'destination-over';
</script>
```

各种组合操作的效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        // 注意这里创建了一个临时canvas，可以理解为内存中绘图所用，用于在正式将图形画到页面之前先把完整的图形在这个临时canvas中画完，然后再一下子拷贝到真正用于显示的myCanvas上，而在页面中的这个临时canvas是不可见的
        var tempCanvas = document.getElementById("tempCanvas");
        var tempContext = tempCanvas.getContext("2d");
        var squareWidth = 55;
        var circleRadius = 35;
        var startX = 10;
        var startY = 30;
```

```
var rectCircleDistX = 50;
var rectCircleDistY = 50;
var exampleDistX = 150;
var exampleDistY = 140;

var arr = new Array();
arr.push("source-atop");
arr.push("source-in");
arr.push("source-out");
arr.push("source-over");
arr.push("destination-atop");
arr.push("destination-in");
arr.push("destination-out");
arr.push("destination-over");
arr.push("lighter");
arr.push("darker");
arr.push("xor");
arr.push("copy");

// 画出十种操作模式
for(var n = 0; n < arr.length; n++) {
  var thisX;
  var thisY;

  var thisOperation = arr[n];

  // 第一行
  if(n < 4) {
    thisX = startX + (n * exampleDistX);
    thisY = startY;
  }

  // 第二行
  else if(n < 8) {
    thisX = startX + ((n - 4) * exampleDistX);
    thisY = startY + exampleDistY;
  }

  // 第三行
  else {
    thisX = startX + ((n - 8) * exampleDistX);
    thisY = startY + (exampleDistY * 2);
  }

  tempContext.clearRect(0, 0, canvas.width, canvas.height);

  // 画矩形
  tempContext.beginPath();
  tempContext.rect(thisX, thisY, squareWidth, squareWidth);
  tempContext.fillStyle = "blue";
  tempContext.fill();

  // 设置全局组合模式
  tempContext.globalCompositeOperation = thisOperation;

  // 画圆
  tempContext.beginPath();
  tempContext.arc(thisX + rectCircleDistX, thisY +
```

```
rectCircleDistY, circleRadius, 0, 2 * Math.PI, false);
            tempContext.fillStyle = "red";
            tempContext.fill();

            // 恢复成默认状态
            tempContext.globalCompositeOperation = "source-over";
            tempContext.font = "10pt Verdana";
            tempContext.fillStyle = "black";
            tempContext.fillText(thisOperation, thisX, thisY + squareWidth
+ 45);

            // 将图像从 tempCanvas 拷贝到 myCanvas
            context.drawImage(tempCanvas, 0, 0);
          }
        }
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="430"></canvas>
    <!-- 下面这个canvas就是用作内存中绘图的，样式被设为不可见 -->
    <canvas id="tempCanvas" width="578" height="430"
style="display:none;"></canvas>
  </body>
</html>
```

2.2 坐标转换

2.2.1 原点的位移

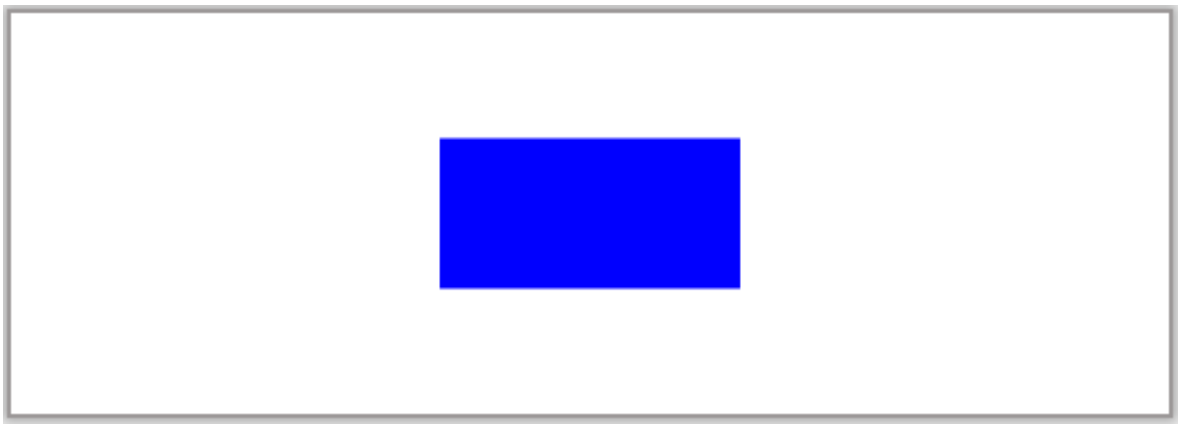
使用 `translate()` 方法可以将绘图原点向横向和纵向移动指定的距离`(x, y)`，结果表现为整张图像的移动。

如：

```
<script>
  context.translate(x, y);
</script>
```

效果图，translate 方法被注释掉的效果

效果图，translate 方法没有注释掉的效果



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 把坐标原点移动到canvas中心点
        // 如果本行被注释掉，结果就是上面第一张图，否则就是第二张图的效果
        context.translate(canvas.width / 2, canvas.height / 2);
```

```
        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

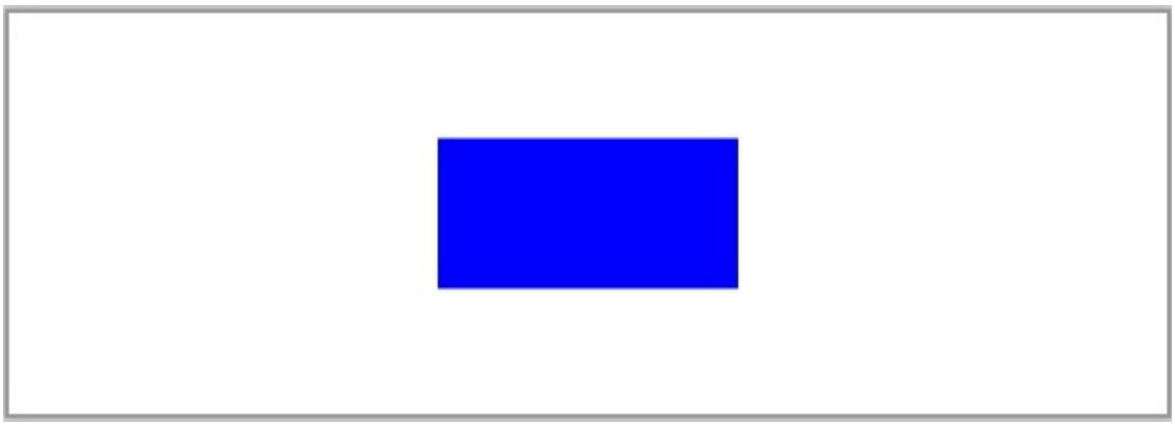
2.2.2 缩放

缩放操作使用 `scale()` 方法，参数 `x`、`y` 分别代表横向与纵向的缩放比例，两个参数都是浮点数，`1.0` 表示不缩放，小于 `1.0` 表示缩小，大于 `1.0` 表示放大。即，元坐标 `(x1, y1)` 上的点经 `scale(x, y)` 缩放后的点将被移动到 `(x1*x, y1*y)`。

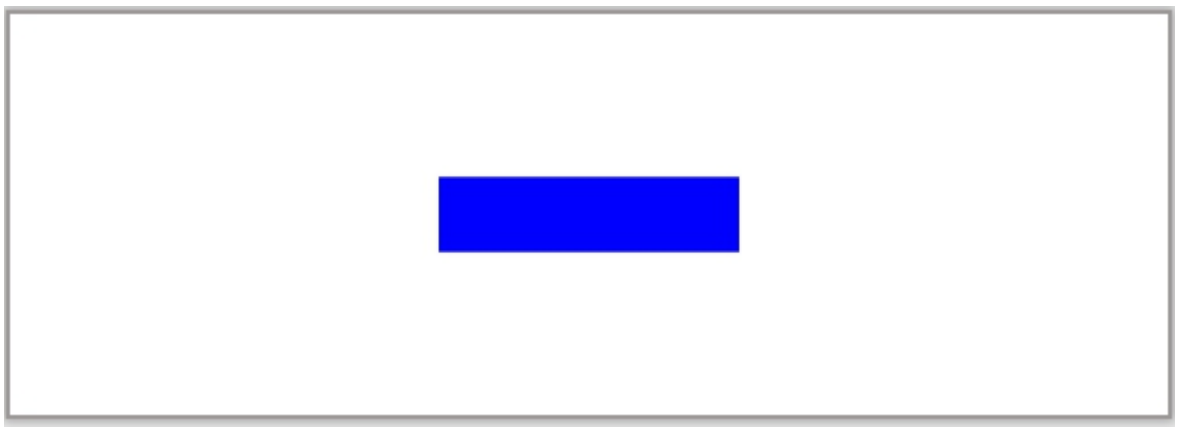
如：

```
<script>
  context.scale(x, y);
</script>
```

效果图，scale 方法被注释掉的效果



效果图，scale 方法没有被注释掉的效果



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 把坐标原点移动到canvas中心点
        context.translate(canvas.width / 2, canvas.height / 2);

        // 坐标在横向上缩小一半
        context.scale(1, 0.5);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

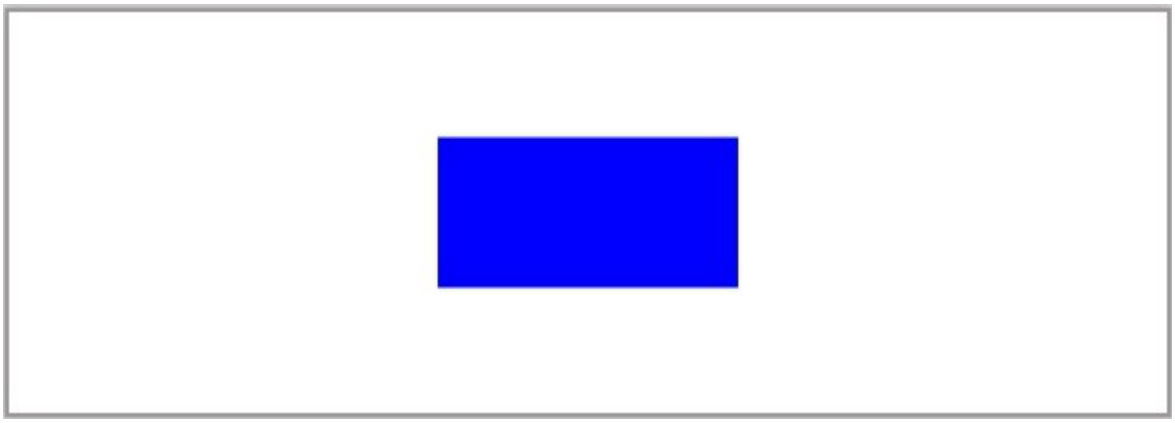
2.2.3 旋转

旋转 canvas 用的方法是 rotate() 。此方法接受一个以弧度为单位的旋转参数，整个 canvas 将以坐标原点，也就是由 translate() 所确定的原点为圆心进行旋转。.在本教程中，我们将原点移动到了 canvas 的中心点上，这样图中的矩形就可以以其中心点进行旋转了。

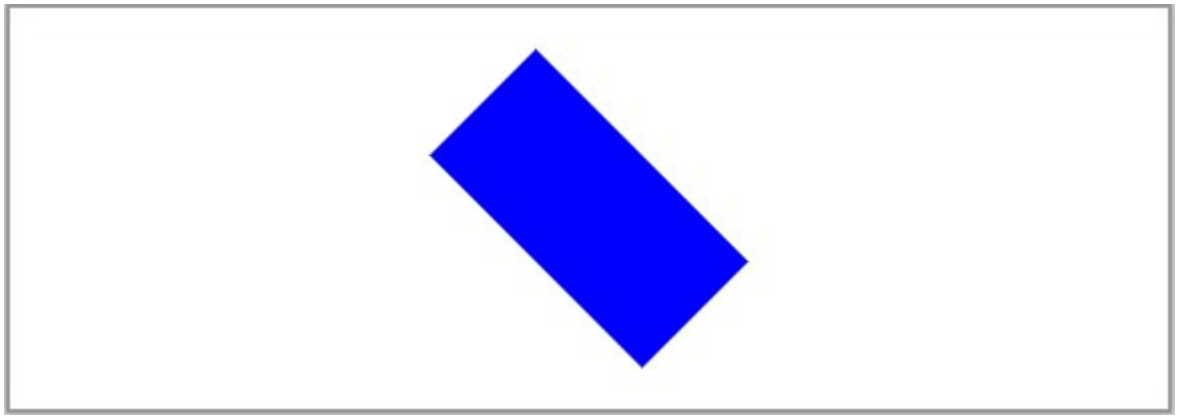
如：

```
<script>
  context.rotate(angle);
</script>
```

效果图，scale 被注释掉的效果



效果图，scale 没有注释掉的效果



代码

```
<!DOCTYPE HTML>
```

```
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 把坐标原点移动到canvas中心点
        context.translate(canvas.width / 2, canvas.height / 2);

        // 顺时针旋转45度
        context.rotate(Math.PI / 4);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.2.4 用户自定义坐标变换

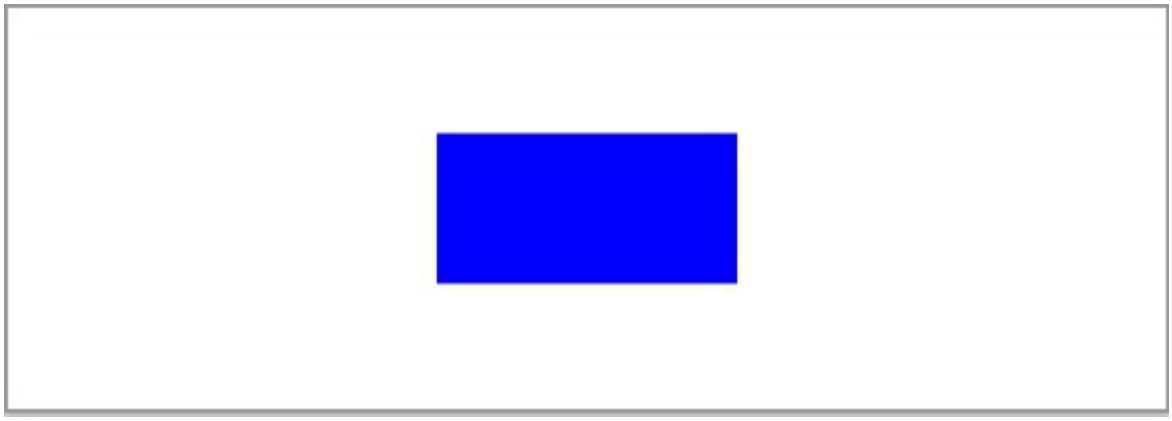
HTML5 可以使用 transform() 方法以用户自定义的变换矩阵对图像坐标进行变换操作。这个方法需要 6 个参数组成一个 3 x 3 的转换矩阵，坐标的由 (x, y) 到 (x', y') 的转换公式如下所示：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
<script>
  context.transform(a, b, c, d, e, f);
```

```
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 转换矩阵
        //  1  0  tx
        //  0  1  ty
        //  0  0  1
        var tx = canvas.width / 2;
        var ty = canvas.height / 2;

        // 应用用户自定义转换
        context.transform(1, 0, 0, 1, tx, ty);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };
```

```
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.2.5 剪切变换

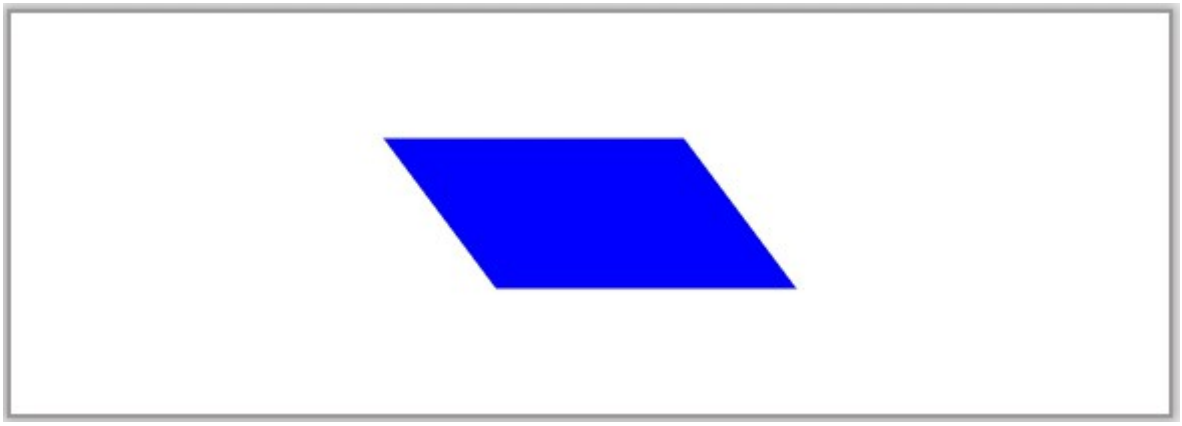
如果我们按照下面公式进行坐标变换的话，就可以对图像进行剪切变换。其中 `sx` 定义了水平方向的剪切， `sy` 则定义了垂直方向上的剪切。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_x & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

如：

```
<script>
  context.transform(1 ,sy, sx, 1, 0, 0);
</script>
```

效果图



代码

```
<!DOCTYPE HTML>
```

```
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 剪切矩阵
        //  1  sx  0
        //  sy  1  0
        //  0  0  1

        var sx = 0.75;
        // .75 水平剪切
        var sy = 0;
        // 无垂直剪切

        // 把坐标原点移动到中心点
        context.translate(canvas.width / 2, canvas.height / 2);

        // 应用用户定义转换
        context.transform(1, sy, sx, 1, 0, 0);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };
    </script>
      </head>
      <body>
        <canvas id="myCanvas" width="578" height="200"></canvas>
      </body>
    </html>
```

2.2.6 镜面转换

镜面转换只需要在 `scale()` 方法的参数中使用负值的参数，比如水平镜像就把 `x` 参数赋值一个负数，垂直镜像则是把 `y` 参数赋值一个负数。

如：

```
<script>
  // 水平镜像
  context.scale(-1,1);

  // 垂直镜像
  context.scale(1,-1);
</script>
```

效果图

代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 把坐标原点移动到canvas中心点
        context.translate(canvas.width / 2, canvas.height / 2);

        // 水平翻转图像
        context.scale(-1, 1);

        context.font = "30pt Calibri";
        context.textAlign = "center";
        context.fillStyle = "blue";
        context.fillText("Hello World!", 0, 0);
```

```
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.2.7 重置转换

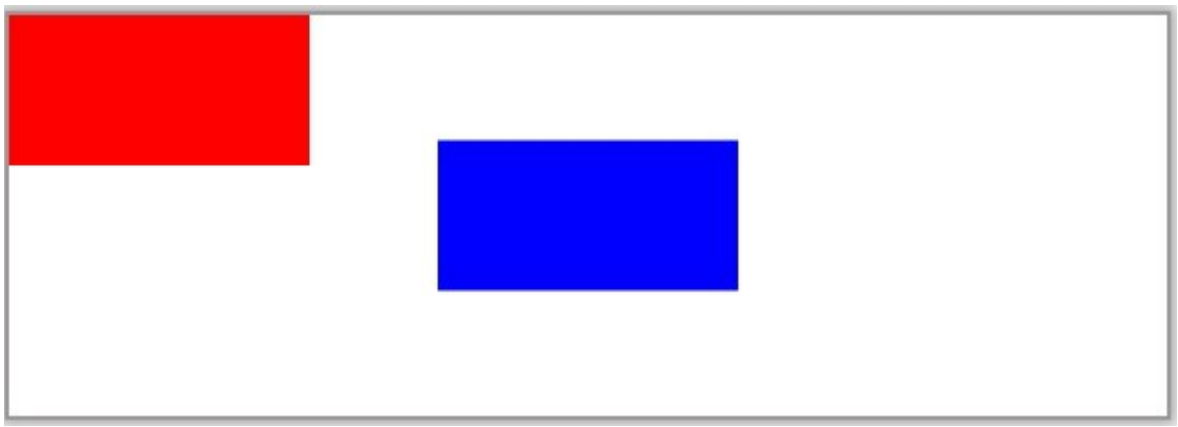
如果要将坐标转换为原始状态，需要使用下面的公式调用 `setTransform()` 方法。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

如：

```
<script>
  context.setTransform(1, 0, 0, 1, 0, 0);
</script>
```

效果图，红色为重置坐标到原始状态的效果



代码

```
<!DOCTYPE HTML>
```

```
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var rectWidth = 150;
        var rectHeight = 75;

        // 把坐标原点移动到canvas中心点
        context.translate(canvas.width / 2, canvas.height / 2);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);

        // 重置转换
        // 1 0 0
        // 0 1 0
        // 0 0 1

        // 应用用户自定义转换
        context.setTransform(1, 0, 0, 1, 0, 0);

        context.fillStyle = "red";
        context.fillRect(0, 0, rectWidth, rectHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.2.8 变换状态栈

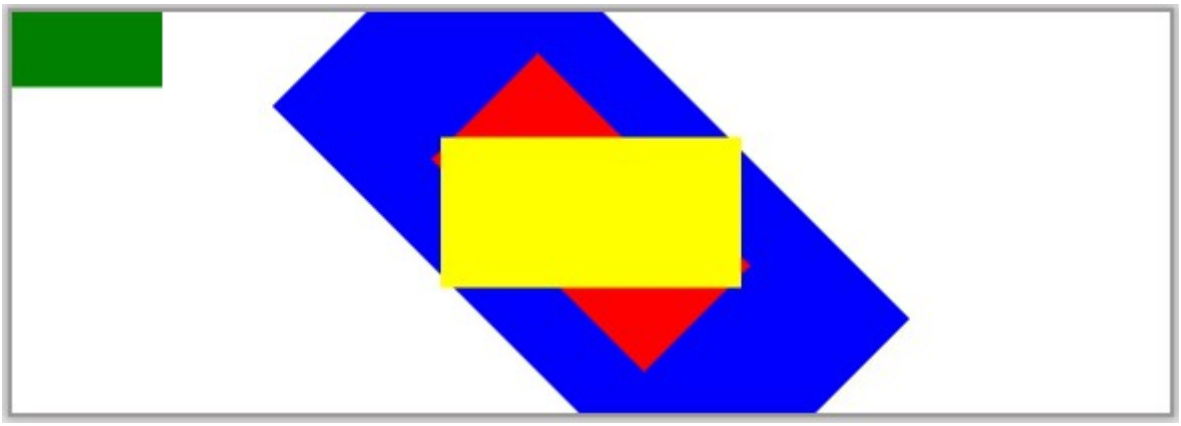
使用 `save()` 和 `restore()` 方法可以实现对坐标变换状态的保存与恢复。

在本教程中，我们在每次进行坐标转换前把当前的转换状态推进栈中。首先画一个蓝色的矩形，从状态栈中弹出恢复上一个转换状态，然后再画个红色的矩形，从状态栈中再弹出恢复上一个转换状态，再画个黄色的矩形，最后再从状态栈中再弹出恢复上最后一个转换状态，画一个绿色矩形。

如：

```
<script>
  context.save();
  context.restore();
</script>
```

效果图



代码

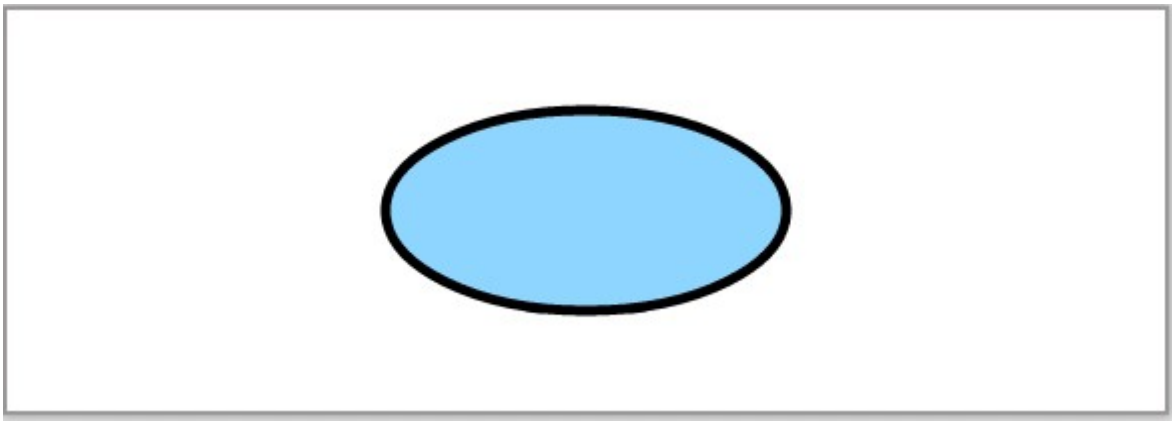
```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        var rectWidth = 150;
        var rectHeight = 75;

        context.save();
        // 保存状态 1
        context.translate(canvas.width / 2, canvas.height / 2);

        context.save();
        // 保存状态 2
```

```
        context.rotate(Math.PI / 4);

        context.save();
        // 保存状态 3
        context.scale(2, 2);

        context.fillStyle = "blue";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);

        context.restore();
                // 恢复状态 3
        context.fillStyle = "red";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);

        context.restore();
        // 恢复状态 2
        context.fillStyle = "yellow";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);

        context.restore();
        // 恢复状态 1
        context.fillStyle = "green";
        context.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.2.9 椭圆

要画一个椭圆，我们可以这样做，首先保存转换状态，然后将 canvas 水平拉伸，画一个圆，恢复转换状态，然后应用样式。

效果图



代码

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        var centerX = 0;
        var centerY = 0;
        var radius = 50;

        // 保存状态
        context.save();

        // 移动原点
        context.translate(canvas.width / 2, canvas.height / 2);

        // 水平缩放
        //context.scale(2, 1);

        // 这里画的圆会被拉伸成椭圆
        context.beginPath();
        context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);

        // 恢复原始状态
        context.restore();

        // 应用样式
        context.fillStyle = "#8ED6FF";
        context.fill();
```

```
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
      };

    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

2.3 图像数据与 URLs

2.3.1 图像数据

我们在 HTML5 的 canvas 中可以使用 getImageData() 方法和图像对象的属性来获取图像中每一个像素的数据。图像数据中的每个像素都由 red，green，blue，和 alpha 值来表示。我们就可以用 putImageData() 方法来设置图像的像素值，然后重画经过修改的图像。

如：

```
<script>
  context.drawImage(imageObj, destX, destY);
  var imageData = context.getImageData(0, 0, canvas.width,
canvas.height);
  var data = imageData.data;
  // 处理像素数据
  context.putImageData(imageData, 0, 0);
</script>
```

注意：方法 getImageData() 要求图像数据存储在 web 服务器上，并且操作这个图像的代码必须是在同一台服务器上，如果这两条中的任一条不满足的话，将会抛出 SECURITY_ERR 异常。

代码

```
window.onload = function(){
    var imageObj = new Image();
    imageObj.onload = function(){
        drawImage(this);
    };
    imageObj.src = "darth-vader.jpg";
};

function drawImage(imageObj){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var destX = 69;
    var destY = 50;
    var sourceWidth = imageObj.width;
    var sourceHeight = imageObj.height;

    context.drawImage(imageObj, destX, destY);
    var imageData = context.getImageData(0, 0, canvas.width,
canvas.height);
    var data = imageData.data;

    // 快速遍历所有的像素
    for (var i = 0; i < data.length; i += 4) {
        var red = data[i]; // red
        var green = data[i + 1]; // green
        var blue = data[i + 2]; // blue
        // i+3 就是 alpha 值
    }

    // 另一种遍历像素的方式
    for (var y = 0; y < sourceHeight; y++) {
        // 遍历每一列
        for (var x = 0; x < sourceWidth; x++) {
            var red = data[((sourceWidth * y) + x) * 4];
            var green = data[((sourceWidth * y) + x) * 4 + 1];
            var blue = data[((sourceWidth * y) + x) * 4 + 2];
        }
    }
    // 如果修改了像素数据，就用下面这行代码把图像数据再写回canvas显示
    context.putImageData(imageData, 0, 0);
}
```

2.3.2 反转颜色

如果要反转图像上每个像素的颜色，我们可以用 255 去减每个像素的 red、 green、 blue 的值，然后再写回 canvas 中去。

注意：方法 getImageData() 要求图像数据存储在 web 服务器上，并且操作这个图像的代码必须是在同一台服务器上，如果这两条中的任一条不满足的话，将会抛出 SECURITY_ERR 异常。

效果图

代码

```
window.onload = function(){
    var imageObj = new Image();
    imageObj.onload = function(){
        drawImage(this);
    };
    imageObj.src = "darth-vader.jpg";
};

function drawImage(imageObj){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var destX = 69;
    var destY = 50;

    context.drawImage(imageObj, destX, destY);

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

    for (var i = 0; i < data.length; i += 4) {
        data[i] = 255 - data[i]; // red
        data[i + 1] = 255 - data[i + 1]; // green
        data[i + 2] = 255 - data[i + 2]; // blue
        // i+3 是 alpha值
    }
```

```
    // 覆盖原数据
    context.putImageData(imageData, 0, 0);
}
```

2.3.3 灰度图

要把一个图像转换为灰度图，我们只需要遍历图像中的每个像素，计算像素的亮度值，然后把像素的 `red`、 `green`、 `blue` 值都设为亮度值就可以了。

效果图

代码

```
window.onload = function(){
    var imageObj = new Image();
    imageObj.onload = function(){
        drawImage(this);
    };
    imageObj.src = "darth-vader.jpg";
};

function drawImage(imageObj){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
```

```
    var destX = 69;
    var destY = 50;

    context.drawImage(imageObj, destX, destY);

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

    for (var i = 0; i < data.length; i += 4) {
        var brightness = 0.34 * data[i] + 0.5 * data[i + 1] + 0.16 *
data[i + 2];

        data[i] = brightness; // red
        data[i + 1] = brightness; // green
        data[i + 2] = brightness; // blue
        // i+3 是 alpha 值
    }

    // 覆盖原数据
    context.putImageData(imageData, 0, 0);
}
```

2.3.4 获得图像数据 URL

通过 toDataURL() 方法，我们可以得到一个指向当前图像的 64 bit PNG 格式图像文件的 URL.

如：

```
<script>
  canvas.toDataURL();
</script>
```

注意：如果 canvas 包含一个来自其他主机的图像，那么这个 canvas 被称为“脏”的。如果在这个“脏”的 canvas 上调用 toDataURL() 方法的话，将抛出 SECURITY_ERR 异常。

代码

```
window.onload = function(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
```

```
    // 画云
    context.beginPath(); // 开始用户定义图形
    context.moveTo(170, 80);
    context.bezierCurveTo(130, 100, 130, 150, 230, 150);
    context.bezierCurveTo(250, 180, 320, 180, 340, 150);
    context.bezierCurveTo(420, 150, 420, 120, 390, 100);
    context.bezierCurveTo(430, 40, 370, 30, 340, 50);
    context.bezierCurveTo(320, 5, 250, 20, 250, 50);
    context.bezierCurveTo(200, 5, 150, 20, 170, 80);
    context.closePath(); // 完成图形
    context.lineWidth = 5;
    context.fillStyle = "#8ED6FF";
    context.fill();
    context.strokeStyle = "#0000ff";
    context.stroke();

    // 将canvas图像保存为url（默认是PNG格式）
    var dataURL = canvas.toDataURL();
};
```

2.3.5 加载图像数据 URL

如果已知一个图像的数据 URL ，我们可以用 drawImage() 方法来把图像输出到 canvas 上。

在这里我们用 ajax 技术获取图像数据的 url，由此 url 创建一个图像对象，然后用 drawImage() 输出图像。

代码

```
function loadCanvas(dataURL){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    // 从图像url加载图像数据
    var imageObj = new Image();
    imageObj.onload = function(){
        context.drawImage(this, 0, 0);
    };

    imageObj.src = dataURL;
}

window.onload = function(){
    // 使用 ajax 获取图像数据 url
    var request = new XMLHttpRequest();
    request.open("GET", "dataURL.txt", true);
    request.onreadystatechange = function(){
        if (request.readyState == 4) { // 确保文档已经做好解析准备
```

```
            if (request.status == 200) { // 确保能够获取文件
                loadCanvas(request.responseText);
            }
        }
    };
    request.send(null);
};
```

2.3.6 将绘制的图形保存为图像

如果想将在 canvas 中绘制的图形保存为图像文件，我们只需将图像的数据 url 赋值给 HTML 页面上的 img 标签的 src 属性，用户就可以通过右键保存的形式把图形保存到本地机器上了。

注意：如果 canvas 包含一个来自其他主机的图像，那么这个 canvas 被称为“脏”的。如果在这个“脏”的 canvas 上调用 toDataURL() 方法的话，将抛出 SECURITY_ERR 异常。

代码

```
<head>
    <script>
        window.onload = function(){
            var canvas = document.getElementById("myCanvas");
            var context = canvas.getContext("2d");

            // 画云
            context.beginPath(); // 开始用户定义图形
            context.moveTo(170, 80);
            context.bezierCurveTo(130, 100, 130, 150, 230, 150);
            context.bezierCurveTo(250, 180, 320, 180, 340, 150);
            context.bezierCurveTo(420, 150, 420, 120, 390, 100);
            context.bezierCurveTo(430, 40, 370, 30, 340, 50);
            context.bezierCurveTo(320, 5, 250, 20, 250, 50);
            context.bezierCurveTo(200, 5, 150, 20, 170, 80);
            context.closePath(); // 完成用户定义图形
            context.lineWidth = 5;
            context.fillStyle = "#8ED6FF";
            context.fill();
            context.strokeStyle = "#0000ff";
            context.stroke();

            // 将canvas图像保存为数据url
            var dataURL = canvas.toDataURL();

            // 将数据 url 作为 canvasImg 的图像源
            // 这样就可以保存到客户机器上了
            document.getElementById("canvasImg").src = dataURL;
        };
    </script>
```

```
</head>
<body onmousedown="return false;">
    <canvas id="myCanvas" width="578" height="200" style="display:none;">
    </canvas>
    <img id="canvasImg" alt="Right click to save me!">
</body>
```

2.4 动画

译者注：*动画功能部分不属于 HTML5 Canvas API 的一部分，但是我们可以用一些方法实现动画的功能。*

2.4.1 清除 canvas 上的内容

要清除 canvas 上的内容，只需使用 clearRect() 方法清除整个 canvas 区域就行了。

如：

```
<script>
  context.clearRect(0,0,canvas.width,canvas.height);
</script>
```

代码

```
function clearCanvas(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.clearRect(0, 0, canvas.width, canvas.height);
}
```

2.4.2 使用 requestAnimationFrame 方法创建动画

如果要用 HTML5 的 canvas 创建动画，那么就需要用到 requestAnimationFrame 方法，这个方法能够使浏览器智能的判断帧率 FPS ，而且对于动画的每一帧我们都可以进行更新、清除 canvas 、重画 canvas，然后申请下一帧动画。

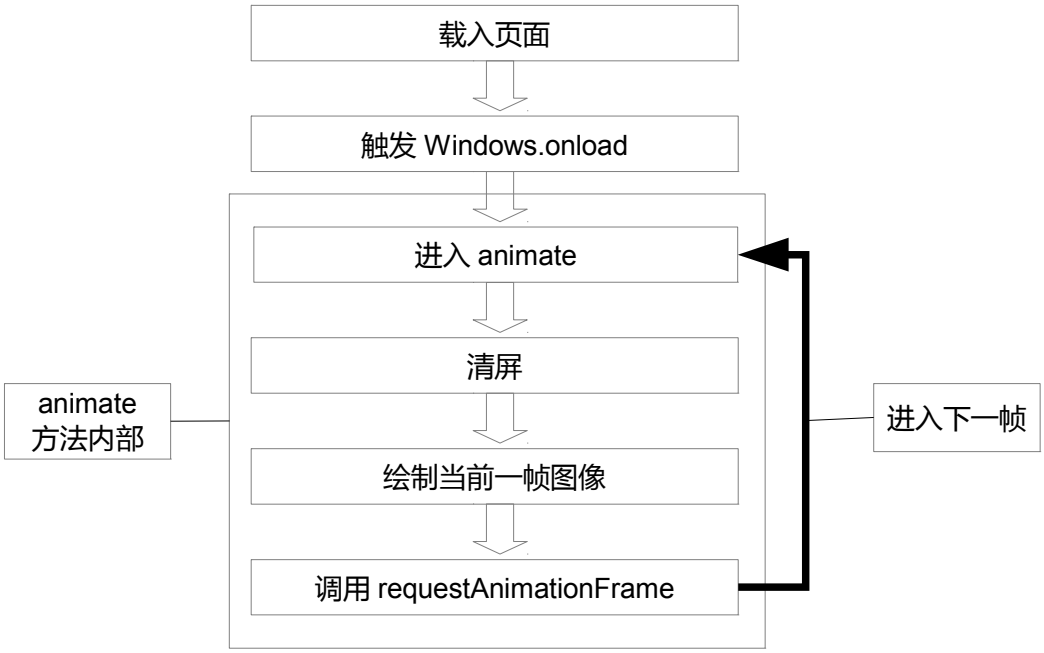
译者注：*在这里，动画的帧率跟你的机器硬件、浏览器软件的具体情况有一定关系，一般来说流畅的动画帧率在 40FPS 到 60FPS 之间。那么，对于程序员来说，很难判断应该针对每一个具体的用户使用多大的帧率，因此，各大浏览器都为我们提供了一个 windows.requestAnimationFrame 方法来*

自动的判断具体的帧率，这就给程序员省去了很多麻烦。

这个 `requestAnimationFrame` 方法接受一个由用户自定义的回调函数对象作为参数，`requestAnimationFrame` 将在当前帧完成后自动调用这个回调函数，而我们要做的就是在这个回调函数中进行我们的下一帧图像绘制操作，并在此回调函数最后再次调用 `requestAnimationFrame` 方法，使动画一帧帧的连续绘制下去。

比如说，我们的绘图方法叫做 `animate`，将此方法注册为 `windows.onload` 方法，每次页面调用就启动此方法，每次调用 `animate` 就在 `canvas` 绘制当前时间应该显示的那一帧图像，并在 `animate` 结尾返回之前，以其自身，也就是以 `animate` 为回调函数参数调用 `requestAnimationFrame`，由 `requestAnimationFrame` 自动再次调用 `animate` 绘制下一帧图像。

这个基本的流程可以参见下图：



不过，由于不同的浏览器对 `requestAnimationFrame` 的支持程度不同，所以，是使用的时候一般要稍作调整，通过对 `requestAnimationFrame` 的再封装提高程序对不同浏览器的兼容度。具体可以声明一个如下的 `requestAnimFrame` 方法来替代 `requestAnimationFrame`（注意名字不同）：

```
window.requestAnimFrame = (function(callback){
    return window.requestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
```

```
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60);
    };
})();
```

上面代码中的 webkitRequestAnimationFrame、 mozRequestAnimationFrame、 oRequestAnimationFrame 和 msRequestAnimationFrame 分别是兼容 Chrome、 Firefox、 Opera 和 IE。

这样，在 animate 的最后只要添加如下一行就可以了。

```
requestAnimFrame(function() { animate(); });
```

比较完整的实现方法参见下面的代码。至于 animate 中如何确定当前具体应该是哪一帧图像，可以参见下一节的内容。

代码

```
window.requestAnimFrame = (function(callback){
    return window.requestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60);
    };
})();

function animate(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    // 更新图像

    // 清屏
    context.clearRect(0, 0, canvas.width, canvas.height);

    // 绘制当前帧

    // 请求绘制下一帧
    requestAnimFrame(function(){
        animate();
    });
```

```
}

window.onload = function(){
    // 初始化

    animate();
};
```

2.4.3 线性运动

创建线性运动的动画，我们需要在每一帧里根据运动速度和方向调整运动对象的 (x, y) 坐标的值。其中的速度就是来自等式“距离=速度×时间”。

效果动画

http://www.html5canvastutorials.com/advanced/html5-canvas-linear-motion-animation/

译者注：本例中的 animate 带有两个参数，其中的 lastTime 指的是上一帧调用的时间，通过与当前时间的比较来确定当前帧中运动物体的位置。

代码

```
window.requestAnimFrame = (function(callback){
    return window.requestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60);
    };
})();

function animate(lastTime, myRectangle){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    // 更新图像
    var date = new Date();
    var time = date.getTime();
    var timeDiff = time - lastTime;
    var linearSpeed = 100; // 像素 / 秒
    var linearDistEachFrame = linearSpeed * timeDiff / 1000;
    var currentX = myRectangle.x;
```

```
    if (currentX < canvas.width - myRectangle.width -
myRectangle.borderWidth / 2) {
        var newX = currentX + linearDistEachFrame;
        myRectangle.x = newX;
    }
    lastTime = time;

    // 清屏
    context.clearRect(0, 0, canvas.width, canvas.height);

    // 绘图
    context.beginPath();
    context.rect(myRectangle.x, myRectangle.y, myRectangle.width,
myRectangle.height);

    context.fillStyle = "#8ED6FF";
    context.fill();
    context.lineWidth = myRectangle.borderWidth;
    context.strokeStyle = "black";
    context.stroke();

    // 请求绘制下一帧
    requestAnimFrame(function(){
        animate(lastTime, myRectangle);
    });
}

window.onload = function(){
    var myRectangle = {
        x: 0,
        y: 50,
        width: 100,
        height: 50,
        borderWidth: 5
    };

    var date = new Date();
    var time = date.getTime();
    animate(time, myRectangle);
};
```

2.4.4 加速运动

加速运动可以通过修改水平 `vx` 和垂直方向上 `vy` 的加速度来实现。

效果图

http://www.html5canvastutorials.com/advanced/html5-canvas-quadratic-motion-animation/

代码

```
window.requestAnimFrame = (function(callback){
    return window.requestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60);
    };
})();

function drawRectangle(myRectangle){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.beginPath();
    context.rect(myRectangle.x, myRectangle.y, myRectangle.width,
myRectangle.height);

    context.fillStyle = "#8ED6FF";
    context.fill();
    context.lineWidth = myRectangle.borderWidth;
    context.strokeStyle = "black";
    context.stroke();
}

function animate(lastTime, myRectangle){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    // 更新图像
    var date = new Date();
    var time = date.getTime();
    var timeDiff = time - lastTime;
    var gravity = 2; // 像素 / 秒^2
    var speedIncrementEachFrame = gravity * timeDiff / 1000; // 像素 / 秒
    myRectangle.vy += speedIncrementEachFrame;
    myRectangle.y += (myRectangle.vy * timeDiff);

    if (myRectangle.y > canvas.height - myRectangle.height -
myRectangle.borderWidth / 2) {
        myRectangle.y = canvas.height - myRectangle.height -
myRectangle.borderWidth / 2;
    }

    lastTime = time;

    // 清屏
    context.clearRect(0, 0, canvas.width, canvas.height);

    // 绘图
    drawRectangle(myRectangle);

    // 请求绘制下一帧
```

```
    requestAnimFrame(function(){
        animate(lastTime, myRectangle);
    });
}

window.onload = function(){
    var myRectangle = {
        x: 239,
        y: 0,
        vx: 0,
        vy: 0,
        width: 100,
        height: 50,
        borderWidth: 5
    };

    drawRectangle(myRectangle);

    // 矩形下坠前等待一分钟
    setTimeout(function(){
        var date = new Date();
        var time = date.getTime();
        animate(time, myRectangle);
    }, 1000);
};
```

2.4.5 震荡

震荡效果我们用这样一个公式来计算像素的坐标位置：x(时间) = 振幅 * sin(时间 * 2PI / 周期) + x0。（x0 为像素点的原始位置）

效果图

http://www.html5canvastutorials.com/advanced/html5-canvas-oscillation-animation/

代码

```
window.requestAnimFrame = (function(callback){
    return window.requestAnimationFrame ||
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60);
    };
})();
```

```
function animate(myRectangle){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    // 更新图像
    var date = new Date();
    var time = date.getTime();
    var amplitude = 150;
    var period = 2000; // 单位是毫秒
    var centerX = canvas.width / 2 - myRectangle.width / 2;
    var nextX = amplitude *
    Math.sin(time * 2 * Math.PI / period) +
    centerX;
    myRectangle.x = nextX;

    // 清屏
    context.clearRect(0, 0, canvas.width, canvas.height);

    // 绘图
    context.beginPath();
    context.rect(myRectangle.x, myRectangle.y, myRectangle.width,
myRectangle.height);
    context.fillStyle = "#8ED6FF";
    context.fill();
    context.lineWidth = myRectangle.borderWidth;
    context.strokeStyle = "black";
    context.stroke();

    // 请求绘制下一帧
    requestAnimFrame(function(){
        animate(myRectangle);
    });
}

window.onload = function(){
    var myRectangle = {
        x: 250,
        y: 70,
        width: 100,
        height: 50,
        borderWidth: 5
    };

    animate(myRectangle);
};
```

2.4.6 开始和停止动画

要开始动画，我们只需要调用一个不断请求下一帧的方法就可以了，而要停止已启动的动画，那么就只要不再继续请求下一帧就行了。

效果

http://www.html5canvastutorials.com/advanced/html5-canvas-start-and-stop-an-animation/

代码

```
    window.requestAnimFrame = (function(callback){
     return window.requestAnimationFrame ||
     window.webkitRequestAnimationFrame ||
     window.mozRequestAnimationFrame ||
     window.oRequestAnimationFrame ||
     window.msRequestAnimationFrame ||
     function(callback){
         window.setTimeout(callback, 1000 / 60);
     };
})();

function drawRect(myRectangle){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.rect(myRectangle.x, myRectangle.y, myRectangle.width,
myRectangle.height);

    context.fillStyle = "#8ED6FF";
    context.fill();
    context.lineWidth = myRectangle.borderWidth;
    context.strokeStyle = "black";
    context.stroke();
}

function animate(lastTime, myRectangle, animProp){
    if (animProp.animate) {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // 更新
        var date = new Date();
        var time = date.getTime();
        var timeDiff = time - lastTime;
        var linearSpeed = 100;
        // 像素 / 秒
        var linearDistEachFrame = linearSpeed * timeDiff / 1000;
        var currentX = myRectangle.x;

        if (currentX < canvas.width - myRectangle.width -
myRectangle.borderWidth / 2) {
            var newX = currentX + linearDistEachFrame;
            myRectangle.x = newX;
        }
        lastTime = time;

        // 清屏
        context.clearRect(0, 0, canvas.width, canvas.height);
```

```
        // 绘图
        drawRect(myRectangle);

        // 请求绘制下一帧
        requestAnimFrame(function(){
            animate(lastTime, myRectangle, animProp);
        });
    }
}

window.onload = function(){
    var myRectangle = {
        x: 0,
        y: 50,
        width: 100,
        height: 50,
        borderWidth: 5
    };

    /*
     * 这里用于判断是否停止的属性 animProp 需要被声明为一个对象
     * 如此才可以在被作为参数传递的时候是以引用传递
     * 这样其值才能被修改
     */
    var animProp = {
        animate: false
    };

    // 向canvas添加鼠标 click 事件的监听方法
    document.getElementById("myCanvas").addEventListener("click",
function(){
        if (animProp.animate) {
            animProp.animate = false;
        }
        else {
            animProp.animate = true;
            var date = new Date();
            var time = date.getTime();
            animate(time, myRectangle, animProp);
        }
    });

    drawRect(myRectangle);
};
```

2.5 鼠标

***译者注：**HTML5 并没有提供 canvas 的鼠标消息处理 API，但还是可以用 Javascript 的消息捕捉功能来实现对鼠标事件的监听与相应。*

2.5.1 鼠标坐标

为了获取鼠标在 canvas 中的相对位置，我们在这里创建了一个 getMousePos() 方法，在此方法中根据鼠标的位置、canvas 的位置以及页面的偏移量来计算出鼠标相对于 canvas 的位置。

效果

http://www.html5canvastutorials.com/advanced/html5-canvas-mouse-coordinates/

代码

```
function writeMessage(canvas, message){
    var context = canvas.getContext('2d');
    context.clearRect(0, 0, canvas.width, canvas.height);
    context.font = '18pt Calibri';
    context.fillStyle = 'black';
    context.fillText(message, 10, 25);
}

function getMousePos(canvas, evt){
    // 获得 canvas 位置
    var obj = canvas;
    var top = 0;
    var left = 0;
    while (obj && obj.tagName != 'BODY') {
        top += obj.offsetTop;
        left += obj.offsetLeft;
        obj = obj.offsetParent;
    }

    // 返回鼠标相对位置
    var mouseX = evt.clientX - left + window.pageXOffset;
    var mouseY = evt.clientY - top + window.pageYOffset;
    return {
        x: mouseX,
        y: mouseY
    };
}

window.onload = function(){
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');

    canvas.addEventListener('mousemove', function(evt){
        var mousePos = getMousePos(canvas, evt);
        var message = "Mouse position: " + mousePos.x + "," + mousePos.y;
```

```
        writeMessage(canvas, message);
    }, false);
};
```

ysm整理

最后更新：2012年6月16日